In many cases, there will be more elements in the reading order than in the tab order because screen reader users must access not only keyboard-accessible controls and form fields but also text and images that do not receive keyboard focus. Flex application developers should define `tabIndex` values for all objects that are to be read, including text fields and other non-focusable interface controls.

In practice, there are two main strategies for defining `tabIndex` values for components in an application. For applications in which the overall reading and tab order can be defined at compile time, developers can set the order using static `tabIndex` values. In this case, each component that is to be read explicitly defines a `tabIndex` value in its MXML definition. When manually setting values it is often a good idea to leave space for future components. For example, use `10`, `20`, `30`, `40`, and `50` as `tabIndex` values rather than `1`, `2`, `3`, `4`, and `5`. If additional fields are later needed between the first and second components, for example, they can be assigned `tabIndex` values of `14` and `16`, and the existing values will not need to be renumbered. To ensure a consistent, logical order, use `tabIndex` values that are greater than zero and do not use the same value more than once in the order.

For complex applications in which the number and placement of components is not known at compile time, often a better strategy is to place all of the components in order in an array, collection, or vector and update the `tabIndex` property via ActionScript. When new components are added, the application inserts them into the collection at the proper location, and updates the `tabIndex` property for each element in the collection. A similar approach is used for deleting or reordering components. This approach is easy to scale and can be more convenient than reconciling the order of individual components at compile time.

Addressing reading order issues with compound components

Many compound components, such as Accordion, TabNavigator, and Panel containers, comprise several visual components. Accordion components, for example, contain header components. When the `tabIndex` property is set on the Accordion container, the change does not automatically propagate to the individual headers. The header components are likely to appear in the incorrect reading order if their individual `tabIndex` properties are not set. Developers must explicitly set the `tabIndex` property on child components for the following components:

- MX Accordion
- MX TabNavigator
- MX Panel

The following example illustrates one way to set the reading order for children of compound components. The `init()` function, which is called when the application is initialized, loops through each child of the Accordion container and sets the `tabIndex` property on the child header to the value of the Accordion container's `tabIndex` plus the position of the child. In this example, the Accordion container has a `tabIndex` of `50`, so the first child header is assigned a `tabIndex` of `51`, the second `52`, and so on.

```
...
<fx:Script>
<![CDATA[
function init(): void
{
  // set the tabIndex of accordion headers so they appear in the correct reading order

  var i:int = 0;
  while (i < a1.numChildren)
  {
    if (a1.getHeaderAt(i))
      a1.getHeaderAt(i).tabIndex = a1.tabIndex+i+1;
    i++;
  }
```