KRAD - Keyboard support

(Draft in progress below ... not completed)

Purpose

Primary: KRAD should include standard encoded semantics that support keyboard access to all its functionality. People who can't use a mouse should be able to use a keyboard to do all the tasks that are presented through the KRAD UI framework.

Secondary: Applications should be able to define custom keyboard shortcuts for their unique functions that are not covered by KRAD's support for the standard keyboard shortcuts.

Tertiary: When KRAD includes the architecture for supporting user preferences, the user should be able to add custom keyboard shortcuts for unique functions that they do repetitively that do not already have shortcuts associated with them.

These objectives support "power" users in doing frequent repetitive tasks, for example, in bulk data entry scenarios, in addition to user who can't manipulate a mouse.

Detailed Description

This is in research/analysis phase with the target to identify which of the standard keystrokes we are not inheriting today and potential fixes to do so (we know we need to fix the Enter key implementation in several places and some lightbox deficiencies, etc.).

There are 3 aspects to supporting keyboard navigation: the tabindex element, standard keyboard shortcuts inherited from the browser/OS, and custom keyboard shortcuts supplied by an application:

- **Tabindex:** Note that the standard user-manipulable elements are included by default in the tab order by the browser, without the application having to manage these (set in the CSS and DOM order): this includes links, buttons, input fields, other controls, navigation elements. The Tab & the Shift+Tab keys move focus forward/backward among these standard HTML controls and widgets. The tabindex element enables keyboard-only users to tab to elements that would not otherwise, by default, be included in this tab order:
  - New in ARIA, the tabindex attribute can now be applied to any displayable HTML element, making it easier to add items on a page into the keyboard tab order. (Prior to ARIA, the tabindex attribute was limited to form and anchor elements.) HTML5 includes these keyboard aspects of ARIA.
  - Widgets with tabindex=0 will be added to the tab sequence based on document order.
  - Widgets with tabindex>0 will be added to the tab sequence based on the TABINDEX value.
  - Widgets with tabindex<0 will not be added to the tab sequence but are enabled to receive keyboard focus (when jscript changes its tabindex value).
  - You can use a roving tabindex or the aria-activedescendant property to manage tab order within a widget or group.
  - (From WAI-ARIA:) "Once a widget has keyboard focus, the arrow keys,Space, Enter key, or other keyboard commands can be used to navigate the options of the widget, change its state, or trigger an application function associated with the widget."

- **Standard keyboard shortcuts** (Inherited from the Browser/Operating System).
  - Summarized in the table below is a list of the major standard keyboard shortcuts that web pages should be able to inherit from the browser (assuming standard-encoded web content). Applications should not assign these same shortcuts to other custom functions.
  - After looking through the table, if you'd like additional information on standard keyboard shortcuts, see the references section below.

- **Custom keyboard shortcuts** defined by the application (AccessKey). There are pros and cons to creating custom shortcuts.
  - First research!
    - Research and understand the standard keyboard shortcuts (shown in table below and in references section that follows). Don't try to map a standard shortcut key to a different custom function. Many browsers will override your custom function and for those that don't, users will be confused as to why the standard action for the shortcut doesn't apply within your application.
    - Research whether other entrenched applications in your same task domain have already implemented custom shortcuts for the same functions you are considering doing so. Users can't transfer their learning from application to application when they have different custom shortcuts for the same user task.
    - Research alternative ways to display the shortcuts so that users can more easily discover and learn them. Unless your application is expected to be used by many users in dedicated fashion for frequent and repetitive tasks, they might not learn and remember your custom shortcuts.
    - Consider whether you really need a custom shortcut for a particular task or function, or if there are other ways to enable faster navigation to them (nav elements, regions, skip links, roles and so on).
  - If a custom keyboard shortcut is really NEEDED, see the AccessKey attribute in the references section below.
    - Webaim
    - WAI-ARIA - keyboard support
    - HTML5 - keyboard support

Information below outlines relevant keyboard shortcuts we should respect/inherit, not interfere with or map to other functionality.

Extract of Standard Keyboard Shortcuts:

(Assuming standard-encoded web content, many standard keyboard shortcuts/behavior can be inherited, in appropriate context.)

STANDARD KEYBOARD SHORTCUTS	**MS Windows (Compare across IE, Firefox, Chrome)**	**MAC OS (Compare across Safari, Firefox, Chrome)**
"Global" meaning & focus/position should be Irrelevant:		
Open browser's help	F1	Command-?
Go to top of page *Go to bottom of page* *(DHTML proposal = with focus within a grid or table, Home/End goes to 1st/last cell of current row)*	Home End	FN+Arrow-Up FN+Arrow-Down
Jump 1 page up *Jump 1 page down* *(DHTML proposal = with focus within a grid or table, Page Up/Down goes to 1st/last cell of current column)*	Page-Up Page-Down	Option+Arrow-up Option+Arrow-down
Scroll up *Scroll down*	Arrow-Up Arrow-Down	Arrow-Up Arrow-Down
Jump back to previous page visited *Jump forward to former-next page visited*	ALT+Arrow-left ALT+Arrow-right	Command+[Command+]
Undo / redo last operation	CTRL+Z CTRL+Y	Command+Z Shift+Command+Z
Reload current web page	CTRL+r (… or F5)	Command+r
Zoom in *Zoom out* *Reset zoom*	CTRL+plus sign CTRL+minus sign CTRL+0	Command+plus sign Command+minus Command+0
Increase screen contrast *Decrease screen contrast*	(LeftALT+LeftShift+PrtScn)	Option+Control+Command+, Option+Control+Command+.
Activate the menu bar in the active program	F10	Control+F2
Open the title bar menu	ALT+Space	Control+F5
Alternate focus btw address bar & webpage	ALT+d (or F6) Shift+ALT+t	Command+l
Move forward / backward through browser tabs (some browsers won't use these when the focus is within the tab page)		CTRL+PageUp CTRL+PageDown
Move forward thru tabs in the browser's tabgroup *Move back thru tabs*	CTRL+Tab CTRL+Shift+Tab	
Jump to specific tab in the browser's taborder	CTRL+# (#=1-9)	
"Global" meaning but focus/state is relevant to the meaning (the object it will act on):		
Show context-sensitive help for currently-focused element	Shift+F1	Control+F1
Show context menu for the currently-focused element (like a mouse right-click)	Shift+F10	

Jump to next focus element/control in the viewport / *Jump to previous focus element/control (across all controls: address bar, toolbar, search bar, & page elements: navigation groups, links, & into/out of a radio-button control, a check-box control, a grid or a table)*	Tab / Shift+Tab	Tab / Shift-Tab
Jump to next item on a webpage or within a radio-button control, menu, tabgroup, table or grid *Jump to previous item* *(in the page area - a link, input field or control – this selects it)*	Arrow keys (down & right = forward) (up & left = backward)	Option+Tab / Shift+Option+Tab
Jump to next message in a list / *Jump to previous message in a list*	CTRL+> / CTRL+<	
Move to value or cell within a view, such as a table		Arrow keys
Open the next menu to the right (in a menu bar) / *Open the next menu to the left*	Arrow-Right Arrow-Left	
Move down through the menu options (in drop-down menus & radio-buttons, etc.) / *Move up through the menu options (in drop-down menus)*	Arrow-Down / Arrow-Up	
Move back to the tab (or accordion header) from anywhere within the tab page	CTRL+Arrow-Up or Arrow-Left	
Move focus between the application and an open non-modal window	F6	
Select a checkbox or radio button or toggle button (or clear it if selected), or open a combobox or drop-down list	Space	Space
Activate the selected item (link, button, menu bar item to open a drop-down or pull-aside menu, etc.) – same as mouse click	Enter (S/A clicking mouse) Space	Enter Space
Cancel the current task: Stop webpage loading or close a selected drop-down list or menu, or close a dialog box, or close a tooltip	ESC	ESC
"Local" meaning and focus/position is relevant to the object it will act upon:		
View a folder one level up	ALT+Arrow-Up	
Open a drop-down list (this is DHTML recommendation, but it is in violation of the above, which is the space key for this)	ALT+Arrow-Down	

Usage Scenarios

Include at least one usage scenario, from the user's task perspective, that might be helpful in understanding the issue:

Mocks and Diagrams

Include any mocks (for UI enhancements) or diagrams that might be helpful in understanding the issue:

Performance

If applicable, list expectations for performance (optimal and worst cases would be fine, give time in seconds):

References

Standard keyboard mappings:

- Wikipedia article
- DHTML key mappings (workgroup led by AOL)
- Yahoo keyboard mappings
- Firefox article
- About.com article
- Microsoft article
- Mapping keys article

TabIndex and AccessKey attribute:

- Webaim
- WAI-ARIA - keyboard support
- HTML5 - keyboard support

Older Jiras:

- https://jira.kuali.org/browse/KULRICE-5368
- https://jira.kuali.org/browse/KULRICE-5512
- https://jira.kuali.org/browse/KFSMI-742
- https://jira.kuali.org/browse/KFSMI-2761
- https://jira.kuali.org/browse/KFSMI-4545
- https://jira.kuali.org/browse/KSLAB-1339
- https://jira.kuali.org/browse/KSLAB-1341
- https://jira.kuali.org/browse/KULFDBCK-555
- https://jira.kuali.org/browse/KSENROLL-546
- https://jira.kuali.org/browse/KSENROLL-545

Requirements Listing

List all requirements (individual verifiable statements) that indicate whether the work for this item has been complete. If there are requirements that are not essential to the functionality but would be nice to have if time allows, enter those under 'Secondary':

Primary:

1. *item*
2. *item*

Secondary:

1. item
2. item

Dependencies

List any functional or technical work that must be completed before work on this item can begin:

1. item

Issues

List any issues that need to be resolved before work on this item can begin:

Functional:

1. item
2. item

Technical:

1. item

2. item

QA or Regression Testing Plan

List steps needed to test the basic functionality of this update, enhancement, bug fix

1. test/steps
2. test/steps

Checkoff

Functional Analysis Complete? **No (completed by SME)**

Needs Review by KAI? **No (completed by SME)**

Technical Analysis Complete? **No (completed by DM)**

Needs Review by KTI? **No (completed by DM)**

Estimate: **30 hours (completed by DM)**

Technical Design: Link Here **(completed by DM)**

Jira: Link Here **(completed by SME)**

Final Documentation: Link Here **(completed by DM)**

Added to QA: **No (completed by SME)**