```
            </mx:VBox>
      </s:Panel>
</mx:VBox>
    </mx:HBox>
<s:Label paddingLeft="10" text="Payment required at checkout" tabIndex="27"/>
</mx:VBox>
</s:Application>
```

The code above explicitly sets the reading order by assigning values to the `tabIndex` attribute for each component via MXML (`tabIndex` can also be set via ActionScript).

Reading order best practices

There are several best practices to keep in mind when defining the reading order for an application.

Be logical

For anything but simple Flex applications, explicitly define the reading order so that screen reader users hear the contents of a page in a logical, intuitive way—usually the same way a sighted user would read the page. Since the same property (`tabIndex`) is used to control both reading order and tab order, setting `tabIndex` appropriately to specify a logical reading order will also enable keyboard-only users to tab through a page without excessive jumping around and to follow form fields in a way that makes sense. See "Keyboard accessibility" on page 15 for more information.

Order error messages first

Error messages that pertain to an entire form should be read before the form itself, so they need to be placed at the top of the form and first in the reading order. Field-specific error messages should be provided inline (appended to the component's accessible name) and at the beginning of the form; for example:

```
// ActionScript
txtFormError.tabIndex=2;
// other form fields here
txtPhoneNumberError.tabIndex=9;
txtPhoneNumber.tabIndex=10;

// MXML
<s:Text id="txtFormError" tabIndex="2" />
<s:Text id="txtPhoneNumberError" tabIndex="9" />
<s:TextInput id="txtPhoneNumber" tabIndex="10" />
```

See "Forms" on page 32 for more information on providing accessible error messages.

Disable accessibility for decorative and non-relevant content

All purely decorative content should be hidden from assistive technology. When nonessential content is not presented to users of assistive technology such as screen readers, the important content is easier to access.

Likewise, content that is hidden behind a modal TitleWindow layout container or simulated dialog box should be made inaccessible. Modal windows require the user to interact with them when they appear on top of other content, so the accessibility of all elements in the background should be disabled. When the simulated dialog or modal TitleWindow is closed, accessibility should be enabled for the components once again. For example, when a help button is activated and a TitleWindow appears with help content in it, set `accessibilityEnabled` to `false` for all of the content in the background, including the Help button:

```
// turn off accessibility for the Help button
btnHelp.accessibilityEnabled = false;
// turn off accessibility for all other background content --
     in this case in a VGroup component and the components contained in it
vgMainGrouping.accessibilityEnabled = false;
```

Only the content in the TitleWindow container should have accessibility enabled.