Keyboard Access Functional Specification, RC2

Earl Johnson
Chair
Sun Microsystems, Inc.

Bill Haneman
Sun Microsystems, Inc.

Mark Novak
University of Wisconsin, Madison

Willie Walker
Sun Microsystems, Inc.

Revision History

Revision RC2 25 October 2006 Revised by: wph
Incorporated comments and corrections from Olaf and Gunnar Schmidt and made corrections to some parameters.
Revision RC1 15 November 2005 Revised by: ej
Did a final readability review.
Revision 0.75 15 October 2005 Revised by: ej
Factored in input from FSGA. Added "visual bell" to section 7 of table 1, added a note identifying where a difference fr
Revision 0.7 21 June 2005 Revised by: ej
Made edits of suggested edits; split the test assertions out into a separate doc.
Revision 0.61 22 May 2005 Revised by: ej
Add more test assertions, includes table 2 assertions now
Revision 0.6 14 May 2005 Revised by: ej
Add more test assertions, reformat table one assertions
Revision 0.5 09 Mar 2005 Revised by: wdw
Add manual assertion examples for discussion
Revision 0.1 17 Feb 2005 Revised by: wdw
Convert to DocBook

Table of Contents

What This Specification Defines

This functional specification defines the minimum level of keyboard accessibility support and notification that must be provided to the user by an FSG Certified X Windowing System. This support ensures that users with a variety of physical disabilities will always have a basic level of access to the windowing system's core functions - keyboard input and controlling the system pointer (where available).

The features in this specification are largely dependent on support provided at the operating system level of the platform. This functional specification covers the 3 areas described below:

- "Configuration and Setting Requirements" enumerates the features and associated controls, configurability, and minimum ranges provided to the user.
- ""End-User Notification, Keyboard Invocation, and Pointer Emulation Requirements" enumerates the additional notification and controllability a platform must support in the user interface developed to meet the above section's requirements.
- "Feature Behavior Requirements" describes the type of behavior that must occur when the functionality of a keyboard access feature is exercised.

Limitations

The features and range of support prescribed in this document should not be taken as state-of-the-art, nor do they constitute the current best available practice.

Compliance with this specification is necessary, but not sufficient, for supporting adequate end-user keyboard access to the desktop environment. Meaningful end-user access to the desktop environment, and the applications hosted thereon requires that those applications are designed with keyboard access in mind; conformance of an operating environment to this specification does not imply that such applications are themselves accessible.

Features

The features defined in this specification provide the user with the following type of support:

- *StickyKeys*: Enables the user to keep the Control, Shift, Alt, or other modifier keys temporarily latched or locked while other keys are pressed in a sequential rather than a simultaneous fashion. This enables the user to correctly type capital letters, Control-C, Alt-M, Alt-Control-Backspace, and much more. Target users are people who can't press more than one key at a time, someone who uses a mouth-held stick for example.
- *MouseKeys*: Enables a group of keys to emulate a pointing device when present. Pressing keys in this group will move the pointer around the screen and perform associated pointer button actions (e.g., double-clicking). Target users are people who can't use or move a pointing device (e.g. a mouse) or operate its buttons.
- *RepeatKeys*: Enables the user to control various auto-repeat parameters of keys when they press. Target users are people who have trouble releasing a key before it starts to repeat.
- *SlowKeys*: Enables users to control the amount of time a key must be pressed before the system sends a key press event to the active application or tool. Target users are those who accidently press more than one key when they when interacting with the platform's.
- *BounceKeys*: Enables the user to introduce a delay time between keystrokes during which the system will not acknowledge repeated key presses of the same key. Tar-

get users are people whose tremors cause them to unintentionally bounce on, or repeatedly press and release the same key.

- *ToggleKeys*: ToggleKeys notifies users when they have locked or unlocked a locking key on the keyboard [e.g., Caps Lock]. Target users are people with reduced or no sight.

Scope

This functional specification defines the support that must be built into the system. The capabilities provided by the features in it define the pointer based events and capabilities that must be emulated by the system as well as how the system should interpret and process a user's keypresses. For the most part, defining and/or specifying the exact user interfac[s] these features are presented in is beyond the scope of this specification. The one exception area is keyboard shortcuts that are already considered de facto standards. These shortcuts are explicitly defined in the second table ("End-User Notification and Keyboard Invocation Requirements"). Specifying the keysequences for how a user controls and interacts the desktop and its contents, e.g. Control-C is Copy, is also outside the scope of this specification.

An exact set of test assertions for the user interface cannot be provided in this document because it does not prescribe what the user interface must look like. It is left to others, most likely the platform provider, to develop their own set of assertions for testing their specific implementation against this functional specification. The companion *"Generic Test Assertions for Manual Testing"*, which identifies how to test for compliance with this specification, was developed to address this limitation. A third document that will be produced, the *"XKB's Keyboard Access Support Specification"*, provides the first free software based implementation of this specification off which many platforms have based their keyboard access implementation.

Functional Specification

Configuration and Setting Requirements

This section identifies the configurable functionality each feature must make available to the user, the controls associated with the defined functionality, and the required ranges that must be supported in variable controls. It is important to note that the ranges associated with a variable type control are the minimum that must be supported; it is acceptable to provide a superset of these ranges, i.e. to support a wider range of settings than the minimum specified below. It is also acceptable to provide finer granularity, linear or non-linear, than that specified herein as long as the values defined below are supported. Unless otherwise noted, sub-features of features with a primary boolean control are only required to be available to the user when the primary boolean control has a value of "true". For instance, feature 1.3 need not be available to the end user unless 1.1 (the StickyKeys boolean) is "true". However, exposing these options in the user interface while the control is "false" should not in itself be deemed a conformance failure.

Table 1. Configuration and Setting Requirements

Feature	Functionality	Type of Control	Option's Variable Ranges	
1. StickyKeys	1.1. Turn StickyKeys on/off.	Boolean	N/A	

Feature	Functionality	Type of Control	Option's Variable Ranges	
	1.2. Press modifier key twice to lock.	Boolean	N/A	
	1.3. Turn StickyKeys off when two keys are pressed simultaneously.	Boolean	N/A	
	1.4. Provide the ability to request an audible signal when a modifier is latched, locked, or unlocked.	Boolean	N/A	
2. MouseKeys	2.1. Turn MouseKeys on/off.	Boolean	N/A	
	2.2. Delay before the acceleration that follows the initial step starts.	Variable	Acceleration start delay: 1-1000 ms Required adjustment granularity at low end of range: 10 ms.	
	2.3. Initial pointer velocity/repeat interval.	Variable	Initial velocity: 1-200 pixels/sec Required adjustment granularity at low end of range: 2 pixel/sec. *Note: this may be exposed as either a velocity in pixels/sec or a repeat interval between succesive pointer motions (if the step size is also exposed).*	
	2.4. Delay till the pointer reaches maximum speed.	Variable	Time to reach full acceleration: 100-10000 ms Required adjustment granularity at low end of range: 200 ms.	
	2.5. Maximum pointer speed, in pixels/sec. May alternatively be exposed as a repeat interval with range 1-1000 ms and granularity 1 ms.	Variable	Maximum speed: 1-2000 pixels/sec Required adjustment granularity at low end of range: 10 pixel/sec.	

Feature	**Functionality**	**Type of Control**	**Option's Variable Ranges**	
	3. RepeatKeys	3.1. Turn RepeatKeys on/off	Boolean	N/A
3.2. Repeat delay		Variable	Range: 0.10 to 5.0 seconds Required adjustment granularity at low end of range: 0.1 seconds.	
3.3. Repeat rate		Variable	Range: 0.2 characters/second (i.e. 1 character takes 50 seconds) to 10 characters/second. Required adjustment granularity at low end of range: 0.3 characters/second.	
4. SlowKeys	4.1. Turn SlowKeys on/off.	Boolean	N/A	
	4.2. Provide the ability to request an audible signal when SlowKeys is about to be turned on/off via the keyboard.	Boolean	N/A	
	4.3. Provide the ability to request an audible signal when a key is pressed.	Boolean	N/A	
	4.4. Provide the ability to request an audible signal when a key is accepted.	Boolean	N/A	

Feature	Functionality	Type of Control	Option's Variable Ranges	
	4.5. Provide the ability to request an audible signal when a key is rejected.	Boolean	N/A	
	4.6. Acceptance delay.	Variable	Range: 0.05-10 seconds Required adjustment granularity at low end of range: 0.25 seconds.	
5. BounceKeys	5.1. Turn BounceKeys on/off.	Boolean	N/A	
	5.2. Provide the ability to request an audible signal when a key is rejected.	Boolean	N/A	
	5.3. Debounce time.	Variable	Range: 0.1-5 seconds Required adjustment granularity at low end of range: 0.1 seconds.	
6. ToggleKeys	6.1. Turn ToggleKeys on/off.	Boolean	N/A	
7. Supporting Features	7.1. Provide a means of requesting a warning dialog any time the SlowKeys or StickyKeys feature is invoked from the keyboard.	Boolean	N/A	

Feature	Functionality	Type of Control	Option's Variable Ranges	
	7.2. Provide an option for requesting an audible signal when a Keyboard Access feature is turned on/off from the keyboard.	Boolean	N/A	
	7.3. Provide a visual indication showing when the system generates an audible ('bell') signal.	Boolean	N/A	
	7.4. Provide a means of enabling the keyboard shortcuts for StickyKeys and SlowKeys to be turned on/off; turning this functionality off turns the features off and disables the keyboard shortcuts[a].	Boolean	N/A	

Feature	Functionality	Type of Control	Option's Variable Ranges	
	7.5. Provide a time-out option that turns StickyKeys and SlowKeys off automatically after a specified period of time without keyboard activity; it still must be possible to turn StickyKeys or SlowKeys on from the keyboard when this feature is turned on[a]>.	Variable	Never TimeOut then Range: 1 to 30 minutes. Required adjustment granularity at low end of range: 4 minutes.	
	7.6. Provide a never time out option for StickyKeys and SlowKeys[a]>.	Optional	Never time out.	
Notes: a. Common practice today is this functionality turns off all the keyboard access features. This specification only requires that this boolean control affect the SlowKeys and StickyKeys features, in line with what the XKB specification in the X Windows System provides.				

End-User Notification, Keyboard Invocation, and Pointer Emulation Requirements

This section describes the notifications that must be provided the user when specified changes to keyboard state occur - these include visual and audio notifications that are not already implicitly defined in Table 1.

Table 2. End-User Notification, Keyboard Invocation, and Pointer Emulation Requirements

General Requirement	Feature	Functionality Required
1. Provide a visual indication showing the state of keys and buttons.	1.1. StickyKeys	1.1.1. Indicate when StickyKeys is on.
		1.1.2. Indicate when a modifier(s) is in a latched state.
		1.1.3. Indicate when a modifier(s) is in a locked state.

General Requirement	Feature	Functionality Required
	1.2. MouseKeys	1.2.1. Indicate when MouseKeys is on.
		1.2.2. Indicate which pointer button is active.
		1.2.3. Indicate when the active pointer button is up or down.
	1.3. SlowKeys	1.3.1. Indicate when SlowKeys is on.
		1.3.2. Indicate when a key is pressed.
		1.3.3. Indicate when a key press is accepted.
		1.3.4. Indicate when a key press is rejected.
2. Provide a keyboard on/off guesture.	2.1. StickyKeys	2.1.1. Provide the ability to toggle StickyKeys on and off from the keyboard. On systems that utilize a keyboard: press Shift key 5 consecutive times.
	2.2. MouseKeys	2.2.1. Provide the ability to toggle MouseKeys on and off from the keyboard on systems that utilize a pointing device.
	2.3. SlowKeys	2.3.1. Provide the ability to toggle SlowKeys on and off from the keyboard. On systems that utilize a keyboard: hold down the Shift key for 8 seconds.
3. Provide an audible signal that indicates when a keyboard access feature or feature functionality has changed state.	3.1. General	3.1.1. Each keyboard access feature must provide an audible signal when it is turned on or off.
		3.1.2. The signal that is generated when a keyboard access feature has been turned on or off must be the same as that used on the other keyboard access features.
		3.1.3. A keyboard access feature's "On" audible signal must be different from its "Off" audible signal.
		3.1.4. Functionality in a keyboard access feature that essentially turns something On or Off should use an audible signal with a timbre similar to the one generated when a keyboard acess feature is turned "On" or "Off".
	3.2. StickyKeys	3.2.1. Provide an audible signal when a modifier is latched.
		3.2.2. Provide an audible signal when a modifier is locked.
		3.2.3 Provide an audible signal when a modifier is unlatched or unlocked.

General Requirement	Feature	Functionality Required
	3.3. SlowKeys	3.3.1. Provide an audible signal when SlowKeys is about to be turned on/off via the keyboard.
		3.3.2. Provide an audible signal when a key is pressed.
		3.3.3. Provide an audible signal when a key is accepted.
		3.3.4. Provide an audible signal when a key is rejected.
	3.4. BounceKeys	3.4.1. Provide an audible signal when a key is rejected.
	3.5. ToggleKeys	3.5.1. Provide an audible signal when a locking key is locked or unlocked.
4. For systems with a pointing device, provide the ability to manipulate the pointer from the keyboard.	4.1. MouseKeys	4.1.1. Move the pointer via key press.
		4.1.2. Activate the active mouse button via a key press (e.g., single and double click).
		4.1.3. Change the active mouse button via a key press.
		4.1.4. Hold down a pointer button via a keypress.
		4.1.5. If the system supports multiple pointer buttons, allow multiple pointer buttons to be pressed at the same time.

Feature Behavior Requirements

The following subsections provides more detailed descriptions of how the various keyboard access features and functionality should operate from a user interaction standpoint.

StickyKeys

Some users are unable to physically press more than one key at a time. With StickyKeys, the user can first press a modifier key, release it, and then press another key. For example, to get an exclamation point on a US QWERTY keyboard, the user can press the "Shift" key, release it, and then press the "1" key.

StickyKeys is required for systems that have a keyboard that requires the user to press more than one key at a time. When StickyKeys is enabled, the keyboard must behave as follows:

- When the user physically presses and releases a modifier key, the modifier logically *latches* ("sticks") down even though the user has released the modifier key.

  The modifier will stay in this latched state until a non-modifier key has been pressed. When a non-modifier key has been pressed, the latched modifier key(s) will automatically unlatch and will revert to the unmodified state when the non-modifier key is released. The net effect is as if the user were holding the modifier key(s) down in conjunction with the non-modifier key.

- A user may *latch* more than one modifier at a time by pressing any number of unique modifier keys before pressing a non-modifier key.
- A user may *lock* a modifier by pressing/releasing the modifier key twice in a row.

  When a modifier is locked, it is as though it has been permanently latched and will not be automatically unlatched when a non-modifier key has been pressed. To unlock a modifier, the user must press the modifier key a third time, which places the modifier in an unlatched and unlocked state.
- A user may *lock* more than one modifier at a time.
- Any *latched* or *locked* modifiers will also be used in conjunction with pointer events.
- Pressing pointer buttons will result in the same behavior as pressing a non-modifier key.

MouseKeys

Some users are able to use a keyboard but are unable to use devices such as a mouse or trackball. MouseKeys permits the user to emulate the pointer and its functionality from the keyboard.

When MouseKeys is enabled, the system must behave as follows:

- When the user presses and holds a key for moving the pointer's cursor, the cursor will move slowly at first and then accelerate based upon the configuration settings listed in Table 1. When the user releases the key, the cursor will stop immediately.
- When the user presses a key for selecting the "active" pointer's button, no input events will be generated. Instead, the key will set the button to be used for the button actions described below.
- When the user presses the key for performing a single-click action, the system will generate a button click (i.e., a sequential button press and release events) using the current "active" button (see above).
- When the user presses the key for performing a double-click action, the system will generate a button double-click (i.e. sequential button press, release, press, release events) using the current "active" button (see above).
- When the user presses the key for holding down a button, the system will generate a button press event using the current "active" button (see above). The button will remain logically pressed even when the user releases the key. Following this action, the user may use other keys to move the pointer cursor so as to perform a drag operation.
- When the user presses the key for releasing a button, the system will generate a button release event using the current "active" button (see above).
- More than one pointer button can be logically pressed at a time.

RepeatKeys

RepeatKeys is of primary use to users who have difficulty removing their fingers from keys before the auto-repeat behavior is activated. RepeatKeys is required for systems that provide an auto-repeat behavior. When RepeatKeys is enabled, the auto-repeat behavior is identical to the normal auto-repeat behavior of the system, but the system will use the repeat delay and repeat rate timings described in Table 1.

SlowKeys

SlowKeys enables users who regularly hit multiple keys by accident while typing. SlowKeys does so by requiring the user to press and hold a key for a period of time (the SlowKeys "acceptance delay" described in Table 1) before the key is accepted, and is required on systems that provide a keyboard.

When SlowKeys is activated, the user must press and hold a key for the "acceptance delay" period of time before it is accepted. Once the key is accepted, if the user continues to hold the key, the RepeatKeys settings will be used to handle the auto-repeat behavior. If the user releases a key before the "acceptance delay," the system will treat the press/release as though they never happened.

BounceKeys

BounceKeys requires a delay (the "debounce time") between keystrokes before accepting the next press of the same key, and is typically used by users with tremors to prevent inadvertent keypresses. BounceKeys is required on systems that provide a keyboard.

When BounceKeys is enabled, the first press of a key will be immediately accepted by the system. When the user releases the key, they must wait for the "debounce time"to be met before pressing that same key again. If the user presses the key before the "debounce time" has expired, that key press/release will be ignored. Note that the "debounce time" only applies to the last key released - that is, if a user presses a *different* key immediately after releasing a key, that different key will be accepted.

ToggleKeys

ToggleKeys notifies users when they have locked and unlocked a "self locking" key on the keyboard. Target users are people with reduced or no sight. Examples of "self locking" keys include the "Caps Lock," "Num Lock," and "Scroll Lock" keys.

When ToggleKeys is enabled, the system will provide the user with audible notification when a self locking key is locked or unlocked, preferably with unique signals that indicate whether the key has become locked or unlocked. Note that there is a distinction between keys that are locked/unlocked via StickyKeys and self locking keys. StickyKeys is used for keys that are not self locking. As such, when a key is locked or unlocked via StickyKeys, ToggleKeys should not provide audible notification of the event.