```
// The following code would cause a forced focus change when the user
activates the up/down arrows in a closed dropdown list or combo box
// this type of event should be avoided unless there is a method to
provide keyboard access without a forced focus change
cbSelectState.addEventListener(Event.CHANGE, shiftFocus);
function shiftFocus(): void
{
   txtZipCode.setFocus();
}
```

Forced focus changes can be confusing to all users, and particularly users of screen readers. When a forced focus change is made, developers should notify the user where focus will be placed. For example, if tabbing out of a set of radio buttons after a selection causes focus to move to a different field based on the selected radio button, this should be indicated in the `accessibilityName` of each radio button; for example:

```
rbtUserAccountTypeAdministrator.accessibilityName = "Administrator -
moves focus to the administrator password field when selected after
navigating away from the radio button group";
rbtgrpUserAccountType.addEventListener(FocusEvent.FOCUS _ OUT, setFocusToField);
function setFocusToField(e: FocusEvent): void
{
  switch (e.target)
  {
    case rbtUserAccountTypeAdministrator:
    {
      txtAdministratorPassword.setFocus();
    }
    ...
  }
}
```

Note: A forced focus change is generally acceptable if it is in response to the activation of a form's submit button. The focus should move to the top of the next screen or to an error message at the top of the current one. This will assist keyboard-only users in quickly interacting with the new content and enable screen reader users to easily learn if the submission was successful.

```
btnPreferences.addEventListener(MouseEvent.CLICK, goToCart);
function goToCart(e: MouseEvent): void
{
  // if user is not logged in, open login screen (code not shown) and
  // set focus to the first field on the screen, the username field
  txtUserName.setFocus();
}
```

Radio buttons and combo boxes

Forced focus changes on radio button groups and combo boxes, in which the focus moves to the next element or another screen once a selection has been made, are also to be avoided.

For example, consider a radio button group with five selections. To select the fourth item using the default keyboard control for the group, the user would tab to the radio button group and press the Down Arrow key three times. However, if a forced focus change is implemented to the `onChange` event handler for the radio button group, pressing the Down Arrow key to move from the first radio button to the second would initiate the forced keyboard focus before the user could select the desired option. While this simple example clearly illustrates a problem, the situation is made significantly more frustrating when a selection change triggers an unintended screen change or a more intensive process, such as a lengthy database search.