elements provide a label using one of the techniques described in that section. In addition to simply providing the component name information, however, developers must make any supplementary visual information required to complete the form available in the form labels to ensure that the form is usable across the widest array of assistive technology. Commonly this issue arises when form field constraints are conveyed visually, most often to indicate a required field. For the field to be accessible, this constraint information must also be included in the accessible label of the form field. For example, when a field is required then "(required)" or similar text should be added to the component name of the form field. An asterisk may also be used as long as text indicating what the asterisk means is added to the beginning of the form.

```
// ActionScript
txtSearch.accessibilityName = "*Search Term:";

// MXML
<mx:Text id="txtSearch" accessibilityName="*Search Term:" />
```

When FormItem components are used for required fields, set the `required` property to `true`. Screen readers will announce "required" after the accessible name of these required components.

```
<mx:FormItem label="Search Term:"><mx:TextInput id="txtSearch" required="true"/>
     </mx:FormItem>
```

Developers should ensure that any visible form label text is provided in a location that is aligned with operating system style guides and language-specific reading paradigms. In general, this means providing text fields to the right of check boxes and radio buttons, and to the left or above all other form controls. These guidelines are implemented by default by Flex components, but label placement for some controls can be set explicitly using the `labelPlacement` attribute:

```
// ActionScript
chkGender.labelPlacement = mx.controls.ButtonLabelPlacement.RIGHT;

// MXML
<mx:CheckBox id="chkGender" labelPlacement="{ mx.controls.ButtonLabelPlacement.RIGHT}" />
```

Note: For Spark radio buttons and check boxes there is no `labelPlacement` attribute. Instead a custom skin must be created to place the check box or radio button label anywhere other than to the right of the control.

In general, developers should ensure that visible labels are not present in the reading or tab order of the document. While this may seem counterintuitive, it prevents assistive technology from rendering the labels multiple times. As a basic example, consider a TextInput control with the visual label "Name" provided above the text entry field:

```
<s:Label text="Name" />
<s:TextInput width="252" height="21" accessibilityName="Name" />
```

When a typical screen reader encounters this set of elements it will first read the visible Label text, "Name." It will then process the text entry field and read, "Text - Name." The user will hear "Name . . . Text – Name". To prevent the screen reader from announcing the name twice, set the `accessibilityEnabled` field of the Spark Label element to `false`; for example:

```
<s:Label text="Name" accessibilityEnabled="false" />
<s:TextInput width="252" height="21" accessibilityName="Name" />
```

When a FormItem component is used to set the label, setting `accessibilityEnabled` to `false` is not feasible because it will disable accessibility for the text input also. See "Silence form items and form headings" on page 28 for instructions on handling this situation.

Form element grouping

Related form elements should be grouped together with any information that is required to understand them. As an example, consider two radio buttons labeled "Yes" and "No". The button label text makes no sense unless the buttons are associated with the relevant question. When the radio buttons are grouped with the