Introduction

Application accessibility is all about removing the obstacles for users who have disabilities. Some users may be unable to see, move a mouse, or may face any number of challenges when using a computer. IBM® has a long history of making its products accessible to all users; it makes good business sense to have applications be usable by the broadest possible customer base. Therefore, IBM has made a commitment to bring technology supporting the World Wide Regulations and Standards to users with disabilities.

Additional testing, called Accessibility Verification Testing (hereafter referred to as AVT), is required to ensure that products are compliant with these accessibility standards. The AVT Checklist is a list of checkpoints with which products that support accessibility comply. It's generated according to the worldwide accessibility standards such as the US section 508 and the W3C Web Content Accessibility Guidelines (hereafter referred to as WCAG). In this article, we'll look at a few practical examples to explain some basic and important items in the checklist. We'll then share what we've found to be best practice for how to plan and execute AVT for a Web 2.0 product.

Background

WAI-ARIA (hereafter referred to as ARIA), the Accessible Rich Internet Applications Suite, develops a way to make Web content more accessible to people with disabilities. It especially helps make dynamic content, which is generated using Asynchronous JavaScript + XML (Ajax), JavaScript, or some other technology, be more accessible. Specifically, WAI-ARIA provides a framework for adding attributes to identify features for user interaction, how they relate to each
other, and their current states. As you'll see in the upcoming examples, most of the attributes are WAI-ARIA-specific.

The examples and information provided about AVT are based on our experience of testing a Web 2.0 application, which supports the Firefox 3 browser and JAWS 10, because some attributes like WAI-ARIA markup can only be read using this specific combination.

Key elements of the accessibility checklist

So, what should be identified as the accessibility-related techniques? The accessibility checklist has several key principles:

- Choices for input methods: Keyboard, mouse, voice, and other assistive devices.
- Choices for output: Display, sound, print, and user interface elements to make visual indications for sound.
- Consistency and flexibility: Be consistent with the user settings of color, font, and so on.

These principles are the foundation of the accessibility checklist. In this article, we focus on accessibility for Web 2.0 applications, so we only discuss the Web checklist. Web applications should be compliant with the checklist to make the product usable by people with disabilities. Our examples will cover some basic, important points in the checklist as well as the fields where issues can be found most frequently.

1. Alternative text: All non-text content should have alternative text

Some browsers don’t support images, and visually impaired users cannot see images. Therefore, it’s important to provide alternative text for all images so that all users can receive the information. With alternative text the assistant technology, such as a screen reader, can convey the meaning of the image in an alternative way.

As the WCAG SC 1.1.1 guidelines described, all non-text content should have an alt text attribute. Redundant images can be set with an empty alt value, but meaningful images should be set with a meaningful sentence. Alternative text should be accurate and concise. Following are some examples:

1.1 Empty alt content for non-functional images

Figure 1 shows a priority setting. The red and green flag icons before the words "High Priority" and "Medium Priority," in this case, actually have the same meaning as the words, so the icons are non-functional or redundant. As the checklist 1.1 indicated, you should set the alt attribute for the icons to have an empty value. Listing 1 shows the corresponding HTML code.

Figure 1. non-functional images

<table>
<thead>
<tr>
<th>Prioritized Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="#" alt="High Priority" /></td>
</tr>
<tr>
<td><img src="#" alt="Medium Priority" /></td>
</tr>
</tbody>
</table>
Listing 1. Code for non-functional image

```html
<img class="lconnSprite lconnSprite-iconPriorityMedium16"
alt="" src="/activities/javascript/dojo-ibm/dojo/resources/blank.gif"/>
```

1.2 Meaningful alt content for functional/important images

Figure 2 shows an information card for a user. The photo on the card is an important image, so there should be an alt attribute with a meaningful description. Listing 2 shows the HTML code.

**Figure 2. Important image**

Listing 2. Example code for functional image

```html
<img height="35" width="35" alt="Profile photo"
```

Figure 3 shows another example of a functional image. The lightning bolt icon before "myactivity" stands for the activity's priority, which has no word expression beside it, so the alt attribute should be set to a meaningful sentence to tell users what the picture is for.

**Figure 3. Functional images**

Listing 3. Example code for another functional image

```html
<img class="lconnSprite lconnSprite-iconActivities16" dojoattachpoint="icon_AP"
src="/activities/javascript/dojo-ibm/dojo/resources/blank.gif"
alt="Normal Priority (Default)"/>
```

1.3 For multimedia, there should be a transcript describing the content

Figure 4 shows an example of a video transcript. It's a common practice in Web 2.0 applications to provide multimedia presentations, some of which even interact with users. Visually impaired users cannot perceive this information, so there should be an alternative way for disabled users to be aware of the content. In Figure 4, a text transcript link is provided below the content, so users can still get the demo's information by reading the transcript.

**Figure 4. A transcript for multimedia**

[Click here to read a text transcript of this tutorial]
2. Adaptability: The Web structure presented to users through the UI should have an alternative approach for blind users

The information, structure, and relationship for elements that can be presented to users through the UI should also be programmatically determined or have alternate text. The following two examples describe ways to identify the relationship between forms on the UI.

2.1 If a form has a label by its side, then you can use "for" to identify the relationship between the label and the form

This example works for most typical form structures that have a label next to each entry field. Add the for attribute for the label using a value that matches the id attribute of the connected data entry element. If no for attribute is set, users will hear "Editor" only when the focus is on the input form. However, if you set a for attribute for the label, users will hear "User name column editor." This helps clarify to users what the form is for and what should be entered into this form. Listing 4 shows the code for these labels.

Figure 5. A form having a label beside it

Listing 4. Usage of the "for" parameter in forms

```html
<div>
  <label for="user">User name:</label>
  <input id="user" class="lotusText" type="text" name="j_username"/>
</div>
<div>
  <label for="password">Password:</label>
  <input id="password" class="lotusText" type="password" name="j_password"/>
</div>
```

2.2 If a form has no labels, then set parameters for "title," "aria_title," or "aria_labelledby"

Figure 6 shows a form, called Add Related Activity. There is no label beside the data entry field, but it has a long description, so the title attribute should be set for this form.
Best practices for Web 2.0 applications

Figure 6. A form without a label beside the data entry field

Listing 5. Using the "title" attribute

```html
<select title="Add Related Activity" style="width: 530px;" size="5"
name="target" dojoattachpoint="targetSelect_AP"/>
```

Additionally, there are two other attributes that you can set to produce the same effect: `aria_title` and `aria_labelledby`. These two attributes are used by WAI-ARIA technology, which we'll discuss later in this article.

3. Color

Color is an important element in the user interface. Sometimes, you might prefer to use color to represent some functional meaning without providing a text equivalent. This is a violation of WCAG SC 1.4.1, which says that color can only have a decorative impact; no function can be marked only using color. The following two examples identify and resolve the color-related violations.

Figure 7 and 8 show an example of this kind of violation as well as a possible resolution. If color is used to represent a specific meaning, visually impaired users or users with a system in high contrast mode will be unable identify the message conveyed by the colors alone. The first example, in Figure7, shows a high contrast screen with all the entries having lost their color. The user, therefore, cannot see the differences in the entries. Figure 8 shows a solution that simply adds meaningful text, such as [Requiring Approval] or [Draft], to differentiate the types of entries.

Figure 7. Improper use of color
Figure 8. Proper use of color

<table>
<thead>
<tr>
<th>Published</th>
<th>Updated</th>
<th>Edit Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Today 3:02 PM</td>
<td>Today 3:02 PM</td>
<td>entry</td>
</tr>
<tr>
<td>Today 3:01 PM</td>
<td>[Requiring Approval]</td>
<td>for approval</td>
</tr>
<tr>
<td>Today 3:00 PM</td>
<td>[Draft]</td>
<td>draft</td>
</tr>
</tbody>
</table>

Figure 9 shows another example of a usability color issue. In normal mode, the tab will be highlighted when active, but if the system is set to high contrast mode, there is no way to identify which tab page is active. The solution to this is to add the tab name on the top of each tab page. Therefore, users will know which tab page is active when entering this page.

Figure 9. Color issue in high contrast mode

4. Keyboard

The keyboard is another area to pay attention to when working with accessibility issues.

4.1 All functions can be accessed using keyboard

WCAG SC 2.1.1 says that users should be able to navigate all content and functions using the keyboard.

Figure 10 shows an example of one keyboard issue. If the mouse is moved to the user's name, a card will pop up below it. The user can click the pop-up message to open the full business card. Even for this complicated operation there should be an alternative way to do this for keyboard users. In this case, a shortcut key is a good choice. For example, pressing Ctrl+Enter will open the business card for that person.
Figure 10. Keyboard issue about opening a pop-up message

Figure 11 shows another complicated action, the drag-and-drop function, which is a common feature in Web 2.0 applications. This function allows users to drag widgets on the UI dynamically. This function is difficult to simulate with the keyboard, so it should be implemented using an alternative method. Additional explicit functions on the UI show users where to move the widget.

Figure 11. Keyboard issue about the drag-and-drop function

4.2 No keyboard trap: Focus can enter and exit the current UI

Confusing changes in focus can happen when the user opens a new window on the top of the current one. WCAG SC 2.1.1 says there must be no keyboard trap. If a new window opens, then the focus should be moved to the first editable element on the new window, and there should be a way to exit the newly opened window through closing it or using the ESC key on the keyboard.

Figure 12 shows an example of a pop-up window on the UI. When the user expands the More Actions button and clicks Import Bookmarks, the figure shows that the focus is still on More Actions rather than moving onto the new window. This is a keyboard trap, because the user cannot focus on the window using the keyboard only. Keyboard navigation remains on the original window. In this situation, you should add some code to set the cursor focus specifically onto the first editable object on the newly opened window.

Figure 12. Keyboard issue about the focus

Figure 13 shows another example of the keyboard trap issue. After clicking the Prioritize button, a drop-down list pops up. If the user doesn't want to choose any item, but just wants to exit the list, they can't, no matter what key is pressed. This is a keyboard trap. There should be a way for them to exit, such as pressing ESC.
5. High contrast mode

High contrast (hereafter referred as HC) mode is specific for the display. In HC mode, all the background settings will not take effect, including images and colors, so there should be specific code defined in the application for the HC mode.

Figure 14 shows an example for this checkpoint. Each user’s record has a button to pronounce the name. But that image is a background image and will not be displayed on the UI in high contrast mode. In Figure 14, we show the alternative text to represent the icon. Listing 6 shows how span is used to set alternate text with the `<span>` directive on line 7.

Listing 6. Creating alternate text for HC users

```html
<a title="Play an audio file with the pronunciation of the person's name"
   id="pronunciation"
   href="/profiles/audio.do?key=4815d70b-680d-4bfb-ba05-f32ccedbbcb5
   &amp;lastMod=1248147056328">
   <img src="/profiles/nav/common/styles/images/blank.gif" class="lconnSprite
   lconnSprite-iconAudio16"/>
   <span class="lotusAltText">Pronunciation</span>
</a>
```

6. Others issues

There are other circumstances that affect usability, which are not easily covered by the previous categories. This section explains these circumstances.
6.1 Timing adjustment
WCAG SC 2.2.1 says that it is not suitable to set a time limitation on a Web application without a way to disable or adjust it. Figure 15 shows an example for this checkpoint. The error message will disappear after approximately 5 seconds, and users cannot disable the time-limit configuration. This is not acceptable for users with disabilities, especially for visually impaired users. The solution is to disable the time limit or to just pop up an error message.

Figure 15. A message with timing adjustment

6.2 Navigation feature
WCAG SC 2.4.1 states that all UI components should have some navigation features to help users target the focus to the correct position, such as using "Skip to main content" on the top of the UI with a blank picture. Of course, all user interfaces for a single product should have the navigation features. Listing 7 shows this checkpoint. There are three landmarks here for users to skip the content that they don't want to hear.

Listing 7. Providing ways for users to skip content they don't want to hear

```html
<!--The following lines are required for Accessibility and are used by screen readers. Please do not remove -->
<a href="#mainContent" id="lotusAccessLinkMainContent" accesskey="S"
   class="lotusAccess">
   <img src="/activities/nav/common/styles/images/blank.gif"
   alt="Skip to main content link. Accesskey S" />
</a>

<a href="#tabNavigation" id="lotusAccessLinkTabNavigation" accesskey="1"
   class="lotusAccess">
   <img src="/activities/nav/common/styles/images/blank.gif"
   alt="Skip to tab navigation link. Accesskey 1" />
</a>

<a href="#subNavigation" id="lotusAccessLinkSubNavigation" accesskey="2"
   class="lotusAccess">
   <img src="/activities/nav/common/styles/images/blank.gif" alt="Skip to sub navigation. Accesskey 2" />
</a>
```

Accessibility verification testing
Now that you've seen some of the various problems that can arise in developing a Web application, take a look at how to test for these issues.

Tools used in accessibility verification testing
Testing accessibility for Web content is a matter of following a process, which make the work more efficient.
Step 1: Syntax analysis: WebKing

Step 2: Keyboard navigation and visual focus

Step 3: Visual formatting and timing

Step 4: Audio and video

Step 5: High contrast, and large font

Step 6: Disable style sheets

Step 7: Screen reader

Each step has some related assistive tools. The official tools described in this article are: a Web syntax checker and a screen reader.

**Web syntax checker — WebKing**

We show the Web syntax check as the first step. It's mainly used for a Web content syntax check by developers instead of testers. It's always started at the end of development, but before the official start of AVT. WebKing is a popular Web syntax checker tool that scans the HTML code directly to help you fix the accessibility errors before using other assistive technology.

Dynamic content generation is a feature of Web 2.0 applications. Using Ajax to dynamically generate the object on the Web can make the HTML code different each time a page is loaded. WebKing supports two different methods of scanning the code: static scan and dynamic scan. A static scan can be performed by developers who scan the source code directly in the development environment. A dynamic scan can be done by testers by interacting with applications and dynamically loading different content into the application.

The resulting reports for the WebKing scan are used to analyze the reason and the resolution for the different violations. Following is a list of several violations frequently encountered.

**Duplicate ID**

All elements in the HTML code should have an ID. For assistive technology to identify the element on the HTML page, the ID should be unique, or there will be confusion about which element is the correct one. It is easy to conform in static code such as HTML, but it's difficult for dynamically generated code such as the ones rendered from JavaScript. In those cases, you must have some way to generate a different ID each time. Figure 16 shows an example for the duplicate ID solution. The button "hide details" is rendered from the JavaScript code each time the page is loaded. Each button is set with a unique ID by adding a number as the suffix.

**Figure 16. Duplicate ID example**

```html
<hide details>
```

Best practices for Web 2.0 applications
ALT text & Label — form relation setting

These problems are found so frequently that this is why WebKing should be run as the first step. In this phase, all images are checked as to whether they have the alt attribute set and whether the relationship between objects are set. If not, you will be alerted and should fix the problems before entering into the next phase.

Not all the violations are real problems. Taking duplicate IDs for example, there are some hidden objects, which are not shown to users, but which are used for some functional purposes such as analytic scripts. These types of issues can be suppressed in WebKing or can be considered an exception in the report.

Screen Reader – JAWS

A screen reader is mainly used by visually impaired users. It helps them navigate the whole page using their ears instead of their eyes, and it can also help to identify all elements and their relationships on the user interface. There are some shortcut keys for users to open the "Link list", the "headline list," and so on, when using JAWS to listen to content on the UI. Also, there is a place for JAWS to set the read configuration, such as whether to read the context for the link, or the title, or the alt, and so on. To set the configuration, open the main panel of JAWS, choose Utilities > Configuration Manager > Set Options > HTML Options. The configuration panel is shown in Figure 17.
JAWS 10 can work with Firefox Version 3 or Microsoft® Internet Explorer 8 to read more content on the UI, as these support WAI-ARIA technology. Users can hear more information on the UI, such as the structure of the page or the relationship of the objects.

JAWS has two modes for users to listen to the content on the UI. One is Virtual PC cursor mode, and the other is Forms mode. Sometimes these are simply referred to as Virtual PC cursor mode on or off. When running JAWS to listen to content, the two modes have different results. The Virtual PC cursor mode lets users read through the document line by line or by using several shortcuts. The Forms mode lets users interact with controls and type information into editable controls. The two modes can be switched using the Insert+Z key combination. Users cannot enter any information in the editable field in a Virtual PC cursor mode, so they must change to another mode to make this work.

Testing methodology

AVT for Web 2.0 applications is not a common phase in testing, but it is important for applications to become compliant with the W3C Web checklist. Following are some best practices for the AVT methodology.

- AVT should start after all the major UI changes are complete, because major changes to the UI will bring new elements or update existing element attributes.
- AVT should be completed before the UI freeze. Issues found in this phase will result in a UI change, which will result in an element attribute change, and the issues should be fixed without any exception. If the project has a deadline for the UI change to be checked in, then the AVT should be completed before this deadline.
• Webking is used more like a unit testing tool than a tool used in the AVT phase, especially for the static scan. All other testing should be started after you have finished the WebKing static scan and fixed all the issues found during the scan.
• JAWS is a sensitive tool. Different versions of JAWS will read different content for the UI elements. This is not only true with major version changes, such as JAWS 9 and JAWS 10, but also smaller fixpack changes, such as JAWS 10.0.512 and JAWS 10.0.1142. The earlier version may have some bugs when reading the ARIA attributes, while the later version does not. So, identifying the version of JAWS is also an important step to follow.
• ARIA is a good technology to use in Web 2.0 applications. There are lots of attributes defined in this technology that can help users to identify the structure and meaning of the elements on the UI. This technology can also save you a lot of effort. For example, in the aria-sort attribute, you should set the title of the sort link using a long sentence to tell users whether it's sorted descending or ascending. Using the aria-sort attribute, users can set the title with a normal word, and add aria-sort = ascending/descending to identify the mode of the sorting.
• Sometimes it's hard for a tester to identify whether the alt text is needed or whether it can be left empty. The standard is that if there is alternative way for users to know the information sent out by the picture, then the alt can be empty. Otherwise, it should be filled with meaningful text.

Conclusion

AVT is an important testing phase in a project life cycle. Each product has its own different implementation technology and should be compliant with the WCAG Checklist. This article described some basic, but important, checkpoints and testing tools. Start learning these skills, and apply them to your own test processes.

© Copyright IBM Corporation 2009
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)