Accessibility guidelines for rich interfaces and JavaScript

(Use cases to comply with France's accessibility guidelines)

Date	Version	Author	Status / comments
07 February 2014	1.0	Atalan	English version.

In partnership with:
Air Liquide – Atos – BNP Paribas – Capgemini – EDF – Generali – SFR – SNCF – Société Générale – SPIE

Observers:
AbilityNet – Agence Entreprises & Handicap – AnySurfer (*Belgium*) – Association des Paralysés de France (APF) – Association Valentin Haüy (AVH) – CIGREF – Design For All Foundation (*Spain*) – ESSEC – Handirect – Hanploi – Sciences Po – Télécom ParisTech

Acknowledgements

Special thanks to our AcceDe Web partners for their input and commitment:

Thanks to their support, participation in the working groups, proofreading of documents, and testing of manuals, the partner companies have provided valuable input to the Accede Web project. This has made it possible to create manuals that successfully meet the needs of the different stakeholders of a web project. These companies are also contributing to making the web more accessible by authorizing the free distribution of this content.

We would also like to thank the following observers of the AcceDe Web project whose actions raise awareness of the AcceDe Web manuals and aid their distribution: AbilityNet (*UK*), Agence Entreprises & Handicap, AnySurfer (*Belgium*), Association des Paralysés de France, Association Valentin Haüys, CIGREF, Design for All foundation (*Spain*), ESSEC, Handirect, Hanploi, Sciences Po and Télécom ParisTech.

We thank all the members of the review committees for the quality and relevance of their comments:

- English version: Robin Christopherson (UK - AbilityNet), Ramon Corominas (Spain - Technosite), Deborah Edwards-Onoro (USA - Lireo Design), Thomas Frandzen (Denmark - Agency for Digitisation, Ministry of Finance), Char James-Tanny, (USA - a11yBos), Matt May (USA – Adobe), Sophie Schuermans (Belgium – Anysurfer), Jared Smith (USA - WebAIM), Christophe Strobbe (Germany), Gijs Veyfeyken (Belgium – Anysurfer)

- French version: Benjamin Ach (accessibility expert), Vincent Aniort (APF), Jean-Baptiste Audras (University of Grenoble), Claire Bizingre (accessibility consultant), Victor Brito (accessibility consultant), Anna Castilla (Provaltis), Ève Demazière (Sciences Po), Nicolas Fortin (French Ministry of Media and Culture), Marc-Étienne Vargenau (Alcatel-Lucent) and Éric Vidal (Fédération des aveugles de France), as well as all the teams of our partner companies.

Finally, we would like to thank Laurent Bracquart and Johan Ramon, Christophe Pineau and Marion Santelli of Atalan for their dedication and commitment to this initiative.

Sébastien Delorme, Sylvie Goldfain,
Atalan

Contents

Context and objectives

This manual provides both:

- The general accessibility guidelines to consider during the development of rich interfaces, whatever the target device for displaying the content (computer, mobile phone, etc.).
- Practical help on how to include accessibility in some standard use cases (drop-down menus, tabs, etc.) in order to comply with France's accessibility guidelines (which do not consider ARIA yet).

This manual is part of a set of four complementary manuals that can be downloaded from the http://www.accede-web.com/en/ website:

1. Accessibility guidelines for graphic design.
2. Accessibility guidelines for HTML and CSS.
3. **Accessibility guidelines for rich interfaces and JavaScript (this manual).**
4. Accessibility guidelines for editors (template).

Note
The solutions provided in this manual are not unique—each is often just one of many possible accessibility solutions. With the required knowledge, most of the solutions described in the use cases can be adapted to the required context.

Who should read this user guide, and how should you use it?

This document should be given to stakeholders and/or service providers who create the technical specifications and the HTML/CSS templates, and who are responsible for the different technical developments of rich interfaces. It should be used in addition to the technical specifications of a project, and the Accessibility guidelines for HTML and CSS manual. The recommendations may be supplemented with others, or left out, according to the circumstances—the project manager is often the most appropriate person for this task.

The recommendations should be considered for the HTML/CSS, JavaScript, and Flash integration phases. Some recommendations apply when pages are produced dynamically from templates that are integrated in a content management system (CMS).

There are some annotations that complete the document and should be read to understand each recommendation:

- **Note**: notes help complete the recommendations by providing additional details for specific situations.
- **Warning**: warnings highlight specific points that require attention or traps to avoid in order to guarantee good accessibility.
- **Tip**: Tips are not directly linked to accessibility, but you can improve the general quality of the interfaces by implementing them, or facilitate the integration of accessibility in the subsequent steps in the project. Note that the recommendations in this project, although

aimed at accessibility, are often good practice for ensuring ease of use, and improved user experience, performance, and referencing.

- **HTML5:** The HTML5 insets provide additional details or warnings to the recommendations in the light of the developments proposed in the new HTML specification.

License agreement

Contact

Please send any comments about this document to Atalan, the coordinator of the AcceDe Web project, at the following email address: accede@atalan.ca.

You can also find more information about the AcceDe Web project procedural manuals on the website http://www.accede-web.com/en/ or follow our twitter account @societe_atalan.

Credits

The icons used in the AcceDe Web guidelines are from a set of *24x24 Free Application Icons* (http://www.small-icons.com/packs/24x24-free-application-icons.htm).

[1] For more information on the Creative Commons BY 3.0 license, see http://creativecommons.org/licenses/by/3.0/fr/.

1. General recommendations for rich interfaces and JavaScript

1.1. Make a distinction between buttons and links

To help users understand rich interfaces more easily, buttons should have a different function to links.

- Buttons `<button>`/`<input>` are used to perform actions. They are used to submit information, display the following or previous element in a carousel, close a pop-in, etc.
- The `<a>` links allow the user to navigate. They are used to navigate to another page, or to a specific point on the current page by pointing to an anchor.

1.2. Implement alternatives to JavaScript

Whenever content or functionalities are produced using JavaScript, remember to put an alternative into place. The alternative should allow the user to access similar content or functionalities if JavaScript is not activated on the user workstation.

The table below lists some examples of standard JavaScript uses, with a suggestion of an alternative when JavaScript is deactivated.

When JavaScript is enabled	When JavaScript is disabled
Error management of a form is performed on the client side.	Error management of a form is performed on the server side.
An accordion system allows for the dynamic hiding and display of specific content.	All content is shown on the page, with one section following the other.
An interactive map indicates the physical address of an organisation.	The postal address of the organisation is indicated in text format.

1.3. Implement alternatives to Flash

Each time that content or functionalities are produced with Flash, remember to put an alternative into place. The alternative should allow the user to access similar content or functionalities, in the event that a Flash reader is either not installed or activated on the user workstation.

The table below shows some standard uses of Flash, with alternatives for when the Flash reader is either deactivated or not installed.

When Flash is enabled	When Flash is disabled
A Flash video player can be used to show video content.	An HTML5 video player can be used to show video content, or a link to download the content can be supplied for each video.
A chronological frieze in Flash recounts the history of a company.	The history of a company is shown in HTML content.

Note
For accessibility, it is considered good practice to provide a link to the alternative version near the Flash animation. The idea is to allow users to access the alternative version, even if Flash is available. For a chronological frieze, for example, you could provide a link “Alternative version of chronological frieze”.

Warning
If JavaScript is used to load a Flash animation, make sure the animation is loaded properly when JavaScript is not activated. It is possible for a Flash reader to be installed even though JavaScript is deactivated.

1.4. Allow scripts to be controlled using both the mouse and the keyboard

Whenever a script is controlled by the mouse, it must also be possible to control it with the keyboard, and conversely.

In other words, each time the mouse can be used to control a script, using an element such as a link or button, the user must be able to:

- Reach the script using the keyboard.
- Control the script using the keyboard once the focus is set on the corresponding element.

Warning
Whenever possible, use generic event listeners methods rather than specific ones for a keyboard button. For example, use `onfocus`/`onblur` rather than `onkeydown`.

This is important because keyboard shortcuts are not always the same on different systems and browsers. For example, in the Opera browser, the `shift` key, together with the arrow keys, allows the user to navigate from one link to another. However, most other browsers use the `Tab` key for this shortcut.

Note
It is very important to avoid **keyboard traps**. Keyboard traps block a user who navigates with the keyboard, because they prevent interaction with specific zones on the page.

Keyboard traps arise when a user is blocked on a page when navigating with a keyboard:

- Either because an element prevents the user from moving the focus to the previous or next interactive element.
- Or because an element does not let the user move on from the current focus, once it is applied to an element.

Each time that scripts are used on a page, make sure they do not contain keyboard traps by running a simple test on the interface with the keyboard.

1.5. Allow the user to pause automatic animations

Whenever elements on a page are likely to move, implement a system to control the movement.

Including play and pause buttons is sufficient. You could include both actions in the same button, which is updated dynamically.

Some standard cases of moving content:

- A carousel of news articles.
- An animated advertisement.
- A video.
- Etc.

Warning
Whatever the visual position of this button on the page, it must always be placed before the animated elements in the DOM. With a multimedia player, make sure this button is the first focusable element of the player.

Note
You don't need a pause button on progress bars for moving elements which stop automatically in five seconds or less.

2. Use cases

2.1. Navigation

2.1.1. Drop-down menus

Purpose

Drop-down menus are dynamic menus which show only the first level items by default, but which show other, corresponding, drop-down panels when you move the cursor over, or set the focus on, the first level items.

Warning
If the items on the first level are not links, and their only function is to display or conceal the panels, then the drop-down menu should be considered as an accordion menu.

In this situation, it may be a good idea to modify the script for a standard accordion, so that the content of the panels is also displayed when you move the mouse pointer over the item.

Tip
If there are a great deal of interactive elements on the drop-down panels (links, form elements, etc.), it is a good accessibility practice to give priority to the accordion behaviour.

Displaying the drop-down panels only when the user interacts with the corresponding first level item makes it possible to optimise the page navigation with the keyboard, by limiting the number of elements that can receive the focus, by default.

Basic HTML code

An example of the basic HTML code for a drop-down menu is shown below. This code needs to be modified according to the context, especially with respect to the content of the drop-down menus.

```
<ul id="menu">
   <li>
      <a href="#">First level link 1</a>
      <ul>
         <li><a href="#">Second level link 1</a></li>
         <li><a href="#">Second level link 2</a></li>
         <li><a href="#">Second level link 3</a></li>
      </ul>
   </li>
   <li>
      <a href="#">First level link 2</a>
      <ul>
         <li><a href="#">Second level link 4</a></li>
         <li><a href="#">Second level link 5</a></li>
         <li><a href="#">Second level link 6</a></li>
      </ul>
   </li>
   <li>
      <a href="#"> First level link 3</a>
      <ul>
```

```
            <li><a href="#">Second level link 7</a></li>
            <li><a href="#">Second level link 8</a></li>
            <li><a href="#">Second level link 9</a></li>
         </ul>
      </li>
</ul>
```

Without JavaScript, all the items of the menu (first level items and panels) are displayed.

Behaviour with JavaScript

JavaScript must manage the following points:

1. When the script is loaded, the panels are positioned outside the screen using the CSS property `position: absolute;` and `left: -999999px;`.
2. Then:
    - When a first level item receives the focus, or when the mouse pointer is over this element, the corresponding panel is displayed on the screen.
    - When the first level item loses the focus, or when the mouse pointer is no longer over this item, the corresponding panel is moved outside the screen.

Demonstration

A demonstration is available in the online version of this recommendation at: http://wiki.accede-web.com/en/notices/interfaces-riches-javascript/.

2.1.2. Accordions (fold/unfold)

Purpose

Accordions are dynamic modules that are used to optimise page content display in a small space. They use a system of “fold/unfold”, controlled by the user, often by clicking on a title or a link above the content block.

Basic HTML code

An example of the basic HTML code for an accordion with panels concealed by default is shown below. This code may be modified according to the context, notably with respect to the different heading levels.

```
<h1>Heading of the first panel</h1>
<div id="panel-1">
    <p>Content of the first panel</p>
</div>

<h1>Heading of the second panel</h1>
<div id="panel-2">
    <p>Content of the second panel</p>
</div>

<h1>Heading of the third panel</h1>
<div id="panel-3">
    <p>Content of the third panel</p>
</div>
```

Without JavaScript, all of the system is expanded (displayed).

Note

Even though it is not so recommended, it is possible, without JavaScript, to keep the links inside the headings. In this case, they are links that point to the corresponding anchors:

```
<h1><a href="#panel-1">Heading of the first panel</a></h1>
<div id="panel-1">
    <p>Content of the first panel </p>
</div>
```

Behaviour with JavaScript

The scripts are then added to the basic HTML code to handle the following points:

- The panel headings include links that make it possible to fold or unfold the panels.
- The panels are hidden by default using the property `display: none` in the CSS.
- By using the mouse or the keyboard to execute a link-heading, you can display or hide the corresponding panel (`display: none` to hide it and `display: block` to display it).
- The `aria-expanded="false"` attribute is applied by default on the link-heading. The value of this attribute then changes dynamically to `true` when the corresponding content is displayed and `false` otherwise.
- The `aria-controls` attribute is applied to the link-heading. The value of this attribute is retrieved from the ID attribute of the corresponding panel.

The resulting code with JavaScript is as follows:

```
<h1><a href="#panel-1" aria-expanded="false" aria-controls="panel-1">First
panel heading</a></h1>
<div id="panel-1">
    <p>First panel content (hidden)</p>
</div>

<h1><a href="#panel-2" aria-expanded="true" aria-controls="panel-2">Second
panel heading</a></h1>
<div id="panel-2">
    <p>Second panel content (displayed)</p>
</div>

<h1><a href="#panel-3" aria-expanded="false" aria-controls="panneau-
3">Third panel heading</a></h1>
<div id="panel-3">
    <p>Third panel content (hidden)</p>
</div>
```

Demonstration

A demonstration is available in the online version of this recommendation at: http://wiki.accede-web.com/en/notices/interfaces-riches-javascript/.

2.1.3. Tabs

Purpose

Tabs are dynamic modules that enable you to optimize the display of page content in a small space. They are usually shown in the form of a list of side-by-side links that display the corresponding content of the selected tab. You can only activate one tab at a time.

Basic HTML code

The HTML code includes internal links that point to the corresponding panels.

```
<ul>
    <li><a href="#panel-01">First tab</a></li>
    <li><a href="#panel-02">Second tab</a></li>
    <li><a href="#panel-03">Third tab</a></li>
</ul>

<div id="panel-01">
    Content of the first panel.
</div>

<div id="panel-02">
    Content of the second panel.
</div>

<div id="panel-03">
    Content of the third panel.
</div>
```

Without JavaScript, all the panels are displayed, and the tabs are used as links to directly access the content of the corresponding panel.

Behaviour with JavaScript

The scripts are then added to the basic HTML code to handle the following points:

- With the exception of the active panel, the panels are hidden by default using the `display: none` attribute in the CSS.
- The `role="tabpanel"` attribute value is applied to each panel.
- The `role="tablist"` attribute value is applied to the tab container.
- The `role="tab"` attribute value is applied to each of the tabs.
- The `aria-controls` attribute is applied to each of the tabs. The value of this attribute is retrieved from the `id` attribute of the corresponding panel.
- The `aria-selected="false"` attribute value is applied to each tab, except for the active tab that has the attribute value `aria-selected="true"`. The value of this attribute then changes dynamically to `true` when the corresponding content is displayed and `false` otherwise.
- The `<strong>` tag is added systematically around the content of the active tab. This tag is deleted as soon as the tab is no longer active.
- The `tabindex="0″` attribute is applied to each of the panels so that they can receive the focus. As soon as a tab is selected, the focus must be positioned dynamically on the corresponding panel.

The resulting code with JavaScript is as follows:

```
<ul role="tablist">
```

```
    <li role="tab" aria-selected="true" aria-controls="panel-01"><a
href="#panel-01"><strong>First tab</strong></a></li>
    <li role="tab" aria-selected="false" aria-controls="panel-02"><a
href="#panel-02">Second tab</a></li>
    <li role="tab" aria-selected="false" aria-controls="panel-03"><a
href="#panel-03">Third tab</a></li>
</ul>

<div id="panel-01" role="tabpanel" tabindex="0">
    Content of the first panel.
</div>

<div id="panel-02" role="tabpanel" tabindex="0">
    Content of the second panel.
</div>

<div id="panel-03" role="tabpanel" tabindex="0">
    Content of the third panel.
</div>
```

Demonstration

A demonstration is available in the online version of this recommendation at http://wiki.accede-web.com/en/notices/interfaces-riches-javascript/.

2.1.4. Slideshows

Purpose

A carousel or slideshow is a scrollable content panel that fits into a small space.

They generally include a system of pagination, navigation arrows, and/or a system for pausing or advancing the content in the case of a carousel with automatic scrolling.

With regards to accessibility, two types of carousel can be distinguished:

1. Carousels with automatic scrolling.
2. Carousels with manual scrolling.

Basic HTML code

An example of the basic HTML code for both types of carousel is shown below. This code should be modified according to the context, especially with respect to the content of the panels.

```
<ul id="carousel">
    <li class="panel" id="panel-1">
        Content of panel 1.
    </li>
    <li class="panel" id="panel-2">
        Content of panel 2.
    </li>
    <li class="panel" id="panel-3">
        Content of panel 3.
    </li>
</ul>
```

Without JavaScript, all the panels are displayed.

Note

The recommendations to follow when writing the JavaScript code will vary according to whether you require an automatic scrolling carousel or not.

Behaviour with JavaScript for carousels with automatic scrolling

The scripts must handle the following points:

- An element for pausing or restarting the movement of the carousel is included in the DOM, before the content of the panels.
- The title of this element changes dynamically according to whether the carousel is paused or scrolling.
- The panels that are not displayed on the screen are moved outside the screen with the CSS `position: absolute;` and `left: -999999px;`. They are not hidden with `display: none;`.
- Then, if the panels contain interactive elements, a script handles the navigation with the keyboard:
    - Only the content of the active panel is tabulable. To obtain this, apply a `tabindex="0"`, or delete the `tabindex` attribute on all the interactive elements contained in the panel (links, buttons, form fields, etc.).
    - To prevent the focus being set on the content of panels that are not displayed on the screen, a `tabindex="-1"` is applied on all the interactive elements contained in these panels.
    - This behaviour is applied each time that a new panel is displayed.

The resulting code with JavaScript is as follows:

```
<button id="pause-play">Pause</button>

<ul id="carousel">
    <li class="panel" id="panel-1">
        Content of panel 1 (hidden) with <a href="#" tabindex="-1">a 
link</a>.
    </li>
    <li class="active-panel panel" id="panel-2">
        Content of panel 2 with <a href="#">a link</a>.
    </li>
    <li class="panel" id="panel-3">
        Content of panel 3 (hidden) with <button tabindex="-1">a button
</button>.
    </li>
</ul>
```

Note

This code should be modified according to the context, especially if the pause button is an image. You would then need a code similar to this:

```
<button id="pause-play"><img src="images/pause.png" alt="Pause" /></button>
```

Warning
The pausing system (in this case, the “Play/Pause” button) must always be inserted in the DOM before the elements it interacts with, whatever its position on the screen.

Behaviour with JavaScript for carousels with manual scrolling

The scripts must handle the following points:

- An element for displaying the previous panel is included in the DOM, before the content of the panels.
- An element for displaying the following panel is included in the DOM, after the content of the panels.
- The panels that are not displayed on the screen are hidden using CSS with the property `display: none;`.

The resulting code in JavaScript is as follows:

```
<button id="previous">Previous</button>

<ul id="carousel">
    <li class="panel" id="panel-1">
        Content of panel 1 (hidden).
    </li>
    <li class="active-panel panel" id="panel-2">
        Content of panel 2.
    </li>
    <li class="panel" id="panel-3">
        Content of panel 3 (hidden).
    </li>
</ul>


<button id="next">Next</button>
```

Note
This code should be modified according to the context, especially if the “Previous” and “Next” buttons are images. In this case you need a code similar to the one below:

```
<button id="previous"><img src="images/previous.png" alt="Previous"
/></button>

[...]

<button id="next"><img src="images/next.png" alt="Next" /></button>
```

2.2. Display of information

2.2.1. Pop-in windows

Purpose

Pop-in windows appear directly on the inside of the current browser window, above the web page or the application. They are different from pop-up and pop-under windows that open new external windows, respectively in front of and behind the current window.

Pop-in windows are modal windows, in that they take control of the current page, for as long as they are displayed on the screen. To continue navigating, the user must therefore take some action in the pop-in.

Basic HTML code

Pop-in content can be included in a page using either of the two methods below (with the same result):

1. Either the content is present by default in the DOM, but hidden, before being displayed in the pop-in by a user action. In this case, make sure the reading order is logical when CSS is deactivated (see recommendation “1.2.1 Write the HTML source code by following the logical reading order” in the Accessibility guidelines for HTML and CSS).
2. Or the content is absent by default from the DOM, before being dynamically loaded in the pop-in by a user action. In this case, make sure that the content is deleted from the DOM when the pop-in is closed.

According to the choice of integration method, the behaviour will be different when JavaScript is not activated:

1. In the first case, the element that triggered the pop-in must point to the content of the pop-in, present by default in the DOM.
2. In the second case, the element that triggers the display of the pop-in must point to another page where the user can access the content of the pop-in.

JavaScript behaviour

When a pop-in is displayed on the screen:

1. The focus must be set on the container of the pop-in. To do this add a `tabindex="0"` on the container, and set the focus there dynamically in JavaScript.
2. The user must not be able to tab on the rest of the page, behind the pop-in. To do this, add a `tabindex="-1"` on all the tabulable elements on the rest of the page.
3. The `Esc` key on the keyboard must allow the user to close the pop-in.

When a pop-in is closed:

1. All the tabindex="-1" added dynamically on opening the pop-in must be deleted from the source code to re-enable page navigation from the keyboard.
2. The focus must be re-set on the element that triggered the opening of the pop-in.

2.2.2. Tooltips

Purpose

Simulated tooltips contain informative content that is displayed when hovering over an element with the mouse or the focus is set on an element. They can also be displayed when the element is activated.

They are referred to as “simulated” as they do not use the `title` attribute included in HTML, often to make them more visually attractive.

Note
In 2012, the technique based on ARIA suggested here is not operational on all screen readers currently on the market. The `aria-label` attribute should nevertheless be correctly interpreted in the near future.

While waiting for improved compatibility, the following solutions are available:

- Consider that the content proposed in the simulated tooltips is secondary (because it is accessible by other methods, for example) and implement this technique from now on.
- Consider that the content proposed in simulated tooltips is essential, and prioritise other techniques, such as, for example, adopting the recommendations for pop-in windows, which can easily be adapted for simulated tooltips.

Basic HTML code

The basic HTML code for a simulated tooltip is very simple:

```
<a href="#" aria-label="Content of the simulated tooltip">Link heading </a>
```

Warning
Without JavaScript, the proposed content in the tooltip will not be accessible for those who do not use screen readers.

You need to make sure that users can access this content, (for example, by including the content on the target page of the link), or show the content on the initial page, in the event that JavaScript is deactivated.

Behaviour with JavaScript

When the user hovers with the mouse over the element, or the focus is set on the element:

1. An empty container of type `<span>` or `<div>` is created in the DOM. This container will be used as the tooltip.
2. The attribute value for `aria-label` of the element is duplicated into the newly created container.
3. CSS files are used to position and style the tooltip as required.

When the mouse-over moves off the element or the focus is lost, the container is deleted from the DOM.

This behaviour may be applied when the element is activated. Consequently, when the element is deactivated, the simulated tooltip is then deleted from the DOM.

Demonstration

A demo is available in the online version of this document at: http://wiki.accede-web.com/en/notices/interfaces-riches-javascript/.

2.3. Forms

2.3.1. Simulated checkboxes

Purpose

Simulated checkboxes are said to be “simulated” because they do not use the default form elements in HTML, namely the `<input type=”checkbox”/>` fields. They are often used because they are more visually attractive.

Basic HTML code

The basic HTML code uses standard checkboxes:

```
<form action="index.html" method="post">
   <h2 id="question">Which sports do you play?</h2>

   <ul id="answers">
      <li class="choice">
         <label for="golf">
            <input type="checkbox" name="golf" id="golf" />
            Golf
         </label>
      </li>
      <li class="choice">
         <label for="polo">
            <input type="checkbox" name="polo" id="polo" />
            Polo
         </label>
      </li>
      <li class="choice">
         <label for="tennis">
            <input type="checkbox" name="tennis" id="tennis" />
            Tennis
         </label>
      </li>
   </ul>

   <input type="submit" value="Submit information" />
</form>
```

Without JavaScript, a standard form is provided to the user.

Behaviour with JavaScript

With JavaScript, the behaviour of simulated checkboxes must be the same as those of standard checkboxes.

The scripts must handle the following points:

- The `<label>` elements and their content are replaced by the text that was initially within the `<input type=“checkbox” />` tags.
- An image that represents an unchecked checkbox is added before each label. The technical characteristics of this this image are as follows:
  - An `aria-hidden="true"` attribute is applied to the image to hide it from users with screen readers.
  - The `alt` attribute associated with the image describes the status of the checkbox; for example `alt="Unchecked: "`.

- The `role="group"` attribute is applied to the container of the checkboxes. In the example, the `<ul>` tag is used.
- The `role="checkbox"` attribute is applied to each of the checkboxes. In the example, the `<li>` tag is used.
- The `aria-checked="false"` attribute is applied to each of the checkboxes by default. This attribute value then changes dynamically to `true` when the checkbox is selected and `false` when it is not.
- The checkboxes may be selected either by clicking on them, or by using the `space` key when the focus is set on the checkbox.
- The attribute value `tabindex="0"` is applied to each of the simulated checkboxes so that they can receive the focus.
- Lastly, if the checkboxes are preceded by a question, instruction, or any important indication for understanding how to use them, the `aria-labelledby` attribute is applied to the container of the checkboxes. The value of this attribute is retrieved from the `id` associated with the question, instruction, etc.

The resulting code in JavaScript is as follows:

```
<form method="post" action="index.html">
    <h2 id="question">Which sports do you play?</h2>

    <ul id="answers" role="group" aria-labelledby="question">
        <li class="choice" role="checkbox" aria-checked="false"
tabindex="0">
            <img aria-hidden="true" src="images/checkbox-empty.png"
alt="unchecked: ">
            Golf
        </li>
        <li class="choice" role="checkbox" aria-checked="false"
tabindex="0">
            <img aria-hidden="true" src="images/checkbox-empty.png"
alt="Unchecked : ">
            Polo
        </li>
        <li class="choice" role="checkbox" aria-checked="false"
tabindex="0">
            <img aria-hidden="true" src="images/checkbox-empty.png"
alt="Unchecked : ">
            Tennis
        </li>
    </ul>

    <input type="submit" value="Submit information">
</form>
```

Demonstration

A demo is available in the online version of this document at: http://wiki.accede-web.com/en/notices/interfaces-riches-javascript/.

2.3.2. Simulated radio buttons

Purpose

“Simulated” radio buttons do not use the standard HTML form elements for radio buttons, namely the `<input type="radio" />` fields. Simulated radio buttons are often used because they are more visually attractive.

Unlike checkboxes that permit multiple selections, only one radio button can be selected in a group of radio buttons.

Basic HTML code

The basic HTML code uses standard radio buttons:

```
<form action="index.html" method="post">
    <h2 id="question">Which colour was Henri IV’s white horse?</h2>

    <ul id="answers">
        <li class="choice">
            <label for="yellow">
                <input type="radio" name="henri-iv-horse-colour"
id="yellow" />
                Yellow
            </label>
        </li>
        <li class="choice">
            <label for="Green">
                <input type="radio" name="henri-iv-horse-colour" id="green"
/>
                Green
            </label>
        </li>
        <li class="choice">
            <label for="white">
                <input type="radio" name="henri-iv-horse-colour "
id="white" />
                White
            </label>
        </li>
    </ul>

    <input type="submit" value="Submit information" />
</form>
```

Without JavaScript, a standard form is shown to the user.

Behaviour with JavaScript

With JavaScript, the behaviour of the simulated radio buttons must be the same as for standard radio buttons.

The scripts must handle the following points:

- The `<label>` elements and their content are replaced by the text that was initially in the `<input type="radio" />` tag.
- An image that represents an unselected radio button is added before each label. This image includes the following technical features:

    - The attribute value `aria-hidden="true"` is applied to the image to hide it from users of screen readers.
    - The `alt` attribute of the image describes the status of the radio button; for example, `alt=”not selected`“.
- The `role="radiogroup"` attribute is applied to the container of the radio buttons. In the example, the `<ul>` tag is used.
- The `role="radio"` attribute is applied to each of the radio buttons. In the example, `<li>` tags are used.
- The `aria-checked="false"` is applied to each of the radio buttons by default. This attribute value then changes dynamically to `true` when the radio button is selected and `false` when it is not.
- The radio buttons can be selected either by clicking on them or by using the `space` key when the focus is set on the radio button.
- By default, the `tabindex="0"` attribute is applied to the first and last radio buttons in the group, so that the user can set the focus either at the start or the end of the group of radio buttons. The `tabindex="-1"` attribute is applied to the other radio buttons in order to prevent access from the keyboard by default.
- As soon as a radio button is selected, the `tabindex` attribute of this button is set to `0` while the value for `tabindex` is set at `-1` for the other buttons in the group.
- When the focus is positioned on one of the radio buttons:
    - Pressing on the `Up arrow` or `Left arrow` key moves the focus to the previous button, and selects it. If the focus was set on the first button in the group, then the last button in the group is selected.
    - Pressing on the `Down arrow` or `Right arrow` moves the focus to the next radio button, and selects it. If the focus was set on the last button in the group, then the first button in the group is selected.
- Finally, if the checkboxes are preceded by a question, instruction, or important indication for understanding the meaning of the checkboxes, the `aria-labelledby` attribute is applied to the container of the checkboxes. The value of this attribute is retrieved from the `id` associated with the question, instruction, etc.

The resulting code in JavaScript is as follows:

```
<form method="post" action="index.html">
    <h2 id="question">Which colour was Henri IV’s horse?</h2>

    <ul id="answers" role="radiogroup" aria-labelledby="question">
        <li class="choice" role="radio" aria-checked="false" tabindex="0">
            <img aria-hidden="true" src="images/unselected-radio-
button.png" alt=" Not selected: ">
            Yellow
        </li>
        <li class="choice" role="radio" aria-checked="false" tabindex="-1">
            <img aria-hidden="true" src="images/unselected-radio-
button.png" alt="Not selected: ">
            Green
        </li>
        <li class="choice" role="radio" aria-checked="false" tabindex="0">
            <img aria-hidden="true" src="images/unselected-radio-
button.png" alt=" Not selected: ">
            White
        </li>
    </ul>
    <input type="submit" value="Submit information">
</form>
```

Demonstration

A demo is available in the online version of this document at http://wiki.accede-web.com/en/notices/interfaces-riches-javascript/.

2.3.3. Simulated drop-down lists

Purpose

Drop-down lists referred to as “simulated” only use some of the default HTML form elements for drop-down lists, namely, the `<select>` fields and `<option>` tags. Simulated drop-down lists make use of a useful feature of CSS that makes it possible to customize a part of lists obtained.

Note
There are other more advanced techniques to simulate drop-down lists in their entirety, which give you the tools to fine tune the appearance of choice lists. These techniques are much more complicated than the solution proposed here, and are not discussed in this document.

Basic HTML code

The basic HTML code for a standard drop-down list:

```
<form action="index.html" method="post">
    <label for="eye-colour">
        Which colour are your eyes?
        <select name="eye-colour" id="eye-colour">
            <option value="black">Black</option>
            <option value="blue">Blue</option>
            <option value="green">Green</option>
            <option value="brown">Brown</option>
            <option value="other">Other</option>
        </select>
    </label>

    <input type="submit" value="Submit information" />
</form>
```

Without JavaScript, a standard form is shown to the user.

Behaviour with JavaScript

With JavaScript, the behaviour of simulated drop-down lists must be the same as those of standard drop-down lists.

To implement the essentials of this method, you need to respect the following steps:

1. The CSS property `opacity: 0;` is applied to the `<select>` tag when JavaScript is active. This property makes the list invisible by default, but the options are displayed when you click `<select>`. The list stays in the same place on the screen, since it is only made invisible by changing the opacity.

2. An empty container is created in the DOM, just after the `<select>`. For example, a `<span>` can be used. This container is then dynamically populated in JavaScript, with the current value of the drop-down list. The script is called each time the status of the drop-down list changes.
3. The invisible `<select>` is positioned in the CSS just above the created container, so that when you click on the container it triggers the opening of the drop-down list.

Going into more detail, the scripts must handle the following points:

- The CSS property `opacity: 0;` is applied to the `<select>` tags.
- An empty container is added in the DOM just after the `<select>`. In our example, a `<span>` is used.
- The `aria-hidden="true"` attribute is applied to the container, in order to hide it from users of screen readers, who otherwise would have access to the invisible `<select>`.
- The container cannot receive the focus, but a class is added to the container with the class attribute as soon as the focus is positioned on the `<select>`. This class is deleted as soon as the `<select>` loses the focus. The CSS uses this class to simulate visually that the focus has been set on the container.
- An image is included in the container using the `<img/>` tag (with the `alt` attribute left empty) to indicate that it is a drop-down list. A good accessibility practice is to use a down arrow for this element.
- As soon as the status of the list changes, the value of the container is updated. The container retrieves the value of the option selected in the drop-down list.
- The script that populates the container is always called once by default, when the page is loaded.

The resulting code in JavaScript is as follows:

```
<form method="post" action="index.html">
    <label for="eye-colour">
        Which colour are your eyes ?
        <div class="list-container">
            <select id="eye-colour" name="eye-colour">
                <option value="black">Black</option>
                <option value="blue">Blue</option>
                <option value="green">Green</option>
                <option value="brown">Brown</option>
                <option value="other">Other</option>
            </select>
            <span class="container-choice" aria-hidden="true">Black<img
alt="" src="images/arrow.png"></span>
        </div>
    </label>

    <input type="submit" value="Submit information">
</form>
```

Demonstration

A demonstration is available with the online version of this document at http://wiki.accede-web.com/en/notices/interfaces-riches-javascript/.

2.3.4. Sliders

Purpose

Sliders are form fields that let the user select a specific value within a range of predefined values. Normally, they are shown as a slider that can be moved along an axis, between the minimal value and a second value which is the maximum value.

Sometimes, you can move the slider anywhere on the scale; other times you have to choose one of the predefined values on the scale.

They are said to be “simulated” when they do not use the default form elements included in HTML, namely the `<input type=“range” />` fields. They are often used because they are more attractive.

Basic HTML code

The basic HTML code uses a standard HTML5 slider:

```
<form action="index.html" method="post">
    <label for="sleep-quality" id="sleep-quality-label">
        Are you satisfied with the quality of your sleep?
        (whole number, from 1 for “Very dissatisfied” to 5 for “Very
satisfied")
    </label>
    <input type="range" id="sleep-quality" name="sleep-quality" step="1"
value="3" min="1" max="5" />

    <input type="submit" value="Submit information" />
</form>
```

Warning

It is very important to indicate the format of data required from the user

If HTML5 is not supported by the user’s browser, and if JavaScript is not activated, a text field will be displayed by default. Without instructions, the user will not know the range of authorized values.

Without JavaScript, a standard form is shown to the user.

Behaviour with JavaScript

With JavaScript, the behaviour of simulated sliders must the same as those of standard sliders.

Essentially, the method consists of the following steps:

1. The standard slider is hidden in JavaScript.
2. An empty container is created in the DOM, in the place of the initial slider. It will contain the other slider elements. When it has been styled with CSS, it will symbolize the axis of the slider.
3. A second empty element is created in the DOM as a child of the previous element. When styled in CSS, it will be used as the slider.
4. JavaScript is then used to add ARIA roles and manage the behaviour of the simulated slider.

Going into more detail, the scripts must handle the following points:

- The CSS property `display: none;` is applied to the `<input type=“range” />` tags.
- An empty container is added in the DOM just after the `<input type=“range” />`. In the example, a `<div>` is used.

- An empty element is added in the DOM within the newly created container. In the example, a `<span>` is used. This is the slider.
- If the slider is not tabulable by default, the attribute value `tabindex="0"` is applied to it.
- The attribute value `role="slider"` is applied to the slider.
- The attribute `aria-labelledby` is applied to the slider. The value of this attribute is retrieved from the `id` attribute of the label corresponding to the slider.
- The attribute value `aria-valuemin` is applied to the slider. This attribute value corresponds to the minimum value authorized for the slider.
- The attribute value `aria-valuemax` is applied to the slider. This attribute value corresponds to the maximum value authorized for the slider.
- The attribute value `aria-valuenow` is applied to the slider. This attribute value corresponds to the current value of the slider. This attribute value is updated whenever the slider changes position.
- If required, the attribute `aria-valuetext` is applied to the slider. This attribute value corresponds to the current value of the cursor, but in a format adapted to the internet user. In our example, this attribute could, for example, have the value `aria-valuetext="Very satisfied"` when the `aria-valuenow="5"`. As it is more easily understood, it is the first value, if available, that will be shown to users of assistive devices. This attribute value is updated whenever the slider changes position.
- The slider cannot be positioned before the minimal value, or after the maximal value, on the axis.
- If steps are defined, the slider position must respect them. For example, if each step has a value of 1 unit, and the starting position is 10, it must not be possible to obtain a value that is not a whole number by using the slider.
- The slider can be moved along the axis, by dragging it with the mouse, or by dragging it with the finger.
- The slider can also be moved with the keyboard. When the focus is set on the slider:
    - Pressing the keys `Right arrow` or `Up arrow` increases the value of the slider by one step.
    - Pressing the `Left arrow` or `Down arrow` decreases the value of the slider by one step.
    - Pressing the `Home` key moves the slider to the minimal value of the slider.
    - Pressing the `End` key moves the slider to the maximal value.
    - Pressing the `Page up` key increases the value of the slider, by an arbitrary value, but by more than one step.
    - Pressing the `Page down` key decreases the value of the slider, by an arbitrary value more than one step, and equal to the amount by pressing on the `Page up` key.

The resulting code in JavaScript is as follows:

```
<form action="index.html" method="post">
    <label for="Sleep-quality" id="sleep-quality-label">
        Are you satisfied with the quality of your sleep?
        (Whole number, from 1 for “Very dissatisfied” to 5 for “Very
satisfied")
    </label>
    <input style="display: none;" type="range" id="sleep-quality"
name="sleep-quality" step="1" value="3" min="1" max="5" />
    <div id="slider-container">
        <span id="slider-cursor" role="slider" tabindex="0" aria-
labelledby="sleep-quality-label" aria-valuemin="1" aria-valuemax="5" aria-
valuenow="5" aria-valuetext="Very satisfied"></span>
```

```
    </div>

    <input type="submit" value="Submit information" />
</form>
```

Demonstration

A demo is available in the online version of this document at: http://wiki.accede-web.com/en/notices/interfaces-riches-javascript/.