assistive technology. As with automatically updated non-real-time content, developers should provide controls to stop or pause automatic updates. With real-time content, however, developers should resume the updates at the current value or time, regardless of when the user stopped it. For example, when a user pauses automatic updates for an auction item's price and the time left in the auction, the current bid price and time remaining should be shown when the user is activates the Play button; for example:

```
btnPlayPause.accessibilityName = "Pause display of automatically updating content
-- this does not stop the auction countdown timer";
btnPlayPause.addEventListener(MouseEvent.CLICK, playPause);
function playPause(e:MouseEvent): void
{
 // set accessible name based on the hypothetical Boolean playing
 if (playing)  // stop playing and change the pause button to play
   btnPlayPause.accessibilityName = "Unpause display of automatically updating content
-- display the current information regarding this auction item";
 else  // play and change the play button to pause
   btnPlayPause.accessibilityName = "Pause display of automatically updating content
-- this does not stop the auction countdown timer";
 // change skin of button not shown
 // If content should update, pull content
}
```

Keyboard accessibility

An application is considered *keyboard accessible* if users can control it solely through the keyboard. This is a core capability for accessible applications because it is essential for individuals who are blind or visually impaired, as well as those who have mobility impairments and thus have difficulty using a mouse. For an application to be considered keyboard accessible, *all* functionality it offers must be accessible from the keyboard. Any component that can be manipulated via the mouse must also be accessible via the keyboard. The does not imply that keyboard access should replace mouse access; developers should continue to offer multiple methods for using and activating components. Many users with disabilities are unable to operate a keyboard and rely solely on the mouse or pointing devices.

The accessible Flex Spark and MX components support keyboard accessibility by automatically making mouse-defined events accessible via the keyboard. For example, the user can check a CheckBox control, select a List item, and open a ComboBox control using only the keyboard with the default component implementation. As a result, most Flex applications that use accessible components are keyboard accessible, or can be made so with little effort. Keyboard accessibility issues typically arise when developers use custom or inaccessible components, add enhanced mouse functionality to provide one-click access, or move the keyboard focus based on input during form validation.

Components that are not presented in the correct tab order also cause accessibility problems for keyboard-only users. Keyboard accessibility depends on the correct implementation of tab order and shortcut access.

- **Tab Order** — The tab order of an application is controlled by the `tabIndex` component property, which allows developers to control the tab order of all tab-enabled components within an application. This order is the sequence that the keyboard focus follows when a user presses the Tab key; the reverse order is followed when the user presses Shift+Tab. The `tabIndex` property can be defined in MXML or ActionScript for any display objects with which the user can interact (any InteractiveObject control). All `tabIndex` values must be greater than 0 and should not be duplicated in the tab order. The `tabIndex` values do not need to be sequential.
- **Keyboard Shortcuts** — Keyboard shortcuts enable quick access to components or actions via the keyboard. Proper use of keyboard shortcuts is vital to keyboard accessibility because it minimizes the number of keystrokes necessary to access objects. Keyboard shortcuts can be defined in MXML or ActionScript using the `accessibilityShortcut` property for each UIComponent element.