

Notice d'accessibilité

interfaces riches et JavaScript

Date	Version	Auteur	État / commentaires
06/06/2013	1.5	Atalan	

En partenariat avec :

Air Liquide – Atos – BNP Paribas – Capgemini – EDF – Generali – SFR – SNCF – Société Générale – SPIE

Et le soutien de :

Agence Entreprises & Handicap – AnySurfer (*Belgique*) – Association des Paralysés de France (APF) – Association Valentin Haüy (AVH) – CIGREF – Fondation Design For All (*Espagne*) – ESSEC – Handirect – Hanploi – Sciences Po – Télécom ParisTech

AcceDe Web – www.accede-web.com	Notice d'accessibilité pour la conception graphique	Page 1/30
Réalisé par Atalan – accede@atalan.fr		Juillet 2012

Remerciements

Nous remercions tout particulièrement les 10 entreprises partenaires d'AcceDe Web et leurs équipes pour leur engagement et leur confiance :



Grâce à leur mobilisation, leur participation active aux groupes de travail, leurs relectures attentives des documents, l'utilisation test des notices sur leurs propres projets web, les entreprises partenaires ont apporté au projet AcceDe Web une expérience de terrain indispensable pour permettre la conception de notices bien adaptées aux besoins des différents intervenants d'un projet web. En permettant une diffusion libre et gratuite de ces contenus, ces entreprises contribuent à faire progresser l'accessibilité du web.

Nous remercions également les différents établissements qui nous ont apporté leur soutien et qui, dans leurs actions, sensibilisent et participent à la diffusion des notices AcceDe Web : Agence Entreprises & Handicap, AnySurfer (Belgique), Association des Paralysés de France (APF), CIGREF, ESSEC, Handirect, Hanploi, Sciences Po et Télécom ParisTech.

Nous remercions tous les membres du comité de relecture pour la qualité et la pertinence de leurs commentaires : Benjamin Ach (expert accessibilité), Vincent Aniot (APF), Jean-Baptiste Audras (Université de Grenoble), Claire Bizingre (consultante accessibilité), Victor Brito (consultant accessibilité), Anna Castilla (Provaltis), Ève Demazière (Sciences Po), Nicolas Fortin (Ministère de la Culture et de la Communication), Marc-Étienne Vargenau (Alcatel-Lucent) et Éric Vidal (Fédération des aveugles de France), ainsi que bien sûr toutes les équipes des entreprises partenaires.

Nous souhaitons enfin remercier Laurent Bracquart et Johan Ramon, Christophe Pineau et Marion Santelli qui, au sein des équipes d'Atalan, se sont impliqués sans compter pour permettre à ce projet d'être mené à bien.

*Sébastien Delorme, Sylvie Goldfain,
Atalan*

AcceDe Web – www.accede-web.com	Notice d'accessibilité pour la conception graphique	Page 2/30
Réalisé par Atalan – accede@atalan.fr		Juillet 2012

SOMMAIRE

Introduction.....	5
Contexte et objectifs.....	5
À qui s'adresse cette notice, comment l'utiliser ?	5
Licence d'utilisation.....	6
Contact.....	6
Crédits.....	7
1. Recommandations génériques interfaces riches et JavaScript	8
1.1. Différencier les boutons des liens	8
1.2. Mettre en place des alternatives à JavaScript	8
1.3. Mettre en place des alternatives à Flash	8
1.4. Permettre le contrôle des scripts à la fois à la souris et au clavier.....	9
1.5. Permettre la mise en pause des animations automatiques	10
2. Cas d'usage	12
2.1. Navigation et affichage d'informations	12
2.1.1. Menus déroulants	12
Principe.....	12
Code HTML de base.....	12
Comportement avec JavaScript	13
Démonstration	13
2.1.2. Accordéons (plier/déplier).....	13
Principe.....	13
Code HTML de base.....	13
Comportement avec JavaScript	14
Démonstration	15
2.1.3. Onglets.....	15
Principe.....	15
Code HTML de base.....	15
Comportement avec JavaScript	15
Démonstration	16
2.1.4. Carrousels (slideshows)	16
Principe.....	16
Code HTML de base.....	16
Comportement avec JavaScript pour les carrousels à défilement automatique	17
Comportement avec JavaScript pour les carrousels à défilement manuel	18
2.1.5. Fenêtres modales (pop-in).....	19
Principe.....	19
Code HTML de base.....	19
Comportement avec JavaScript	19
2.1.6. Infobulles (tooltips).....	20
Principe	20
Code HTML de base.....	20
Comportement avec JavaScript	21
Démonstration	21
2.1.7. Cases à cocher simulées (checkboxes)	21
Principe	21
Code HTML de base.....	21
Comportement avec JavaScript	22

Démonstration	23
2.1.8. Boutons radio simulés	23
Principe	23
Code HTML de base.....	23
Comportement avec JavaScript	24
Démonstration	25
2.1.9. Listes déroulantes simulées	25
Principe	25
Code HTML de base.....	26
Comportement avec JavaScript	26
</form>.....	27
Démonstration	27
2.1.10. Curseurs de défilement (sliders)	27
Principe	27
Code HTML de base.....	28
Comportement avec JavaScript	28
Démonstration	30

Contexte et objectifs

Cette notice propose à la fois :

- Les spécifications générales à prendre en compte lors du développement d'interfaces riches, quel que soit le périphérique d'affichage ciblé (ordinateur, mobile, etc.).
- Des fiches pratiques sur la manière de mettre en place de manière accessible certains cas d'usage courants (menus déroulants, onglets, etc.).

Cette notice s'inscrit dans un lot de quatre notices téléchargeables sur le site www.accede-web.com :

1. Notice d'accessibilité pour la conception graphique.
2. Notice d'accessibilité HTML/CSS.
3. **Notice d'accessibilité interfaces riches et JavaScript (présente notice).**
4. Notice d'accessibilité éditoriale (modèle).



Remarque




Les solutions proposées dans cette notice ne sont pas uniques : il s'agit souvent de solutions accessibles parmi d'autres. En d'autres termes, la plupart des solutions évoquées dans les cas d'usages peuvent être adaptées en connaissance de cause.

À qui s'adresse cette notice, comment l'utiliser ?


Ce document doit être transmis aux intervenants et/ou prestataires réalisant les spécifications techniques, les gabarits HTML/CSS et les différents développements techniques des interfaces riches. Il vient en complément aux spécifications techniques d'un projet et à la notice d'accessibilité HTML et CSS. Les recommandations peuvent être complétées ou retirées selon les contextes d'utilisation, ce travail peut être notamment réalisé par la maîtrise d'ouvrage.

Les recommandations doivent être prises en compte en phase d'intégration HTML/CSS, JavaScript, Flash et, pour certaines d'entre elles, lors de la dynamisation des pages lorsque les gabarits sont affectés à un outil de gestion de contenu (CMS).

Certaines annotations viennent compléter le document, celles-ci sont nécessaires pour la compréhension de chaque recommandation :

-  **Remarque** : les remarques permettent de compléter les recommandations en apportant des précisions applicables à des situations techniques précises et ponctuelles.
-  **Attention** : les avertissements mettent en avant des points de vigilance particuliers qu'il est important de respecter ou des pièges à éviter pour garantir une bonne accessibilité.
-  **Astuce** : les astuces ne sont pas directement liées à l'accessibilité, mais leur prise en compte permet généralement d'améliorer la qualité générale des interfaces ou de faciliter la prise en compte de l'accessibilité sur les étapes suivantes du projet. À l'inverse, notez que les recommandations de ce document, bien que dédiées à l'accessibilité, rejoignent souvent des

bonnes pratiques en matière d'ergonomie, d'expérience utilisateur, de performance ou de référencement.

-  **HTML5** : les encarts HTML5 apportent des éclaircissements aux recommandations ou des avertissements en tenant compte des évolutions proposées par la nouvelle spécification d'HTML.

Licence d'utilisation

Ce document est soumis aux termes de la licence *Creative Commons BY 3.0*¹.

Vous êtes libres :

- **de reproduire, distribuer et communiquer cette création au public,**
- **de modifier cette création,**

selon les conditions suivantes :

- **Mention de la paternité** dès lors que le document est modifié :

Vous devez mentionner clairement la mention et les logos Atalan et AcceDe Web, indiquer qu'il s'agit d'une version modifiée, et ajouter un lien vers la page où trouver l'œuvre originale : www.accede-web.com.

Vous ne devez en aucun cas citer le nom de l'auteur original d'une manière qui suggérerait qu'il vous soutient ou approuve votre utilisation de l'œuvre sans accord de sa part.

Vous ne devez en aucun cas citer les noms des entreprises partenaires (Air Liquide, Atos, BNP Paribas, Capgemini, EDF, Generali, SFR, SNCF, Société Générale et SPIE), ni ceux des soutiens (Agence Entreprises & Handicap, AnySurfer, Association des Paralysés de France (APF), CIGREF, ESSEC, Handirect, Hanploi, Sciences Po et Télécom ParisTech) sans accord de leur part.

Les marques et logos Atalan et AcceDe Web sont déposés et sont la propriété exclusive de la société Atalan. Les marques et logos des entreprises partenaires sont la propriété exclusive de Air Liquide, Atos, BNP Paribas, Capgemini, EDF, Generali, SFR, SNCF, Société Générale et SPIE.

Contact

Pour toute remarque à propos de cette notice, merci de contacter Atalan, coordinateur du projet AcceDe Web à l'adresse suivante : accede@atalan.fr.

Vous pouvez également trouver plus d'informations sur les notices méthodologiques du projet AcceDe Web sur le site www.accede-web.com ou suivre notre compte Twitter [@societe_atalan](https://twitter.com/societe_atalan).

¹ Plus d'informations sur la licence Creative Commons BY 3.0 : <http://creativecommons.org/licenses/by/3.0/fr/>.

AcceDe Web – www.accede-web.com	Notice d'accessibilité interfaces riches et JavaScript	Page 6/30
Réalisé par Atalan – accede@atalan.fr		Février 2013

Crédits

Les icônes utilisées dans les contenus des notices AcceDe Web proviennent du set d'icônes 24x24 *Free Application Icons* (<http://www.small-icons.com/packs/24x24-free-application-icons.htm>).

AcceDe Web – www.accede-web.com	Notice d'accessibilité interfaces riches et JavaScript	Page 7/30
Réalisé par Atalan – accede@atalan.fr		Février 2013

1. Recommandations génériques interfaces riches et JavaScript

1.1. Différencier les boutons des liens

Pour une meilleure compréhension des interfaces riches par les utilisateurs, les boutons sont à différencier des liens :

- Les boutons `<button>/<input>` permettent à l'utilisateur de déclencher des actions. Ils sont à employer pour soumettre des informations, pour afficher l'élément suivant ou précédent dans un carrousel, pour fermer une pop-in, etc.
- Les liens `<a>` permettent à l'utilisateur de naviguer. Ils sont à employer pour pointer sur une autre page ou sur une zone précise de la page courante par l'intermédiaire d'un système d'ancres.

1.2. Mettre en place des alternatives à JavaScript

Chaque fois que des contenus ou des fonctionnalités sont proposés par l'intermédiaire de JavaScript, penser à mettre en place une alternative. Cette alternative doit permettre d'accéder à des contenus ou des fonctionnalités similaires, lorsque JavaScript n'est pas activé sur le poste de l'utilisateur.

Voici quelques exemples d'utilisations classiques de JavaScript, avec pour chacune, une proposition d'alternative lorsque JavaScript est désactivé :

Lorsque JavaScript est activé	Lorsque JavaScript est désactivé
La gestion des erreurs d'un formulaire est effectuée côté client.	La gestion des erreurs d'un formulaire est effectuée côté serveur.
Un système d'accordéon permet le masquage et l'affichage dynamique de certains contenus.	Les contenus sont affichés les uns sous les autres dans la page.
Une carte interactive permet d'indiquer l'adresse physique d'un établissement.	L'adresse postale de l'établissement est indiquée sous forme de texte.

1.3. Mettre en place des alternatives à Flash

Chaque fois que des contenus ou des fonctionnalités sont proposés par l'intermédiaire de Flash, penser à mettre en place une alternative. Cette alternative doit permettre d'accéder à des contenus ou des fonctionnalités similaires, lorsque le lecteur Flash n'est pas installé ou activé sur le poste de l'utilisateur.

Voici quelques exemples d'utilisations classiques de Flash, avec alternatives lorsque le lecteur Flash n'est pas installé ou est désactivé :

Lorsque Flash est disponible	Lorsque Flash est indisponible
Un lecteur vidéo en Flash permet de lire les contenus vidéo.	Un lecteur vidéo en HTML5 permet de lire les contenus vidéo ou un lien de téléchargement est proposé pour chaque vidéo.

Lorsque Flash est disponible	Lorsque Flash est indisponible
Une frise chronologique en Flash retrace l'histoire d'une entreprise.	L'histoire de l'entreprise est proposée sous forme de contenu HTML.



Remarque

Une bonne pratique d'accessibilité consiste à proposer à proximité de l'animation Flash un lien vers la version alternative. L'idée est de permettre aux utilisateurs qui le souhaitent d'accéder à la version alternative, même lorsque Flash est disponible. Un lien du type « Version alternative de la frise chronologique » est, par exemple, adapté.



Attention

Si JavaScript est utilisé pour charger une animation Flash, veiller à ce que cette animation soit tout de même chargée lorsque JavaScript n'est pas activé. Il est possible que le lecteur Flash soit installé alors que JavaScript est désactivé.

1.4. Permettre le contrôle des scripts à la fois à la souris et au clavier

Chaque fois qu'un script est contrôlable par la souris, il doit être possible de contrôler ce dernier également au clavier.

C'est-à-dire que chaque fois que la souris permet de contrôler un script depuis un élément (un lien ou un bouton), celui-ci doit :

- Être atteignable au clavier.
- Permettre le contrôle du script au clavier une fois le focus placé sur l'élément.



Attention

Chaque fois que cela est possible, privilégier l'utilisation d'écouteurs d'événements génériques plutôt que spécifiques à l'appui sur une touche du clavier. Par exemple, privilégier `onfocus/onblur` à `onkeydown`.

Ceci car les raccourcis claviers ne sont pas toujours uniformes selon les systèmes ou les navigateurs. Par exemple, avec le navigateur Opera, c'est la touche `Maj`, associée aux flèches directionnelles, qui permet de naviguer de lien en lien au clavier. Contrairement à la plupart des autres navigateurs qui utilisent la touche `Tab`.



Remarque

Il est très important de veiller à l'**absence de piège au clavier**. Les pièges au clavier sont bloquants pour les utilisateurs qui naviguent au clavier car ils empêchent l'interaction avec certaines zones de la page.

Ils se manifestent lorsqu'un utilisateur se retrouve bloqué dans la page lors de la navigation au clavier :

- Soit parce qu'un élément empêche la prise de focus sur les éléments interactifs qui le précèdent ou le suivent.
- Soit parce qu'un élément ne laisse pas le focus le quitter, une fois appliqué sur l'élément.

Chaque fois que des scripts sont utilisés dans une page, veiller à ce que leur présence n'entraîne pas de pièges au clavier en testant simplement l'interface au clavier.

1.5. Permettre la mise en pause des animations automatiques

Chaque fois que des éléments de la page sont susceptibles d'être en mouvement, mettre en place un système de contrôle de ce mouvement.

La mise en place d'un bouton de lecture et de mise en pause suffit. Il peut s'agir d'un seul et même bouton mis à jour dynamiquement.

Quelques cas classiques de contenus en mouvement :

- Un carrousel d'actualités.
- Une publicité animée.
- Une vidéo.
- Etc.



Attention

Quelle que soit la position visuelle de ce bouton dans la page, celui-ci doit toujours être placé avant les éléments animés, dans le DOM. Dans le cas d'un lecteur multimédia, faire en sorte que ce bouton soit le premier élément tabulable du lecteur.



Remarque

Il n'est pas nécessaire de prévoir un système de mise en pause sur les barres de progression animées ou les éléments dont le mouvement s'arrête automatiquement dans un laps de temps inférieur à 5 secondes.

2.1. Navigation et affichage d'informations

2.1.1. Menus déroulants

Principe

Les menus déroulants sont des menus dynamiques dans lesquels seuls les items de premier niveau sont affichés par défaut, mais où des panneaux déroulants associés aux items de premier niveau peuvent être affichés au survol et à la prise de focus des items de premier niveau.



Attention

Si les items de premier niveau ne sont pas des liens, et que leur seule fonction est de permettre l'affichage et le masquage des panneaux, alors le menu déroulant doit être considéré comme un menu en accordéon.

Dans cette situation, il peut être intéressant d'adapter le script proposé pour un accordéon classique, afin que le contenu des panneaux s'affiche également au survol de la souris.



Astuce

Si beaucoup d'éléments interactifs sont présents dans les panneaux déroulants (liens, éléments de formulaire, etc.), une bonne pratique d'accessibilité consiste à privilégier le comportement d'un accordéon.

Le fait de n'afficher les panneaux déroulants que sur action de l'utilisateur sur les items de premier niveau permet d'optimiser la navigation au clavier dans la page en limitant le nombre d'éléments susceptibles de recevoir le focus, par défaut.

Code HTML de base

Le code HTML de base pour un menu déroulant peut par exemple être le suivant. Ce code est à adapter au contexte, notamment pour ce qui a trait au contenu des panneaux déroulants.

```
<ul id="menu">
  <li>
    <a href="#">Lien de premier niveau 1</a>
    <ul>
      <li><a href="#">Lien de second niveau 1</a></li>
      <li><a href="#">Lien de second niveau 2</a></li>
      <li><a href="#">Lien de second niveau 3</a></li>
    </ul>
  </li>
  <li>
    <a href="#">Lien de premier niveau 2</a>
    <ul>
      <li><a href="#">Lien de second niveau 4</a></li>
      <li><a href="#">Lien de second niveau 5</a></li>
      <li><a href="#">Lien de second niveau 6</a></li>
    </ul>
  </li>
</ul>
```

```

<li>
  <a href="#">Lien de premier niveau 3</a>
  <ul>
    <li><a href="#">Lien de second niveau 7</a></li>
    <li><a href="#">Lien de second niveau 8</a></li>
    <li><a href="#">Lien de second niveau 9</a></li>
  </ul>
</li>
</ul>

```

Sans JavaScript, la totalité des items du menu (items de premier niveau et panneaux) sont affichés.

Comportement avec JavaScript

JavaScript doit gérer les points suivants :

1. Au chargement du script, les panneaux sont sortis de l'écran en CSS via `position: absolute; et left: -999999px;`.
2. Puis :
 - Lorsqu'un item de premier niveau prend le focus ou est survolé, le panneau associé à l'item concerné est affiché à l'écran.
 - Lorsqu'un item de premier niveau perd le focus ou n'est plus survolé, le panneau associé à l'item concerné est à nouveau sorti de l'écran.

Démonstration

Une démonstration est disponible sur la version en ligne de cette recommandation : <http://wiki.accede-web.com/notices/interfaces-riches-javascript/>.

2.1.2. Accordéons (plier/déplier)

Principe

Les accordéons sont des modules dynamiques qui permettent d'optimiser l'affichage du contenu d'une page, dans un espace réduit. Ils utilisent un système de "plier/déplier", contrôlable par l'utilisateur, souvent au clic sur un titre ou un lien qui surplombent le bloc de contenu.

Code HTML de base

Le code HTML de base pour un accordéon dont les panneaux sont masqués par défaut peut par exemple être le suivant. Ce code est à adapter au contexte, notamment pour ce qui a trait aux niveaux de titres.

```

<h1>Titre associé au premier panneau</h1>
<div id="panneau-1">
  <p>Contenu du premier panneau</p>
</div>

<h1>Titre associé au deuxième panneau</h1>
<div id="panneau-2">
  <p>Contenu du deuxième panneau</p>
</div>

<h1>Titre associé au troisième panneau</h1>
<div id="panneau-3">

```

```
<p>Contenu du troisième panneau</p>
</div>
```

Sans JavaScript, l'ensemble du système est déplié (affiché).



Remarque

Même si cela est moins recommandé, il est possible, sans JavaScript, de conserver les liens à l'intérieur des titres, dans ce cas, il s'agit de liens pointant vers les ancres correspondantes :

```
<h1><a href="#panneau-1">Titre associé au premier panneau</a></h1>
<div id="panneau-1">
  <p>Contenu du premier panneau</p>
</div>
```

Comportement avec JavaScript

Sur cette base, les scripts doivent ensuite gérer les points suivants :

- Les titres des panneaux sont agrémentés de liens qui permettront de plier ou déplier les panneaux.
- Les panneaux sont masqués par défaut à l'aide de `display: none;` via CSS.
- Actionner à la souris ou au clavier un lien-titre affiche et masque alternativement le panneau associé (`display: none;` pour le masquage et `display: block;` pour l'affichage).
- L'attribut `aria-expanded="false"` est appliqué par défaut sur le lien-titre. La valeur de cet attribut change ensuite dynamiquement : `true` lorsque le contenu associé est affiché et `false` dans le cas contraire.
- L'attribut `aria-controls` est appliqué sur le lien-titre. La valeur de cet attribut reprend celle de l'attribut `id` du panneau associé.

Le code ainsi obtenu avec JavaScript sera le suivant :

```
<h1><a href="#panneau-1" aria-expanded="false" aria-controls="panneau-1">Titre associé au premier panneau</a></h1>
<div id="panneau-1">
  <p>Contenu du premier panneau (masqué)</p>
</div>

<h1><a href="#panneau-2" aria-expanded="true" aria-controls="panneau-2">Titre associé au deuxième panneau</a></h1>
<div id="panneau-2">
  <p>Contenu du deuxième panneau (affiché)</p>
</div>

<h1><a href="#panneau-3" aria-expanded="false" aria-controls="panneau-3">Titre associé au troisième panneau</a></h1>
<div id="panneau-3">
  <p>Contenu du troisième panneau (masqué)</p>
</div>
```

Démonstration

Une démonstration est disponible sur la version en ligne de cette recommandation : <http://wiki.accede-web.com/notices/interfaces-riches-javascript/>.

2.1.3. Onglets

Principe

Les onglets sont des modules dynamiques qui permettent d'optimiser l'affichage du contenu d'une page dans un espace réduit. Ils se présentent généralement sous la forme d'une liste de liens accolés, qui permettent d'afficher au clic du contenu relatif à l'onglet sélectionné. Un seul onglet peut être activé à la fois.

Code HTML de base

Le code HTML de base est constitué de liens internes qui pointent vers leurs panneaux respectifs.

```
<ul>
  <li><a href="#panneau-01">Premier onglet</a></li>
  <li><a href="#panneau-02">Deuxième onglet</a></li>
  <li><a href="#panneau-03">Troisième onglet</a></li>
</ul>

<div id="panneau-01">
  Contenu du premier panneau.
</div>

<div id="panneau-02">
  Contenu du deuxième panneau.
</div>

<div id="panneau-03">
  Contenu du troisième panneau.
</div>
```

Sans JavaScript, la totalité des panneaux sont affichés, et les onglets servent de liens d'accès direct au contenu du panneau associé.

Comportement avec JavaScript

Sur cette base, les scripts doivent ensuite gérer les points suivants :

- À l'exception du panneau actif, les panneaux sont masqués par défaut à l'aide de `display: none;` via CSS.
- L'attribut `role="tabpanel"` est appliqué sur chacun des panneaux.
- L'attribut `role="tablist"` est appliqué sur le conteneur des onglets.
- L'attribut `role="tab"` est appliqué sur chacun des onglets.
- L'attribut `aria-controls` est appliqué sur chacun des onglets. La valeur de cet attribut reprend celle de l'attribut `id` du panneau associé.
- L'attribut `aria-selected="false"` est appliqué par défaut sur chaque onglet, à l'exception de l'onglet actif qui reçoit l'attribut `aria-selected="true"`. La valeur de cet attribut change ensuite dynamiquement : `true` lorsque le contenu associé est affiché et `false` dans le cas contraire.

AcceDe Web – www.accede-web.com	Notice d'accessibilité interfaces riches et JavaScript	Page 15/30
Réalisé par Atalan – accede@atalan.fr		Février 2013

- Une balise `` est rajoutée systématiquement autour du contenu de l'onglet actif. Cette balise est supprimée dès que l'onglet n'est plus actif.
- L'attribut `tabindex="0"` est appliqué sur chacun des panneaux afin de permettre à ces derniers de recevoir le focus. Dès qu'un onglet est sélectionné, le focus doit être positionné dynamiquement sur le panneau associé.

Le code ainsi obtenu avec JavaScript sera le suivant :

```
<ul role="tablist">
  <li role="tab" aria-selected="true" aria-controls="panneau-01"><a
href="#panneau-01"><strong>Premier onglet</strong></a></li>
  <li role="tab" aria-selected="false" aria-controls="panneau-02"><a
href="#panneau-02">Deuxième onglet</a></li>
  <li role="tab" aria-selected="false" aria-controls="panneau-03"><a
href="#panneau-03">Troisième onglet</a></li>
</ul>

<div id="panneau-01" role="tabpanel" tabindex="0">
  Contenu du premier panneau.
</div>

<div id="panneau-02" role="tabpanel" tabindex="0">
  Contenu du deuxième panneau.
</div>

<div id="panneau-03" role="tabpanel" tabindex="0">
  Contenu du troisième panneau.
</div>
```

Démonstration

Une démonstration est disponible sur la version en ligne de cette recommandation : <http://wiki.accede-web.com/notices/interfaces-riches-javascript/>.

2.1.4. Carrousels (slideshows)

Principe

Les carrousels (ou slideshows) sont des modules qui permettent l'affichage de panneaux de contenus défilants dans une zone réduite.

Ils sont généralement agrémentés d'un système de pagination, de flèches de navigation, et/ou encore d'un système de mise en pause et de relance du mouvement dans le cas d'un carrousel à défilement automatique.

Dans le cadre d'une mise en accessibilité, deux types de carrousels sont principalement à distinguer :

1. Les carrousels à défilement automatique.
2. Les carrousels à défilement manuel.

Code HTML de base

Le code HTML de base pour les deux types de carrousels peut par exemple être le suivant. Ce code est à adapter au contexte, notamment pour ce qui a trait au contenu des panneaux.

AcceDe Web – www.accede-web.com	Notice d'accessibilité interfaces riches et JavaScript	Page 16/30
Réalisé par Atalan – accede@atalan.fr		Février 2013


```

<ul id="carrousel">
  <li class="panneau" id="panneau-1">
    Contenu du panneau 1.
  </li>
  <li class="panneau" id="panneau-2">
    Contenu du panneau 2.
  </li>
  <li class="panneau" id="panneau-3">
    Contenu du panneau 3.
  </li>
</ul>

```

Sans JavaScript, tous les panneaux sont affichés.



Remarque

Les recommandations à suivre lors de l'écriture du code JavaScript seront ensuite différentes selon le type de carrousel souhaité : défilement automatique ou non.

Comportement avec JavaScript pour les carrousels à défilement automatique

Les scripts doivent gérer les points suivants :

- Un élément qui permet la mise en pause et la relance de l'animation du carrousel est intégré dans le DOM, avant le contenu des panneaux.
- L'intitulé de cet élément change dynamiquement selon que le carrousel soit en position de lecture ou de pause.
- Les panneaux qui ne sont pas affichés à l'écran sont sortis de l'écran en CSS via `position: absolute; et left: -999999px;`. Ils ne sont pas masqués avec `display: none;`.
- Puis, si les panneaux contiennent des éléments interactifs, un script gère la navigation au clavier :
 - Seul le contenu du panneau actif est tabulable. Appliquer pour cela un `tabindex="0"`, ou supprimer les attributs `tabindex`, sur l'ensemble des éléments interactifs contenus dans le panneau (liens, boutons, champs de formulaires, etc.).
 - Pour empêcher la prise de focus dans le contenu des panneaux non affichés à l'écran, un `tabindex="-1"` est appliqué sur l'ensemble des éléments interactifs contenus dans ces derniers.
 - Ce comportement est appliqué à chaque fois qu'un nouveau panneau est affiché.

Le code ainsi obtenu avec JavaScript sera le suivant :

```

<button id="lecture-pause">Pause</button>

<ul id="carrousel">
  <li class="panneau" id="panneau-1">
    Contenu du panneau 1 (masqué) avec <a href="#" tabindex="-1">un
    lien</a>.
  </li>
  <li class="panneau panneau-actif" id="panneau-2">
    Contenu du panneau 2 avec <a href="#">un lien</a>.
  </li>
  <li class="panneau" id="panneau-3">

```

```
Contenu du panneau 3 (masqué) avec <button tabindex="-1">un
bouton</button>.
</li>
</ul>
```



Remarque

Ce code est à adapter au contexte, notamment si le bouton de mise en pause est en fait une image. Il faudra alors privilégier un code du type :

```
<button id="lecture-pause"></button>
```



Attention

Le système de mise en pause (ici, le bouton « Lecture/Pause ») doit toujours être placé avant les éléments sur lesquels il agit, dans le DOM, quelle que soit sa position à l'écran.

Comportement avec JavaScript pour les carrousels à défilement manuel

Les scripts doivent gérer les points suivants :

- Un élément qui permet d'afficher le panneau précédent est intégré dans le DOM, avant le contenu des panneaux.
- Un élément qui permet d'afficher le panneau suivant est intégré dans le DOM, après le contenu des panneaux.
- Les panneaux qui ne sont pas affichés à l'écran sont masqués à l'écran en CSS via `display: none;`.

Le code ainsi obtenu avec JavaScript sera le suivant :

```
<button id="precedent">Précédent</button>

<ul id="carrousel">
  <li class="panneau" id="panneau-1">
    Contenu du panneau 1 (masqué).
  </li>
  <li class="panneau panneau-actif" id="panneau-2">
    Contenu du panneau 2.
  </li>
  <li class="panneau" id="panneau-3">
    Contenu du panneau 3 (masqué).
  </li>
</ul>

<button id="suivant">Suivant</button>
```



Remarque

Ce code est à adapter au contexte, notamment si les boutons « Précédent » et « Suivant » sont en fait des images. Il faudra alors privilégier un code du type :

```
<button id="precedent"></button>

[ ... ]

<button id="suivant"></button>
```

2.1.5. Fenêtres modales (pop-in)

Principe

Les pop-in sont des fenêtres qui apparaissent directement à l'intérieur de la fenêtre courante du navigateur, au-dessus de la page web ou de l'application. Elles sont à opposer aux pop-up et pop-under, qui déclenchent respectivement l'ouverture de nouvelles fenêtres externes, par-dessus et par-dessous la fenêtre courante.

Il s'agit de fenêtres modales, c'est-à-dire de fenêtres qui prennent le contrôle de la page courante, tant qu'elles sont affichées à l'écran. Pour poursuivre sa navigation, l'utilisateur doit donc nécessairement effectuer une action dans la pop-in.

Code HTML de base

Le contenu des pop-in peut être intégré dans la page, indifféremment, de deux façons :

1. Soit le contenu est présent par défaut dans le DOM, mais masqué, avant d'être affiché dans la pop-in sur action de l'utilisateur. Il faut alors veiller à ce que l'ordre de lecture reste cohérent, lorsque les CSS sont désactivées (cf. recommandation « 1.2.1 Écrire le code HTML en suivant la logique de l'ordre de lecture » dans la notice d'accessibilité HTML et CSS).
2. Soit le contenu est absent par défaut du DOM, avant d'être chargé dynamiquement dans la pop-in sur action de l'utilisateur. Il faut alors veiller à ce que le contenu soit supprimé à nouveau du DOM lorsque la pop-in se referme.

Selon le mode d'intégration privilégié, le comportement sera différent, lorsque JavaScript n'est pas activé :

1. Dans le premier cas, l'élément qui déclenche l'apparition de la pop-in devra pointer sur le contenu de la pop-in, présent par défaut dans le DOM.
2. Dans le second cas, l'élément qui déclenche l'apparition de la pop-in devra pointer sur une autre page où l'utilisateur pourra accéder au contenu de la pop-in.

Comportement avec JavaScript

Lorsqu'une pop-in est affichée à l'écran :

AcceDe Web – www.accede-web.com	Notice d'accessibilité interfaces riches et JavaScript	Page 19/30
Réalisé par Atalan – accede@atalan.fr		Février 2013

1. Le focus doit être placé au niveau du conteneur de la pop-in. Rajouter pour cela un `tabindex="0"` sur le conteneur, et y placer dynamiquement le focus en JavaScript.
2. L'utilisateur ne doit pas pouvoir tabuler sur le reste de la page, à l'arrière de la pop-in. Rajouter pour cela un `tabindex="-1"` sur tous les éléments tabulables du reste de la page.
3. La touche `Echap` du clavier doit permettre à l'utilisateur de fermer la pop-in.

Lorsqu'une pop-in est fermée :

1. Tous les `tabindex="-1"` rajoutés dynamiquement à l'ouverture doivent être supprimés du code source pour permettre à nouveau la navigation au clavier dans la page.
2. Le focus doit être déplacé au niveau de l'élément qui a déclenché l'ouverture de la pop-in.

2.1.6. Infobulles (tooltips)

Principe

Les infobulles simulées sont des contenus informatifs qui apparaissent à l'écran au survol et à la prise de focus d'un élément. Elles peuvent également apparaître lorsque l'élément est activé.

Elles sont dites « simulées », car elles n'utilisent pas l'attribut `title` proposé par HTML, souvent pour des raisons esthétiques.



Remarque

En 2012, la technique à base d'ARIA proposée ici n'est pas exploitable par tous les lecteurs d'écran du marché. L'attribut `aria-label` devrait toutefois être correctement interprété dans un futur proche.

Dans l'attente d'une meilleure compatibilité, plusieurs solutions, au choix :

- Considérer que les contenus proposés dans les infobulles simulées sont secondaires (car accessibles par un autre moyen, par exemple), et mettre en place cette technique dès aujourd'hui.
- Considérer que les contenus proposés dans les infobulles simulées sont essentiels, et privilégier alors d'autres techniques, comme par exemple les recommandations associées aux fenêtres modales (pop-in), qui peuvent être adaptées facilement au cas des infobulles simulées.

Code HTML de base

Le code HTML de base pour une infobulle simulée accessible est très simple :

```
<a href="#" aria-label="Contenu de l'infobulle simulée">Intitulé du lien</a>
```



Attention

Sans JavaScript, le contenu proposé dans l'infobulle ne sera pas accessible pour les personnes qui n'utilisent pas de lecteurs d'écran.

S'assurer qu'un moyen d'accéder à ce contenu reste présent (par exemple, en proposant le contenu sur la page cible du lien) ou que ce contenu soit affiché dans la page initiale, dans le cas où JavaScript est désactivé.

Comportement avec JavaScript

Lorsque l'élément est survolé ou que le focus est positionné sur ce dernier :

1. Un conteneur vide du type `` ou `<div>` est créé dans le DOM. Ce conteneur servira d'infobulle.
2. La valeur de l'attribut `aria-label` de l'élément est dupliquée dans le conteneur nouvellement créé.
3. Les CSS sont utilisées pour positionner et styler l'infobulle tel qu'attendu.

Lorsque l'élément n'est plus survolé ou que le focus est perdu, le conteneur est supprimé du DOM.

Ce comportement peut éventuellement être appliqué lorsque l'élément est activé. La désactivation de l'élément aura alors pour conséquence de supprimer l'infobulle simulée du DOM.

Démonstration

Une démonstration est disponible sur la version en ligne de cette recommandation : <http://wiki.accede-web.com/notices/interfaces-riches-javascript/>.

2.1.7.Cases à cocher simulées (checkboxes)

Principe

Les checkboxes simulées sont des cases à cocher dites « simulées » car elles n'utilisent pas les éléments de formulaires proposés par défaut par HTML, à savoir les champs `<input type="checkbox" />`. Ce choix est souvent lié à des raisons esthétiques.

Code HTML de base

Le code HTML de base utilise les cases à cocher classiques :

```
<form action="index.html" method="post">
  <h2 id="question">Quel sport pratiquez-vous ?</h2>

  <ul id="reponses">
    <li class="choix">
      <label for="peche">
        <input type="checkbox" name="peche" id="peche" />
        La pêche
      </label>
    </li>
    <li class="choix">
      <label for="polo">
```

```

        <input type="checkbox" name="polo" id="polo" />
        Le polo
    </label>
</li>
<li class="choix">
    <label for="diabolo">
        <input type="checkbox" name="diabolo" id="diabolo" />
        Le diabolo
    </label>
</li>
</ul>

<input type="submit" value="Transmettre les informations" />
</form>

```

Sans JavaScript, un formulaire classique est proposé à l'utilisateur.

Comportement avec JavaScript

Avec JavaScript, le comportement des checkboxes simulées doit être le même que celui des cases à cocher classiques.

Les scripts doivent gérer les points suivants :

- Les `<label>` et leur contenu sont remplacés par du texte, celui qui était initialement associé aux `<input type="checkbox" />`.
- Une image qui représente une case non cochée est rajoutée avant chaque étiquette. Les caractéristiques techniques de cette image sont les suivantes :
 - Un attribut `aria-hidden="true"` est appliqué sur l'image pour la masquer aux utilisateurs de lecteurs d'écran.
 - L'attribut `alt` de l'image décrit l'état de la case à cocher, par exemple, `alt="Non coché : "`.
- L'attribut `role="group"` est appliqué sur le conteneur des cases à cocher. Il s'agit du `` dans l'exemple proposé.
- L'attribut `role="checkbox"` est appliqué sur chacune des cases à cocher. Il s'agit des `` dans l'exemple proposé.
- L'attribut `aria-checked="false"` est appliqué sur chacune des cases à cocher par défaut. La valeur de cet attribut change ensuite dynamiquement : `true` lorsque la case est cochée et `false` dans le cas contraire.
- Les cases peuvent être cochées à la fois au clic et à la pression sur la touche `Espace`, lorsque le focus est positionné sur une case à cocher.
- L'attribut `tabindex="0"` est appliqué sur chacune des cases à cocher simulées afin de permettre à ces dernières de recevoir le focus.
- Enfin, si les cases à cocher sont précédées d'une question, d'une indication, ou d'une quelconque mention importante à la compréhension de la fonction de ces dernières, l'attribut `aria-labelledby` est appliqué sur le conteneur des cases à cocher. La valeur de cet attribut reprend la valeur de l'id associé à la question, l'indication, etc.

Le code ainsi obtenu avec JavaScript sera le suivant :

```

<form method="post" action="index.html">
    <h2 id="question">Quel sport pratiquez-vous ?</h2>

```

AcceDe Web – www.accede-web.com	Notice d'accessibilité interfaces riches et JavaScript	Page 22/30
Réalisé par Atalan – accede@atalan.fr		Février 2013

```

        <ul id="reponses" role="group" aria-labelledby="question">
          <li class="choix" role="checkbox" aria-checked="false"
tabindex="0">
            
            La pêche
          </li>
          <li class="choix" role="checkbox" aria-checked="false"
tabindex="0">
            
            Le polo
          </li>
          <li class="choix" role="checkbox" aria-checked="false"
tabindex="0">
            
            Le diabolo
          </li>
        </ul>

        <input type="submit" value="Transmettre les informations">
</form>

```

Démonstration

Une démonstration est disponible sur la version en ligne de cette recommandation : <http://wiki.accede-web.com/notices/interfaces-riches-javascript/>.

2.1.8. Boutons radio simulés

Principe

Les boutons radio dits « simulés » n'utilisent pas les éléments de formulaires proposés par défaut par HTML, à savoir les champs `<input type="radio" />`. Ce choix est souvent lié à des raisons esthétiques.

Contrairement aux checkboxes qui autorisent des choix multiples, un seul bouton radio peut être activé à la fois par groupe de boutons radio.

Code HTML de base

Le code HTML de base utilise les boutons radio classiques :

```

<form action="index.html" method="post">
  <h2 id="question">Quelle est la couleur du cheval blanc d'Henri IV
?</h2>

  <ul id="reponses">
    <li class="choix">
      <label for="jaune">
        <input type="radio" name="couleur-cheval-henri-iv"
id="jaune" />
        Jaune
      </label>
    </li>
  </ul>

```

```

        <li class="choix">
            <label for="vert">
                <input type="radio" name="couleur-cheval-henri-iv"
id="vert" />
                Vert
            </label>
        </li>
        <li class="choix">
            <label for="blanc">
                <input type="radio" name="couleur-cheval-henri-iv"
id="blanc" />
                Blanc
            </label>
        </li>
    </ul>

    <input type="submit" value="Transmettre les informations" />
</form>

```

Sans JavaScript, un formulaire classique est proposé à l'utilisateur.

Comportement avec JavaScript

Avec JavaScript, le comportement des checkboxes simulées doit être le même que celui des cases à cocher classiques.

Les scripts doivent gérer les points suivants :

- Les `<label>` et leur contenu sont remplacés par du texte, celui qui était initialement associé aux `<input type="radio" />`.
- Une image qui représente un bouton radio coché est rajoutée avant chaque étiquette. Les caractéristiques techniques de cette image sont les suivantes :
 - Un attribut `aria-hidden="true"` est appliqué sur l'image pour la masquer aux utilisateurs de lecteurs d'écran.
 - L'attribut `alt` de l'image décrit l'état du bouton radio, par exemple, `alt="Non coché : "`.
- L'attribut `role="radiogroup"` est appliqué sur le conteneur des boutons radio. Il s'agit du `` dans l'exemple proposé.
- L'attribut `role="radio"` est appliqué sur chacun des boutons radio. Il s'agit des `` dans l'exemple proposé.
- L'attribut `aria-checked="false"` est appliqué sur chacun des boutons radio par défaut. La valeur de cet attribut change ensuite dynamiquement : `true` lorsque le bouton radio est coché et `false` dans le cas contraire.
- Les boutons radio peuvent être cochés à la fois au clic et à la pression sur la touche `Espace`, lorsque le focus est positionné sur un bouton radio.
- Par défaut, l'attribut `tabindex="0"` est appliqué sur le premier et le dernier bouton radio du groupe, afin de permettre l'entrée du focus dans le groupe de boutons radio à la fois dans l'ordre de lecture classique et dans l'ordre inverse. L'attribut `tabindex="-1"` est appliqué sur les autres boutons radio afin d'en interdire par défaut l'accès au clavier.
- Dès lors qu'un bouton radio est coché, l'attribut `tabindex` de ce bouton passe à 0 tandis que la valeur du `tabindex` passe à -1 pour le reste des boutons du groupe.
- Lorsque le focus est positionné sur l'un des boutons radio :

AcceDe Web – www.accede-web.com	Notice d'accessibilité interfaces riches et JavaScript	Page 24/30
Réalisé par Atalan – accede@atalan.fr		Février 2013

- Appuyer sur la touche `Flèche haut` ou `Flèche gauche` déplace le focus et coche le bouton radio précédent. Si le focus était positionné au niveau du premier bouton du groupe au moment de l'action, c'est le dernier bouton du groupe qui est coché.
- Appuyer sur la touche `Flèche bas` ou `Flèche droite` déplace le focus et coche le bouton radio suivant. Si le focus était positionné au niveau du dernier bouton du groupe au moment de l'action, c'est le premier bouton du groupe qui est coché.
- Enfin, si les cases à cocher sont précédées d'une question, d'une indication, ou d'une quelconque mention importante à la compréhension de la fonction de ces dernières, l'attribut `aria-labelledby` est appliqué sur le conteneur des cases à cocher. La valeur de cet attribut reprend la valeur de l'`id` associé à la question, l'indication, etc.

Le code ainsi obtenu avec JavaScript sera le suivant :

```
<form method="post" action="index.html">
  <h2 id="question">Quelle est la couleur du cheval blanc d'Henri IV
?</h2>

  <ul id="reponses" role="radiogroup" aria-labelledby="question">
    <li class="choix" role="radio" aria-checked="false" tabindex="0">
      
      Jaune
    </li>
    <li class="choix" role="radio" aria-checked="false" tabindex="-1">
      
      Vert
    </li>
    <li class="choix" role="radio" aria-checked="false" tabindex="0">
      
      Blanc
    </li>
  </ul>

  <input type="submit" value="Transmettre les informations">
</form>
```

Démonstration

Une démonstration est disponible sur la version en ligne de cette recommandation : <http://wiki.accede-web.com/notices/interfaces-riches-javascript/>.

2.1.9.Listes déroulantes simulées

Principe

Les listes déroulantes dites « simulées » n'utilisent que partiellement les éléments de formulaires proposés par défaut par HTML, à savoir les champs `<select>` et les balises `<option>`. Elles exploitent une astuce CSS, qui permet ensuite de customiser en partie les listes obtenues.

AcceDe Web – www.accede-web.com	Notice d'accessibilité interfaces riches et JavaScript	Page 25/30
Réalisé par Atalan – accede@atalan.fr		Février 2013



Remarque

Il existe d'autres techniques plus avancées pour simuler totalement des listes déroulantes, qui permettent notamment un contrôle très fin de l'apparence des listes de choix. Ces techniques sont beaucoup plus lourdes que la solution proposée ici, elles ne sont pas abordées dans ce document.

Code HTML de base

Le code HTML de base utilise une liste déroulante classique :

```
<form action="index.html" method="post">
  <label for="couleur-yeux">
    Quelle est la couleur de vos yeux ?
    <select name="couleur-yeux" id="couleur-yeux">
      <option value="noirs">Noirs</option>
      <option value="bleus">Bleus</option>
      <option value="verts">Verts</option>
      <option value="marrons">Marrons</option>
      <option value="autres">Autres</option>
    </select>
  </label>

  <input type="submit" value="Transmettre les informations" />
</form>
```

Sans JavaScript, un formulaire classique est proposé à l'utilisateur.

Comportement avec JavaScript

Avec JavaScript, le comportement des listes déroulantes simulées doit être le même que celui des listes déroulantes classiques.

La méthode consiste, dans les grandes lignes, à respecter les étapes suivantes :

1. La propriété CSS `opacity: 0;` est appliquée sur la balise `<select>` lorsque JavaScript est activé. Cette propriété a pour effet de rendre la liste invisible par défaut, au détail près que les options continuent d'apparaître au clic sur le `<select>`, qui conserve sa place à l'écran, puisqu'il est seulement rendu invisible en jouant sur l'opacité.
2. Un conteneur vide est créé dans le DOM, juste après le `<select>`. Par exemple un ``. Ce conteneur est ensuite peuplé dynamiquement en JavaScript, avec la valeur courante de la liste déroulante. Le script est appelé à chaque fois que l'état de la liste déroulante change.
3. Le `<select>` invisible est positionné en CSS juste au-dessus du conteneur créé, de manière à ce qu'un clic sur le conteneur déclenche l'ouverture de la liste déroulante.

Plus précisément, les scripts doivent gérer les points suivants :

- La propriété CSS `opacity: 0;` est appliquée sur le `<select>`.
- Un conteneur vide est ajouté dans le DOM juste après le `<select>`. Dans l'exemple, un `` est utilisé.
- L'attribut `aria-hidden="true"` est appliqué sur ce conteneur, afin de le masquer aux utilisateurs de lecteurs d'écran, qui auront eux accès au `<select>` invisible.

- Le conteneur ne prend jamais le focus, mais une classe est rajoutée sur ce dernier via l'attribut class, dès lors que le focus est positionné sur le <select>. Cette classe est supprimée dès que le <select> perd le focus. CSS utilise cette classe pour simuler visuellement la prise de focus sur le conteneur.
- Une image est intégrée dans le conteneur par l'intermédiaire d'une balise avec attribut alt vide, pour indiquer aux utilisateurs qu'il s'agit d'une liste déroulante. Une bonne pratique d'accessibilité consiste à utiliser une flèche orientée vers le bas pour cet élément.
- Dès que l'état de la liste change, la valeur du conteneur est mise à jour. Elle reprend la valeur de l'option sélectionnée dans la liste déroulante.
- Le script qui peuple le conteneur est toujours appelé une fois par défaut, au chargement de la page.

Le code ainsi obtenu avec JavaScript sera le suivant :

```
<form method="post" action="index.html">
  <label for="couleur-yeux">
    Quelle est la couleur de vos yeux ?
    <div class="conteneur-liste">
      <select id="couleur-yeux" name="couleur-yeux">
        <option value="noirs">Noirs</option>
        <option value="bleus">Bleus</option>
        <option value="verts">Verts</option>
        <option value="marrons">Marrons</option>
        <option value="autres">Autres</option>
      </select>
      <span class="conteneur-choix" aria-hidden="true">Noirs</span>
    </div>
  </label>

  <input type="submit" value="Transmettre les informations">
</form>
```

Démonstration

Une démonstration est disponible sur la version en ligne de cette recommandation : <http://wiki.accede-web.com/notices/interfaces-riches-javascript/>.

2.1.10. Curseurs de défilement (sliders)

Principe

Les sliders sont des champs de formulaires qui permettent de sélectionner une valeur précise à l'intérieur d'une gamme de valeurs définie au préalable. Ils prennent habituellement l'apparence d'un curseur à déplacer le long d'un axe, entre une première valeur minimale et une seconde valeur maximale.

Parfois, le contrôle du curseur est libre au sein de cet axe, parfois, il est contraint à une série de valeurs prédéfinies.

Ils sont dits « simulés » lorsqu'ils n'utilisent pas les éléments de formulaires proposés par défaut par HTML, à savoir les champs <input type="range" />. Ce choix est souvent lié à des raisons esthétiques.

AcceDe Web – www.accede-web.com	Notice d'accessibilité interfaces riches et JavaScript	Page 27/30
Réalisé par Atalan – accede@atalan.fr		Février 2013

Code HTML de base

Le code HTML de base utilise un slider HTML5 classique :

```
<form action="index.html" method="post">
  <label for="qualite-sommeil" id="label-qualite-sommeil">
    Êtes-vous satisfait(e) de la qualité de votre sommeil ?
    (nombre entier, de 1 pour "Très insatisfait(e)" à 5 pour "Très
satisfait")
  </label>
  <input type="range" id="qualite-sommeil" name="qualite-sommeil"
step="1" value="3" min="1" max="5" />

  <input type="submit" value="Transmettre les informations" />
</form>
```



Attention

Il est très important d'indiquer à l'utilisateur le format des données attendues.

En l'absence de support de HTML5 par le navigateur de ce dernier, et si JavaScript n'est pas activé, un champ de texte sera affiché par défaut. Sans indications, l'utilisateur ne pourra pas connaître l'étendue des valeurs autorisées.

Sans JavaScript, un formulaire classique est proposé à l'utilisateur.

Comportement avec JavaScript

Avec JavaScript, le comportement des sliders simulées doit être le même que celui des sliders classiques.

La méthode consiste, dans les grandes lignes, à respecter les étapes suivantes :

1. Le slider classique est masqué en JavaScript.
2. Un conteneur vide est créé dans le DOM, à la place du slider initial. Il servira à contenir les éléments suivants. Une fois stylé en CSS, il symbolisera l'axe du slider.
3. Un deuxième élément vide est créé dans le DOM, enfant de l'élément précédent. Il servira quant à lui à symboliser le curseur, une fois stylé en CSS.
4. JavaScript intervient ensuite pour rajouter des rôles ARIA et gérer le comportement du slider simulé.

Plus précisément, les scripts doivent gérer les points suivants :

- La propriété CSS `display: none;` est appliquée sur le `<input type="range" />`.
- Un conteneur vide est ajouté dans le DOM juste après le `<input type="range" />`. Dans l'exemple, un `<div>` est utilisé.
- Un élément vide est ajouté dans le DOM à l'intérieur du conteneur nouvellement créé. Dans l'exemple, un `` est utilisé. Il s'agit du curseur.
- Si le curseur n'est pas un élément tabulable par défaut, l'attribut `tabindex="0"` est appliqué sur ce dernier.
- L'attribut `role="slider"` est appliqué sur le curseur.

- L'attribut `aria-labelledby` est appliqué sur le curseur. La valeur de cet attribut reprend celle de l'attribut `id` de l'étiquette associée au slider.
- L'attribut `aria-valuemin` est appliqué sur le curseur. La valeur de cet attribut reprend la valeur minimale autorisée pour le curseur.
- L'attribut `aria-valuemax` est appliqué sur le curseur. La valeur de cet attribut reprend la valeur maximale autorisée pour le curseur.
- L'attribut `aria-valuenow` est appliqué sur le curseur. La valeur de cet attribut reprend la valeur courante du curseur. La valeur de cet attribut est mise à jour dès lors que le curseur change de position.
- Si besoin, l'attribut `aria-valuetext` est appliqué sur le curseur. La valeur de cet attribut exprime la valeur courante du curseur, mais dans un format adapté à l'internaute. Dans l'exemple, cet attribut peut par exemple être renseigné tel que `aria-valuetext="Très satisfaisant(e)"` lorsque `aria-valuenow="5"`. C'est la première valeur, plus compréhensible, qui sera restituée aux utilisateurs d'aides techniques, si elle est disponible. La valeur de cet attribut est mise à jour dès lors que le curseur change de position.
- Par rapport à l'axe, le curseur ne peut jamais être positionné avant la valeur minimale, ni après la valeur maximale.
- Si un pas est défini, la position du curseur doit toujours respecter ce pas. Par exemple, si un pas de 1 est défini, et que la position de départ est à 10, il ne doit pas être possible d'obtenir une valeur non entière à l'aide du curseur.
- Le curseur peut être déplacé le long de l'axe, par simple glisser-déposer à la souris, ou par simple glisser-relâcher au doigt.
- Le curseur peut également être déplacé au clavier. Lorsque le focus est positionné sur ce dernier :
 - La pression des touches `Flèche droit` ou `Flèche haut` augmente la valeur du slider, d'une valeur de pas.
 - La pression des touches `Flèche gauche` ou `Flèche bas` diminue la valeur du slider, d'une valeur de pas.
 - La pression de la touche `Origine` déplace le curseur au niveau de la valeur minimale du slider.
 - La pression de la touche `Fin` déplace le curseur au niveau de la valeur maximale du slider.
 - La pression de la touche `Page précédente` augmente la valeur du slider, d'une valeur arbitraire, mais supérieure à la valeur de pas.
 - La pression de la touche `Page suivante` diminue la valeur du slider, d'une valeur arbitraire, supérieure à la valeur de pas, et identique à celle obtenue lors de la pression de la touche `Page précédente`.

Le code ainsi obtenu avec JavaScript sera le suivant :

```
<form action="index.html" method="post">
  <label for="qualite-sommeil" id="label-qualite-sommeil">
    Êtes-vous satisfait(e) de la qualité de votre sommeil ?
    (nombre entier, de 1 pour "Très insatisfait(e)" à 5 pour "Très
satisfait")
  </label>
  <input style="display: none;" type="range" id="qualite-sommeil"
name="qualite-sommeil" step="1" value="3" min="1" max="5" />
  <div id="conteneur-slider">
    <span id="curseur-slider" role="slider" tabindex="0" aria-
labelledby="label-qualite-sommeil" aria-valuemin="1" aria-valuemax="5"
aria-valuenow="5" aria-valuetext="Très satisfaisant(e)"></span>
```

```
</div>

<input type="submit" value="Transmettre les informations" />
</form>
```

Démonstration

Une démonstration est disponible sur la version en ligne de cette recommandation : <http://wiki.accede-web.com/notices/interfaces-riches-javascript/>.

AcceDe Web – www.accede-web.com	Notice d'accessibilité interfaces riches et JavaScript	Page 30/30
Réalisé par Atalan – accede@atalan.fr		Février 2013