How to do Web Accessibility

Group 2

Christoph Lipautz, Christian Mayer, Stephanie Reich, and Mario Windpessl

Graz University of Technology
A-8010 Graz, Austria

06 Dec 2016

Abstract

The document on hand discusses common problems with ensuring content to be accessible and their practical solutions, handling of dynamic accessible content within rich internet applications, as well as methodologies to test accessibility.

Keeping content accessible for everyone is an ongoing challenge to modern web authors. With the increasing importance of JavaScript and its power to keep web content dynamic, in the same way the need for proper guidelines and web standards are given. The current state of the World Wide Web using modern browsers offers feature rich web applications that rely on web standards only. The gap between web and mobile applications is getting smaller and smaller, and the internet of things can be found more and more in everyone's home.

Throughout the survey common problems are discussed, containing best practices and explanations for the improvements given. A closer look is given to the web standard WAI-ARIA, that is explained in detail, including sample implementations and common usage description. Further different testing approaches are examined, their advantages and disadvantages shown.

Contents

Contents ii

List of Figures iii

List of Tables v

List of Listings vii

1 Motivation 1

2 Design Patterns 3

2.1 General Aspects . . . 3
2.2 Visual Aspects . . . 4
2.3 Navigation . . . 5
2.4 Non-text-content . . . 6

3 WAI-ARIA 11

3.1 Roles, Properties and States . . . 11
3.2 Attributes . . . 12
3.3 Patterns . . . 13
3.3.1 Alert and Alert Dialog . . . 13
3.3.2 Toolbar . . . 14
3.3.3 Tooltip . . . 16
3.3.4 Datepicker . . . 16
3.4 Landmarks . . . 17
3.5 Bad Practices . . . 17
3.6 Current State and Future . . . 18

4 Accessibility Testing 19

4.1 Manual Testing . . . 19
4.1.1 Keyboard Navigation . . . 19
4.1.2 Screen Readers . . . 20
4.2 Automated Testing . . . 22
4.2.1 Accessibility Developer Tools . . . 23

A About The Authors **27**
Christoph Lipautz 27
Christian Mayer 27
Stephanie Reich 27
Mario Windpessl 27

Bibliography **29**

List of Figures

1.1 Situational Disabilities 2

2.1 Bad Setting for Colour Contrast 4
2.2 Sample Functional Image 6
2.3 Sample Decorative Image 7
2.4 Sample Complex Image 8
2.5 Table Representing Complex Image 8
2.6 Documents for Audio and Video Files 9

4.1 Screen Reader Usage Trends 20
4.2 Claws - Screen Reader Emulator 22

List of Tables

4.1 Mobile Screen Reader Usage 22

List of Listings

2.1 Feedback in Header or Page Title 6
2.2 Bad Practice Decorative Image 7
2.3 Good Practice Decorative Image 8
2.4 Good Practice Complex Image 8
3.1 Alert Bootstrap v2 13
3.2 Alert Bootstrap v3 14
3.3 WAI-ARIA Alert Dialog 14
3.4 WAI-ARIA Toolbar 15
3.5 WAI-ARIA Tooltip 16
3.6 WAI-ARIA and native HTML button 18
3.7 WAI-ARIA and native HTML label 18
4.1 Running ADT as a Standalone Tool. 23
4.2 Testing Multiple Web Pages with A11y. 23
4.3 Using A11y as a Node.js Module. 24
4.4 Output of capybara-accessible 25

Chapter 1

Motivation

In general, accessibility means making something accessible for everyone. Web accessibility, in particular, means making the web accessible for everyone. In order to obtain that certain barriers must be considered and reduced. Disabilities could be permanent or situational, as visualised in Figure 1.1. [WAI, 2016c]

In the following, barriers are listed in order to get an idea of the problems which have to be dealt with by developers of accessible websites:

- Physical disabilities, such as MS, ALS, paraplegic
- Seizures, such as photo epileptic seizures
- Visual disabilities, reaching from difficulties with colour contrast up to blindness
- Hearing disabilities, reaching from difficulties with understanding up to deafness
- Cognitive disabilities, such as dyslexia and low literacy
- Temporary/situational disabilities, such as a broken arm

Some of the people who suffer from one of those disabilities might use a "direct access", which is unassisted, others also might use an "indirect access", meaning an assistive technology, such as screen readers. In chapter 4.1.2 different screen readers will be focused.

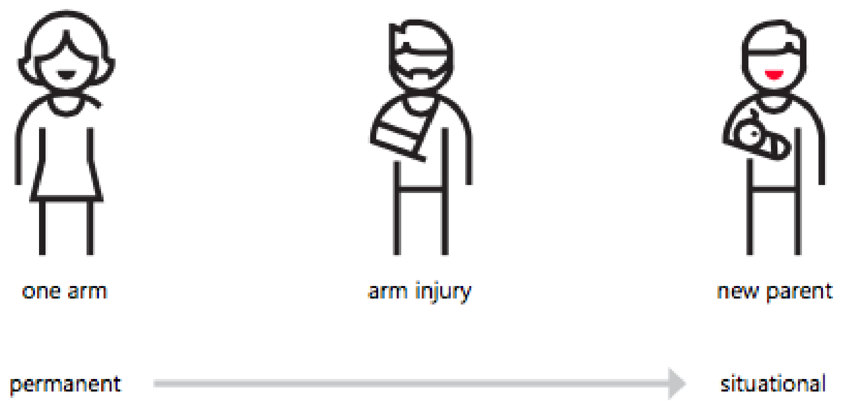


Figure 1.1: Disabilities can reach from permanent to situational [[Storey, 2016]]

Chapter 2

Design Patterns

When designing a new website which should be accessible for everyone and fulfil the WAI guidelines for web accessibility WAI [2016c], various aspects need to be considered.

These points are structured into four main chapters: general aspects, which every website must contain, visual design of a website, navigation on a website and non-text-content, which includes images and audio and video content.

2.1 General Aspects

The first point is the language of a website. Any assistive technology, especially a screen reader, needs to know in which language it should read out the given content. A developer declares the language of a page by using the HTML lang tag (or lang attribute). There is one particular language code for each language, for example "en" for english, "de" for german or "fr" for french.[W3Schools, 2016a]

Setting the language code by the use of HTML on any element can look like the following:

```
<element lang="language_code">
```

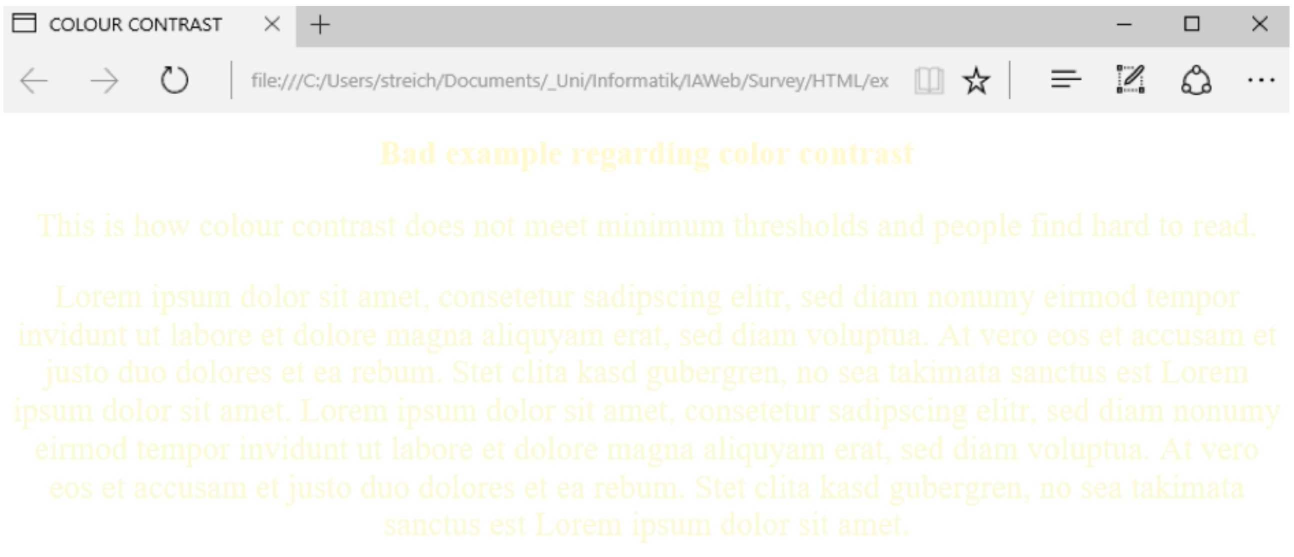
Regarding to the ability of enlarging contents such as texts or images on a webpage, pinch-to-zoom should be allowed. For that, the viewpoint meta tag in HTML has to be applied.

Any page can use the following setting for the viewpoint meta tag to allow pinch-to-zoom:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

The text content of a website should always be kept simple and easy to understand for everybody. Persons with a low literacy or dyslexia cannot interpret too long paragraphs, sentences or words, as well as passive sentences.

Heading elements are mutually dependent. It is an essential precondition to be clear in defining of these heading elements. Hence, one should only use one main element and also only one h1 tag. The main element includes the provided content, which is unique on the website W3Schools [2016b]. The h1 tag specifies the one and only top heading. [Pickering, 2016]


COLOUR CONTRAST
file:///C:/Users/streich/Documents/_Uni/Informatik/IAWeb/Survey/HTML/ex

Bad example regarding color contrast

This is how colour contrast does not meet minimum thresholds and people find hard to read.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Figure 2.1: A bad example for colour contrast.

2.2 Visual Aspects

It is also of great importance to take care of the font size. “The key to inclusive design isn’t to target specific groups, it’s to not exclude groups arbitrarily. . . ”Pickering [2016]. This means that in accessible web design one should constantly think about possible users. Viewing the font size, older generations must be considered in particular. In HTML, a font-size of 100 % declares a size of 16px by default – if the user has not changed his or her system or browser settings on their own. Consequently, it is a better practice to use a percentage size than one in pixel ratio.

In CSS coding, font size declaration should look like this (for example the HTML p tag):

```
p { font-size: 100%; }
```

Another point in order to make the text content readable, is to choose the “right” font. Regarding to this, it is important to consider the following points: There should not be any redundant ornamentation, the proportions of a letter should be consistent, letters should be distinctive and all of the characters on the website should be supported.[Pickering, 2016]

Font icons should be used carefully because this usage might exclude people with dyslexia. The reason for this is that those people might see broken characters instead of the font icons and furthermore, they would not be able to understand the proper meaning.

Developers should permanently ensure that the colour contrast on the website meets minimum thresholds. This point could not only be important for disabled persons, but also for all people in order to make the website pleasant to read and to browse through. The colour contrast could be ensured by the developer by working on the CSS file. Regarding this, it is also essential to use more indications of meaning than only the colour contrast so that people with a bad colour perception would not have any difficulties in understanding. A bad example for colour contrast can be found in Figure 2.1.

How to do better:
When choosing colours for a website using CSS, one should use a table of CSS colours, for example the one provided by w3schools.com (http://www.w3schools.com/cssref/css_colors.asp) and also a common sense.

To position the text content on a website, one should never use the text-align "justify", which is the layout that newspapers are using to guarantee that all lines have the same width. As a result, large spaces between the single words occur and users feel disturbed in their reading flow. Another problem is that hyphenation (by hyphens property in CSS) is poorly implemented.

As an identification of links, not only a different colour should be used, but also an underlining. This helps people who have difficulties in distinguishing colours.

As a basic design principle, developers should ensure building a rather simple than a complex layout. Simple user interfaces are more usable and can easily be adapted to an accessible interface (if accessibility is not secured anyway).

2.3 Navigation

Navigation landmarks are needed to help the user in navigating on a website. Examples for such landmarks are a main menu or sidebars.

Furthermore, making links descriptive should be taken into account. Any handicapped person must understand where links provided on a website are leading to. This is done by adding a description into the HTML link tag and into the a tag. For the website itself this means some information needs to be added in the title tag in order to make the page descriptive. Thus, also author or site information could be appended.

For people who are only using keyboard navigation, it would be supportive to integrate skip links. This means that they can easily jump to the required section on the website. When the site is reloaded, a screen reader always starts at the beginning of the page, such as the content of the title tag or metadata. Consequently, the navigation needs to be in the first section of the encoding of a website.

Links that are opened in a new window, such as links to other websites or non HTML content, are a problem for screen readers. Older screen reader versions do not alert the user and so this can be confusing. To manage that, one possibility is to add the information at the end of the link description (for example "open new window" or "PDF") or use images (with alt attribute).
Declaring an external link via HTML:

```
<a href="form_scholarship.pdf">Form scholarship (PDF)</a>
```

Buttons and links should be large enough to be reached by adult finger pads. For the implementation, the HTML button tag should be used.

Voice control gives users the possibility to complete text fields or forms, and navigate on the page or control buttons. Websites should be able to handle voice control. The basis to obtain good voice control is to design the website for keyboard navigation. [W3Schools, 2016a]

Users should always receive a feedback from the system. They need to know about the success or the failure of their actions. All notifications and messages need to be clear and easy to understand for all users. Keyboard navigation or screen readers should be supported. There are multiple ways to guarantee this for notifications or messages, as described in the following. After a form is submitted, the feedback can be written into the heading or the page title (Listing 2.1).

```
<h1>Payment accepted</h1>
<h1>Payment not accepted - Wrong Credit Card information </h1>

<title>Payment accepted</title>
<title>Payment not accepted - Wrong Credit Card information </title>
```

Listing 2.1: Writing feedback into heading or page title.

Figure 2.2: A sample for an image used as play button [Icon made by Madebyoliver from www.flaticon.com]

Other possibilities for notification are to style the input fields (for example green for success, red for failure), to add dialogues or to give a scaled feedback (for example three different colours used for password strength).

2.4 Non-text-content

Next, it is very important to provide alternative text for a non-text-content. The reason is almost the same as before – assistive technologies need further information. For that alternative text, the HTML alt tag is used.

There are several reasons why the HTML alt tag should always be used for images. When an image cannot be shown to the user or when the user is not able to see the image, the text in the alt tag (alternative text) gives a description. Users with visual impairments like blindness or colour blindness could use screen readers to read out the alt tag. [WAI, 2016b]
The first step to make images accessible is to think about the main functionality of the image. There are some guidelines from the W3C which describe how to deal with different types of images.
The HTML alt tag should always contain a description of the image. However, it must also be thought of cases where the main function of the image is not the image itself, as described in the following.

If an image has a particular functionality, as for example an image that is used as a play button, developers should describe this functionality in the alt attribute, shown in the following example code:
`<img src="play_button.png" alt="Black Triangle">`
A bad practice would be to describe the form of the image like in the following HTML code:
`<img src="play_button.png" alt="Play">`

There are some images which are only used for decorative issues. Consequently, any disabled person would not be interested in them. Developers should tell the screen reader to skip the image by adding an empty alt tag.(Listing 2.3)
For complex images, such as pie or bar charts, the best practice is to use additional tables or provide a text description, shown in Listing 2.4 and Figure 2.5. Screen readers are not good in reading out the content of a pie chart. On the contrary, most screen readers do not have any problems in reading out the content of a table.
A bad practice for complex images is shown in the following HTML code:

Figure 2.3: A sample for a blue bar used as decorative image.

```
<img src="blue_bar.png" alt="Blue Bar"> or
<img scr="blue_bar.png">
```

Listing 2.2: A bad practice for decorative images.

```
<img src="sales_2015.png" alt="Sales 2015">
```

Another important aspect are label form elements. People using an assistive technology should always get information about what must be entered in elements on a website. Therefor the HTML label tag is used.

One essential point is captioning audio and video content. Again, this is really useful for hearing disabled persons, but not only for them. Actually also people without disabilities like captioned videos and audios, because they make it possible to be watched everywhere, even in a university course.
For audio files a text transcript should be added. This is necessary in order to assist deaf people. The text transcript has to be accessible.
Besides the text transcript or captioning, a video description is needed for video files. This should assist people with visual disabilities. [Griffin, 2016]

```
<img src="blue_bar.png" alt=""> or
 <img scr="blue_bar.png" alt " " >
```

Listing 2.3: A good practice for decorative images.

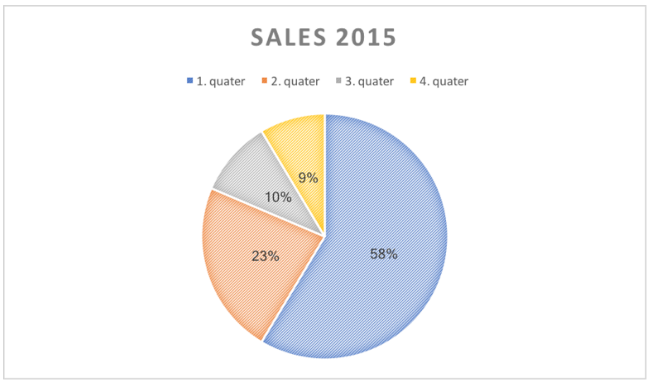
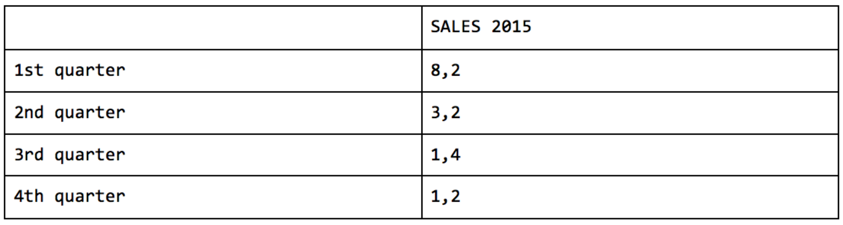


Figure 2.4: A sample for a pie chart used as complex image.

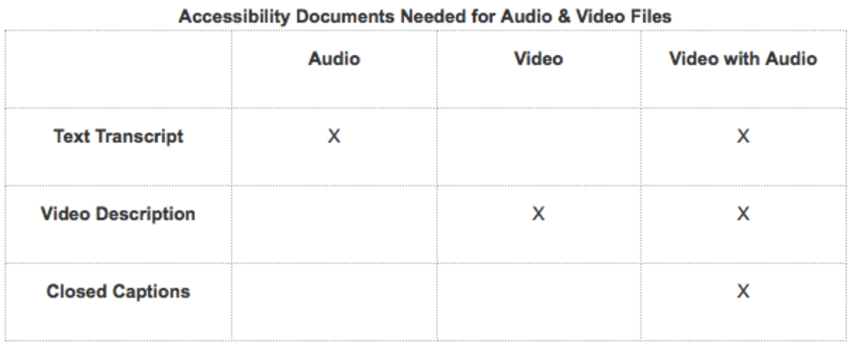
```
<img src="sales 2015.png" alt="Chart showing the Sales 2015">
 <a href="2015-sales.html">2015 sales text description of the bar chart
    </a>
```

Listing 2.4: A good practice for complex images.



	SALES 2015
1st quarter	8,2
2nd quarter	3,2
3rd quarter	1,4
4th quarter	1,2

Figure 2.5: A sample for a table representing a complex image.



Accessibility Documents Needed for Audio & Video Files

	Audio	Video	Video with Audio
Text Transcript	X		X
Video Description		X	X
Closed Captions			X

Figure 2.6: Accessibility Documents needed for audio and video files [[Griffin, 2016]]

Chapter 3

WAI-ARIA

ARIA is an acronym for Accessible Rich Internet Applications and one of the accessibility guidelines created and maintained by the Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C). It is used to extend HTML by descriptive attributes in order to achieve a higher level of content description than native HTML elements can provide. It is not a content technology on it is own but providing attributes to extend HTML [WAI, 2016e]. With this attributes and certain guidelines WAI-ARIA provides proper ways to deal with dynamic content, or complex content structures, and it defines keyboard control. HTML and WAI-ARIA have overlapping features. As a rule of thumb: If a native HTML element exist it should be used and no extended description is needed. WAI-ARIA is a W3C recommendation and therefore a web standard. Additional to HTML, WAI-ARIA is also used in SVG and EPUB.

3.1 Roles, Properties and States

The WAI-ARIA specification can be used as a framework to add meta information to web content that is understandable by assistive technologies. Besides guidelines to define how certain types of content and form controls should be described and behave, there are three types of attributes: roles, properties and states. They can be defined on HTML tags in order to assist in one of the following tasks.

- Define an object or type to tell its purpose.
- Describe a structural role to ease navigation.
- Describe a state in order to explain changes in dynamic content.
- Define live regions that may receive changes in a high interval.
- Support drag and drop for certain assistive technologies and input methods.

Role is currently the only WAI-ARIA attribute that is not prefixed by `aria-`. It defines a landmark or specifies a type of object. A property adds meta-information to an element to either add or point to further information. Dynamic changing data requires a state to inform a reader about its current content. The WAI-ARIA Definition of Roles WAI [2016a] provides many example widgets, including guidelines on their requirements for a proper implementation. If there is no widget to fit an element, guidelines are available to define behaviour and how to deal with additional descriptions if needed [W3C, 2016a].

Additionally to the attributes maintained by WAI-ARIA, the guidelines illustrate best practices in using HTML attributes to improve accessibility. Changing structure requires careful treatment on focus handling and keyboard input interpretation. The HTML attribute `tabindex` regulates user agent navigation using the TAB key per convention. It has to be considered that any newly introduced document fragment may require active focus set and navigation handling defined.

3.2 Attributes

Any object is able to require attributes. The `aria-` prefixed attributes are either states or properties and add additional meta information. Both can change their value over time, although property values are more likely not to differ to their initial value. The following list gives a consise overview, using a few examples, to demonstrate the proper use of WAI-ARIA attributes. A complete list of all available attributes can be found in the states and properties specification WAI [2016d].

aria-label names an object that is not desired to be displayed visually but is required to understand the current context. On some HTML elements the attribute `title` may be available as similar concept. Many web browsers have a built-in behaviour to present the title content on mouse hover, that maybe not desired by the author and therefore cannot be used. Another comparable solution is the `alt` attribute that is available for the HTML `img` tag, but is not available for other tags, in order to provide some alternative text [Pickering, 2016].

aria-labelledby defines a reference to the name of an object. The value is any identifier using the HTML `id` attribute that contains the name of the object. In contrast to the `aria-label` this property should be used if the name is visually presented.

aria-describedby provides verbose information of an object. Similar to `aria-labelledby` this attribute points to another document node holding a description in a higher detail.

aria-selected represents a selected state of an input control. A comparable native HTML element is the combination of `select` and `option` tags. With `aria-select` although it is possible to select elements that could not be marked before. Considering choosing specific images out of a list of images can be made accessible by a screen reader as well.

aria-checked represents a checked state of an input control. A comparable native HTML element is the `input` tag with type set to checkbox. Further the element extends the possible values of a checkbox given by true or false with the value mixed to create tri-state elements and undefined to transport an element can not be checked.

aria-hidden informs the user agent about the visual state of a document fragment. The visibility of any HTML element can be controlled using CSS attributes `display` or `visibility`. To ensure the `aria-hidden` state is correctly changed the CSS styles can use the single rule `[aria-hidden="true"] { visibility: hidden; }` or `[aria-hidden="true"] { display: none; }` so that visibility control relies on the WAI-ARIA attribute only. Additionally, content authors could use techniques like off-screen-positioning to hide elements from an user. This requires the aria-hidden attribute to be adapted accordingly as well. Not relying on CSS there is also a HTML attribute hidden available. Content authors must use this state with caution not to hide or present different content to users relying on assistive technologies.

aria-live indicates an element will receive updates. Depending on the value set there can be three different characteristics described. The value `off` tells the user agent any update does not require the users attention. If the current object is focused though content update is presented. The value `polite` should trigger information about an update on the next suitable opportunity. The last option with value `assertive` marks the requirement of a users attention and should be presented as soon as the content changes.

aria-invalid marks an input to be as being invalid in terms of the applications requirement. This attribute should be only used when a user has submitted data.

aria-busy indicates an element is currently being updated and content may not be complete yet.

aria-valuemin, aria-valuemax, aria-valuenow describe the input values possible for a range widget. Whereas `aria-valuemin` and `aria-valuemax` are statically defined, the value of `aria-valuenow` has to be updated by the application.

3.3 Patterns

The WAI defines more then 30 widgets and common user-interface elements that are used in web applications. With current draft version 1.1 defined ? Authoring Practices W3C [2016a] those are one of the following: accordion, alert, alert dialog, auto complete, button, checkbox, combo box, date picker, dialog, data grid and simple data table, landmark navigation, link, listbox, media player, menu, menu button, popup help, radio group, rich text editor, general, tree or tabbed site navigator, slider, spinbutton, tab panel, toolbar, tooltip, tree grid, tree view, window splitter or wizard. Any pattern is described by its presentation and common behaviour and behaviour in interaction. In order to improve accessibility the requirements, to be correctly interpreted by an assistive technology, can be reviewed, as well as specification for keyboard interaction.

If no pattern matches, there is a common description on how to create an accessible interface use the right meta information to make user agents understand the information present.

Some widgets are discussed in the following sections Alert and Alert Dialog, Toolbar, Tooltip, and Datepicker, in order to give an overview how the guides describe a recommended implementation.

3.3.1 Alert and Alert Dialog

A dynamic application can use JavaScript to accept and submit data. This will be handled client side in the user agent and being processed on server side. To signal the processing has been successful some feedback is required. On a failed processing corresponding message and reason should be presented to the user. An alert message does not require user interaction. Listing 3.1 shows the alert component of the popular HTML, CSS and JavaScript framework Bootstrap. The alert message is grouped together in division tag `div` containing a button to close the message and the content text itself. The close button has a `data-dismiss` attribute set that will be controlled via JavaScript and trigger the deletion of the DOM element. The message will be visually shown to the user either by inserting the alert as new element in the DOM or changing the text of an already existing element that may serve as a template. In both cases the process will be executed using JavaScript. Any assistive technology will have to decide how to handle the new content. A screen reader for example will have to either ignore the message, read the message at the next opportunity or instantly stop current action and switch to reading the new content.

```
<div class="alert">
  <button type="button" class="close" data-dismiss="alert">&times;</
     button>
  <strong>Warning!</strong> Best check yo self, you're not looking too
     good.
</div>
```

Listing 3.1: Alert message from Bootstrap [*Bootstrap v2.3.2* 2013].

A better approach is already used in the current versions of bootstrap, as shown in Listing 3.2. The role attribute defines the alert to be an object of type alert. With that information the user agent now knows there is an alert message and can decide on its own how to deal with it. Further the button receives a descriptive label, as well as the `&times;` character representing an X symbol for visual user agents, must not be interpreted by assistive technology.

```
<div class="alert alert-warning alert-dismissible" role="alert">
  <button type="button" class="close" data-dismiss="alert" aria-label="
     Close">
    <span aria-hidden="true">&times;</span>
  </button>
  <strong>Warning!</strong> Better check yourself, you're not looking
     too good.
</div>
```

Listing 3.2: Alert message from Bootstrap [*Bootstrap v3.3.7* 2016].

Sometimes an alert message may also require the attention of the user. A common case in web applications is a confirmation dialog asking the user if some action should be really executed. WAI-ARIA introduces the role "alertdialog" for this purpose. Listing 3.3 demonstrates a sample implementation. The element is defined with the role "alertdialog" and the relationship for label and description are set. The message is inside a paragraph that is not focusable by default, therefore it receives the role of a "document", and will be focusable due to the attribute `tabindex`. Additionally with the appearing of the message, the focus should be set on the element.

```
<div class="alert" aria-labelledby="alert_title" aria-describedby="
    alert_message">
  <p id="alert_title">Clear cart!<p>
  <p id="alert_message" role="document" tabindex="0">By pressing the
     button
  clear all products will be removed from your shopping cart.</p>
  <button id="alert_accept">Clear</button>
  <button id="alert_cancel">Cancel</button>
</div>
```

Listing 3.3: A sample implementation of an alert dialog that requires user confirmation.

3.3.2 Toolbar

Web applications may have controls. Controls of a certain kind are often grouped together. This group can receive the role of a "toolbar". The main purpose of grouping those controls is to improve the navigation for keyboard users. The focus on a website in a browser is switched with the tabulator key. As soon as the toolbar has focus, the left and right arrow key can be used to change the focus inside the toolbar. To navigate away, the tabulator key can be used again. If a toolbar is in vertical orientation, the WAI-ARIA attribute `aria-orientation` can be set to vertical. In that case the keys change to the up and down arrow keys. Optionally an implementation can allow the switch from the last active item to the first. An `aria-label` attribute for the toolbar is recommended. If the controls have a specific target that gets controlled within the document, this should be referenced with `aria-controls`.

Listing 3.4 shows a minimal implementation of two toolbar elements. Inside two `div` elements controls are represented via buttons. The groups both receive an attribute `tabindex` with value 0 in order to make the group being able to receive the focus by using the TAB key on a keyboard. To avoid moving the focus then to the buttons, the buttons have a `tabindex` of a value -1 set. The further focus handling is now controlled using JavaScript. A key-event handler is registered to the toolbar elements. On keyboard input via the arrow keys left or right the focus inside the element is calculated and changed.

```
<style>
  .toolbar:focus { outline: 1px dashed; }
</style>
<div id="audio-controls" tabindex="0" class="toolbar"
  aria-label="Audio Controls" aria-controls="the-audio">
  <button tabindex="-1">Mute Audio</button>
  <button tabindex="-1">Increase Volumne</button>
  <button tabindex="-1">Decrease Volumne</button>
</div>
<div id="video-controls" tabindex="0" class="toolbar"
  aria-label="Video Controls" aria-controls="the-video">
  <button tabindex="-1">Mute Video</button>
  <button tabindex="-1">Increase Video Size</button>
  <button tabindex="-1">Decrease Video Size</button>
</div>
<script>
  var toolbarHandler = function(event) {
    var childElements = this.querySelectorAll("button");
    (({
      ArrowRight: function(buttons) {
        var indexToFocus = 0;
        for(var index = 0; index < buttons.length; index++) {
          if(buttons[index] !== document.activeElement) {
            continue;
          }
          indexToFocus = index + 1;
        }
        if(indexToFocus >= buttons.length) {
          indexToFocus = 0;
        } else if(indexToFocus < 0) {
          indexToFocus = buttons.length - 1;
        }
        buttons[indexToFocus].focus();
      },
      ArrowLeft: function(buttons) {
        var indexToFocus = 0;
        for(var index = 0; index < buttons.length; index++) {
          if(buttons[index] !== document.activeElement) {
            continue;
          }
          indexToFocus = index - 1;
        }
        if(indexToFocus >= buttons.length) {
          indexToFocus = 0;
        } else if(indexToFocus < 0) {
          indexToFocus = buttons.length - 1;
        }
        buttons[indexToFocus].focus();
      }
    })[event.key] || (function() {}))(childElements);
  };
  var toolbars = document.querySelectorAll(".toolbar");
  for (var index = 0; index < toolbars.length; ++index) {
    toolbars[index].addEventListener("keydown", toolbarHandler);
  }
</script>
```

Listing 3.4: A toolbar element to group controls. For quicker navigation the group will be focused, navigation inside the toolbar can be done using the arrow keys.

```
<style>
  .tooltip { visibility: hidden; }
  input:hover + .tooltip { visibility: visible; }
  input:focus + .tooltip { visibility: visible; }
</style>

<input type="email" name="email" id="email"
  aria-describedby="tooltip-email"
  aria-required="true" />
<span id="tooltip-email" class="tooltip"
  role="tooltip" aria-hidden="true">
  Requires e-mail domain tugraz.at
</span>

<script>
  var input = document.getElementById("email");
  input.addEventListener('onfocus', function(event) {
    document.getElementById("tooltip-"+event.target.id)
      .setAttribute("aria-hidden", false);
  };
  input.addEventListener('onblur', function(event) {
    document.getElementById("tooltip-"+event.target.id)
      .setAttribute("aria-hidden", true);
  }
</script>
```

Listing 3.5: A tooltip describing the requirement of using a certain domain in order to provide an email address for an input field.

3.3.3 Tooltip

A tooltip is an additional description to content, providing detailed or extended information. In Listing 3.5 an HTML `input` of type `email` has a certain application based restriction: The e-mail address provided requires the domain tugraz.at to be present. This significant information is required to be presented to any user that is going to fill out this input. The description of this requirement is given in a HTML `span` element provided after the input. For visual representation the `span` is hidden via CSS but appears on hovering the mouse pointer over the `input` field. Additionally instead of triggering on hover, also focus will make the description appear that already is enough to support keyboard only control. Considering a screen reader user the object requires extra handling, otherwise the message could be missed by this user. By pointing to the tooltip as additional description and changing its `aria-hidden` state on focus or focus-leave will ensure the assistive technologies recognises the notice.

3.3.4 Datepicker

A datepicker widget offers a calendar to select a certain date having a common interface instead of a textual input. Providing only textual input has several faults, like the missing link to what weekday is presented by a selection.

By accessing the element, the focus has to be set on the current day or selected day. Days and optional weekdays container element is presented with the role set to `grid`. The weekdays themselves use a role `columnheader` and the days `gridcell`. On a change of selection the WAI-ARIA attribute `aria-selected` has to be set to true on the current selection, any other days require the attribute to be removed or set to false. Visually it has to be ensured that any focused element is always marked in order to know the current active control. The keyboard navigation uses TAB to focus or leave the focus from the widget, arrow keys to navigate the days grid and space to mark an active selection. Additional commands can be used to quick control the input. Pressing the Home and End key can move to the beginning and end of a month. Page up and down keys change the month. Enter may save the selection to the input and close the currently shown calendar presentation.

Although there is a type `date` of the native HTML tag `input` it is not commonly used as the presentation is very different depending on the user agent, and can not be easily changed.

3.4 Landmarks

A screen reader or similar assistive technology will walk through content as it occurs in the structure of the HTML document. Even if content is well structured, the navigation comes first, and main content is close to the top, a user might want to navigate through one page sections directly [Pickering, 2014]. Although HTML does provide structural tags like `section`, `aside`, `nav`, `header`, `footer`, or `main`, the landmarks can be helpful to improve moving between main points of an application. Landmarks can even be nested, will be hierarchically detected, and provide a useful navigation.

To set a landmark on a HTML element the WAI-ARIA `role` attribute is used. There are seven different types available. Pickering [2014] describes them as following.

banner introduces to the page content and should only occur once per page. Commonly the banner is located at the top of a website, containing the websites logo and often also the main header `h1`.

contentinfo describes that page and what is the page or website about. Websites commonly keep this information inside the footer, including copyright and contact information. It should be used only once per page.

main should be one and only one element on a page containing the main content. The HTML tag `main` can be used. Similar rule exists for the main header, defined by the HTML tag `h1`: Only one main header per content page.

navigation is a landmark containing links to other pages or section of the website or page. The landmark can be used multiple times as well as be nested inside other landmarks. It should be used for navigational purpose only. Any list of links is not useful if treated as navigation.

complementary is recommended to be used only once per page the landmark marks side content that may extend the content of the page but is not in center.

search is a role that marks an element to search the page, website or filter certain content.

form can be used on any important form input sections.

3.5 Bad Practices

Although WAI-ARIA supplies attributes to add meta information into objects, a content author should use it with caution, fallback to native HTML wherever possible and use attributes only where needed. If information provided exceeds the requirements screen reader users have to understand the application, they will be kept busy listening to verbose description of already learned concepts. By relaying on native HTML elements there are several benefits to consider. In listing 3.6 a `div` tag is used to create a control element similar to a button. Any click event to be controlled require JavaScript for this new object. If using the native `button` tag instead

```html
<!-- Bad -->
<div class="btn" role="button" tabindex="0">click me!</div>
<!-- Good -->
<button>click me!</button>
```

Listing 3.6: Using WAI-ARIA attribute to define a button is a bad practice as it will not implement the native features that come along with a button like clickability or focusability.

```html
<!-- Bad -->
<span id="zip">Postcode:</span>
<input aria-labelled-by="zip" />
<!-- Good -->
<label for="zip">Postcode:</label>
<input id="zip" />
```

Listing 3.7: HTML defines a label tag that is the correct way of labelling an input. If using WAI-ARIA attributes to workaround labels some native features get missed that may be offered by the user agent like input focus on label click.

the element will have several extras, like it can have focus without setting the tabindex attribute, or receiving click events and keyboard input properly.

Although WAI-ARIA supports attributes that describe relationships between objects there are also native HTML tags doing so. In Listing 3.7 a form input is described by a generic `span` instead of the `label` tag that was introduced with HTML5 for this purpose. Although the results on understanding the context may be the same, still the native version has some benefits over the solution using a relation. Browsers do link the label tag automatically, so that on click on the label the input get focused. This can come in handy for people missing the connection or using small touchscreens.

3.6 Current State and Future

The first specification of WAI-ARIA was published in 2014, initialising the version 1.0. The current version 1.1 has to be noted as draft and is receiving ongoing changes. Changes from version 1.0 to 1.1 should be minor, any bigger changes will be considered in next version WAI-ARIA 2.0 [WAI, 2016e].

Using WAI-ARIA will not effect any user agent that does not understand WAI-ARIA, since the unknown attributes will be ignored.

Chapter 4

Accessibility Testing

Large parts of the World Wide Web continuous to be not accessible to people with disabilities [Lopes et al., 2010]. Blaming this on the incompetence of webmasters and web developers would be to simple. In fact, according to a study by Lazar, Dudley-Sponaugle et al. [2004], many of them claiming to be familiar with web accessibility best practices.

This discrepancy could be explained by not using the right tools to check for accessibility compliance. Although the latest version of the Web Content Accessibility Guidelines [W3C, 2008] was designed to be more testable, limitations of testability of these rules are still in existence [Brajnik, 2008]. Therefore, manual testing strategies as well as automated testing tools are presented in the following sections.

An extensive list of testing tools - “Web Accessibility Evaluation Tools List” [W3C, 2016b] - can be found on the website of the Web Accessibility Initiative (WAI).

4.1 Manual Testing

Manual testing plays an essential role in creating accessible web pages. In fact, checking manually is indispensable as web accessibility guidelines are only covering around 50% of accessibility complications [Power et al., 2012]. Additionally, it is shown by Sloan et al. [2006] that context is an important factor while evaluating a web site. The importance of manual testing is underlined by Vigo et al. [2013] as they show that the capabilities of automated tools are limited and an over-reliance on automation can even have adverse effects.

Manually testing with a screen reader or trying to navigate with only a keyboard are two techniques a developer can constantly use during the creation of a web site.

4.1.1 Keyboard Navigation

Keyboard accessibility is one of the most important aspects of web accessibility [WebAIM, 2016a]. Therefore, testing with a keyboard should be an integrated part of every accessibility test.

The method is simple: the website should be usable by only using the `TAB` key to navigate through form controls and hyperlinks. The `Enter` or `Spacebar` keys are used to select and activate elements (e.g. buttons).

Common issues one has to look out for, are:

- missing focus indicators
- wrong navigation order
- items that cannot receive keyboard focus
- custom widgets are inaccessible
- long list of navigation items

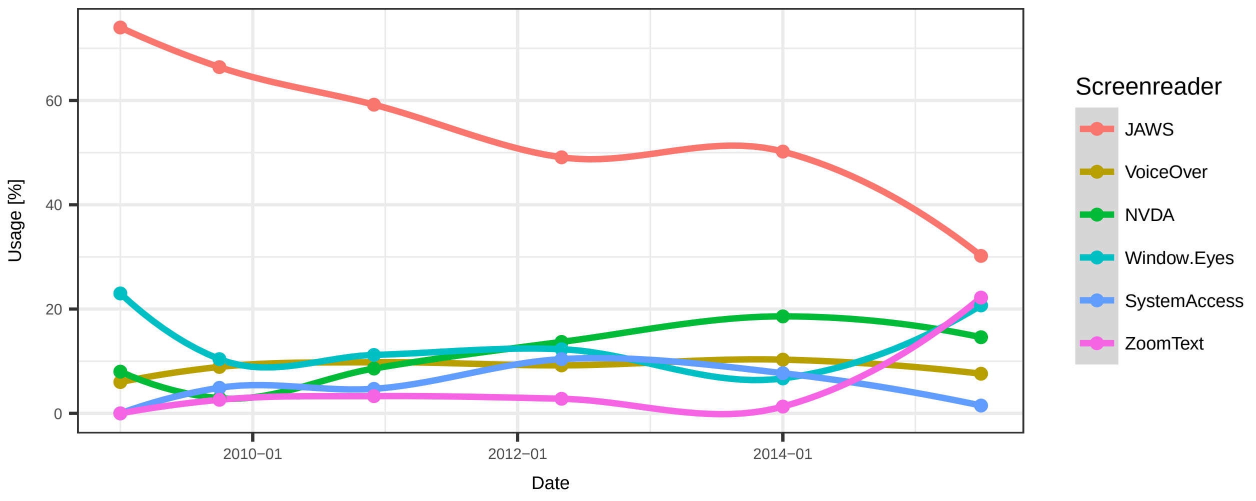


Figure 4.1: Historical trends for primary screen reader usage. [Data from [WebAIM, 2016b]]

How to avoid these problems and best practices are described in Chapter 3 and in detail under "Keyboard and Structural Navigation" in the WAI-ARIA Authoring Practices 1.1 [W3C, 2016a].

4.1.2 Screen Readers

Users of screen readers are facing various challenges while surfing the Web [Lazar, Allen et al., 2007]. Web sites could be perfectly accessible, but are still hard to listen to with a screen reader. For instance, the content of certain web pages (Wikipedia, ...) may contain a huge amount of links which will constantly interrupt the flow with announcements of hyperlinks. In addition, screen reader software naturally has to continuously catch up to the constant development in web technologies (HTML 5.X). And in the worst case, users have to deal with content (Flash, Silverlight, ...) which is just not non-visually accessible.

How screen reader users deal with these challenges and what strategies they are applying was examined by Borodin et al. [2010]. Knowledge of these browsing strategies can help developing new tools and assist in designing accessible websites.

Desktop Screen Readers

The following list provides an overview of the most used screen readers, according a study done by WebAIM [2016b]. Usage trends over time are visualised in Figure 4.1. JAWS' dominant marked share significantly declined in previous years. The leader in market share is Ai Squared, which distribute ZoomText and Window Eyes respectively.

Additionally, two browser extensions are mentioned. ChromeVox and Claws, which are especially useful for developers. Notably, Claws is a screen reader emulator which presents the output of a screen reader in text form. A feature which could be extraordinarily useful in crowded office environments. An example output of Claws can be seen in figure 4.2.

JAWS (**J**ob **A**ccess **W**ith **S**peech) is currently the most used desktop reader (see figure 4.1), although its popularity is declining [WebAIM, 2016b]. The first version was published in January 1995. JAWS is still actively maintained. Licenses start selling at USD 900.

Developer: Freedom Scientific

License: Proprietary

Operating System: Microsoft Windows

Website: www.freedomscientific.com/fs_products/JAWS_HQ.asp

ZoomText is still under active development, despite the fact that its first version was released for DOS in 1988. A license can be bought starting at USD 600.

Developer: Ai Squared
License: Proprietary
Operating System: Microsoft Windows
Website: www.aisquared.com/products/zoomtext/

Window Eyes was first released in 1995 by GW Micro. Recently, GW Micro merged with Ai Squared, the developer of ZoomText.

Developer: Ai Squared / GW Micro
License: Proprietary
Operating System: Microsoft Windows
Website: www.aisquared.com/products/window-eyes/

NVDA (**Non**Visual **D**esktop **A**ccess) was originally started by Michael Curran in 2006. As an open source project, it has more than 50 contributors, according to the projects Github page. The project is still under active development. The software can be downloaded free of charge. To support the project, users are able to donate via the website.

Developer: Michael Curran and others
License: GNU General Public License (Version 2)
Operating System: Microsoft Windows
Website: www.nvaccess.org

Orca is a free and open source software project since 2006. Originally, the project was started by Sun Microsystems, Inc. (now Oracle). It is the default screen reader for the GNOME desktop environment and it is therefore included in most major UNIX-like operating systems.

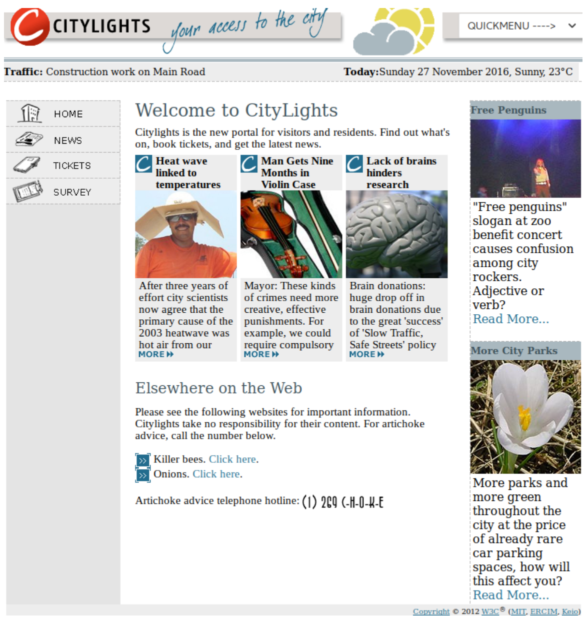
Maintainer: Joanmarie Diggs
License: GNU LGPL (version 2.1)
Operating System: Linux, BSD, Solaris
Website: projects.gnome.org/orca

ChromeVox is an extention to the Chrome Web-Browser, as well as the default screen reader in ChromeOS. It is made by Google. A successor - “ChromeVox Next” - is currently in development. It can be downloaded free of charge from the Chrome WebStore.

Developer: Google
License: Proprietary
Operating System: ChromeOS or a version of Chrome
Website: www.chromevox.com

Claws is a screen reader emulator. A user can choose between three output modes: JAWS, NVDA or a custom verion. See Figure 4.2 for an example.

Developer: Sergio Luján Mora
License: Mozilla Public License, version 2.0
Requirements: Firefox 30.0 - 50.*
Website: addons.mozilla.org/en-US/firefox/addon/claws/



CITYLIGHTS your access to the city

QUICKMENU ---->

Traffic: Construction work on Main Road **Today:**Sunday 27 November 2016, Sunny, 23°C

HOME
NEWS
TICKETS
SURVEY

Welcome to CityLights

Citylights is the new portal for visitors and residents. Find out what's on, book tickets, and get the latest news.

Heat wave linked to temperatures
After three years of effort city scientists now agree that the primary cause of the 2003 heatwave was hot air from our
MORE

Man Gets Nine Months in Violin Case
Mayor: These kinds of crimes need more creative, effective punishments. For example, we could require compulsory
MORE

Lack of brains hinders research
Brain donations: huge drop off in brain donations due to the great 'success' of 'Slow Traffic, Safe Streets' policy
MORE

Elsewhere on the Web

Please see the following websites for important information. Citylights take no responsibility for their content. For artichoke advice, call the number below.

Killer bees. Click here.
Onions. Click here.

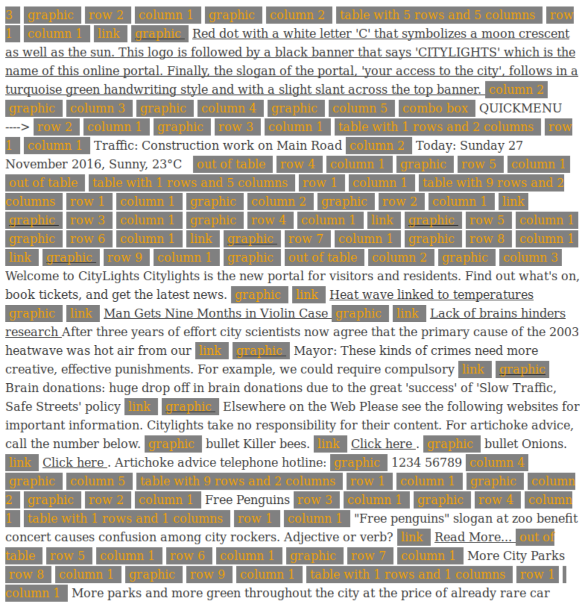
Artichoke advice telephone hotline: (1) 269 C-H-O-K-E

Free Penguins
"Free penguins" slogan at zoo benefit concert causes confusion among city rockers. Adjective or verb?
Read More...

More City Parks
More parks and more green throughout the city at the price of already rare car parking spaces, how will this affect you?
Read More...

Copyright © 2012 W3C® (MIT, ERCIM, Keio)

(a) Demo of an inaccessible website by W3C [2012].



3 graphic row 2 column 1 graphic column 2 table with 5 rows and 5 columns row 1 column 1 link graphic Red dot with a white letter 'C' that symbolizes a moon crescent as well as the sun. This logo is followed by a black banner that says 'CITYLIGHTS' which is the name of this online portal. Finally, the slogan of the portal, 'your access to the city', follows in a turquoise green handwriting style and with a slight slant across the top banner. column 2 graphic column 3 graphic column 4 graphic column 5 combo box QUICKMENU ----> row 2 column 1 graphic row 3 column 1 table with 1 rows and 2 columns row 1 column 1 Traffic: Construction work on Main Road column 2 Today: Sunday 27 November 2016, Sunny, 23°C out of table row 4 column 1 graphic row 5 column 1 out of table table with 1 rows and 5 columns row 1 column 1 table with 9 rows and 2 columns row 1 column 1 graphic column 2 graphic row 2 column 1 link graphic row 3 column 1 graphic row 4 column 1 link graphic row 5 column 1 graphic row 6 column 1 link graphic row 7 column 1 graphic row 8 column 1 link graphic row 9 column 1 graphic out of table column 2 graphic column 3 Welcome to CityLights Citylights is the new portal for visitors and residents. Find out what's on, book tickets, and get the latest news. graphic link Heat wave linked to temperatures graphic link Man Gets Nine Months in Violin Case graphic link Lack of brains hinders research After three years of effort city scientists now agree that the primary cause of the 2003 heatwave was hot air from our link graphic Mayor: These kinds of crimes need more creative, effective punishments. For example, we could require compulsory link graphic Brain donations: huge drop off in brain donations due to the great 'success' of 'Slow Traffic, Safe Streets' policy link graphic Elsewhere on the Web Please see the following websites for important information. Citylights take no responsibility for their content. For artichoke advice, call the number below. graphic bullet Killer bees. link Click here . graphic bullet Onions. link Click here . Artichoke advice telephone hotline: graphic 1234 56789 column 4 graphic column 5 table with 9 rows and 2 columns row 1 column 1 graphic column 2 graphic row 2 column 1 Free Penguins row 3 column 1 graphic row 4 column 1 table with 1 rows and 1 columns row 1 column 1 "Free penguins" slogan at zoo benefit concert causes confusion among city rockers. Adjective or verb? link Read More... out of table row 5 column 1 row 6 column 1 graphic row 7 column 1 More City Parks row 8 column 1 graphic row 9 column 1 table with 1 rows and 1 columns row 1 column 1 More parks and more green throughout the city at the price of already rare car

(b) Output of *Claws* [2016].

Figure 4.2: An example how a screen reader would interpret an inaccessible website using the emulator *Claws* [2016] for Mozilla Firefox.

	December 2010	May 2012	January 2014	July 2015
VoiceOver	27%	49%	61%	57%
TalkBack	3%	5%	22%	18%
Nuance Talks	30%	18%	16%	5%
Mobile Speak	16%	9%	6%	2%

Table 4.1: Mobile screen reader usage. [Data from [WebAIM, 2016b]]

Mobile Screen Readers

The importance of testing web sites on mobile devices is increasing year over year, as mobile phone usage is rapidly growing. Consequently, developers should also test for accessibility problems on mobile devices.

Usage trends, according to a survey by WebAIM [2016b], are presented in Table 4.1. VoiceOver from Apple still holds the biggest market share. TalkBack from Google, which is only used by 18%, comes pre-installed on most Android devices.

4.2 Automated Testing

Testing manually for accessibility problems may play a bigger part, compared to conventional software testing, according to Vigo et al. [2013]. Nevertheless, automated tools are indispensable, as they provide developers with immediate feedback during development. Additionally, results are reproducible quickly and at any time. Thus, automated accessibility tests should also be executed by a Continuous Integration (CI) system.

The Web Accessibility Initiative (WAI) provides an exhaustive list of testing tools: “Web Accessibility Evaluation Tools List” [W3C, 2016b]. However, in the following sections, a few selected tools are described in detail.

```
$ phantomjs tools/runner/audit.js <url-or-filepath>
```

Listing 4.1: Running ADT as a Standalone Tool.

```
$ npm install a11y
$ a11y http://exampl.org http://example.com
```

Listing 4.2: Testing Multiple Web Pages with A11y.

4.2.1 Accessibility Developer Tools

Accessibility Developer Tools (ADT) is an open source library made by Google [2016]. The JavaScript based project provides a collection of audit rules checking for common accessibility problems.

According to the projects documentation [Google, 2016], some of the feature the library provides are:

- contrast ratio calculation and color suggestions
- retrieving and validating ARIA attributes and states
- accessible name calculation using an algorithm by W3C [2014]

Although, the primary goal of this project is to provide an audit API, it can also be used as a standalone tool. Listing 4.1 shows how to run ADT via phantomjs.

Installing ADT is as simple as cloning the projects source code repository. But, only recently, it became possible, to install the project via `npm`, which is even more convenient.

Finally, it is noted, that there exists an extension [Google Accessibility, 2015] to the Chrome web browser, which provides the same results in the Chrome Developer Tools.

Node.js

Based on Google's ADT API, Addy Osmani build a command line tool and a Node.js module named `A11y` [Osmani, 2016]. This project provides various conveniences over using the underlying ADT library. For example, installing (including all dependencies like phantomjs) and testing multiple endpoints at once is shown in Listing 4.2.

Integrating `A11y` in automated tests is possible by using the provided API. Listing 4.3 demonstrates how to use this project as a Node.js module. In this example, two types of differently formatted report output are obtained.

Additionally, it is possible to try `A11y` directly using only a web browser on `a11y-app.herokuapp.com`.

Ruby on Rails

`capybara-accessible` by Case Commons [2016] is built to integrate accessibility tests in Ruby on Rails integration tests. To accomplish this, the authors used Google's ADT library. Additionally, they could remove

```
let a11y = require('a11y');

a11y('http://example.com', (err, reports) => {
    let output = JSON.parse(reports);
    // a11y custom format
    let audit = output.audit;
    // DevTools Accessibility Audit formatted report
    let report = output.report;
});
```

Listing 4.3: Using A11y as a Node.js Module.

the dependencies on phantomjs by using capybara - a web driver abstraction layer. Which has the advantage, that existing projects, which already are using Selenium oder Poltergeist, can easily integrate ADT tests.

Adding `capybara-accessible` as a dependency to the Rails project is sufficient to get a fully working accessibility test environment. The output of an inaccessible web page, as seen in Figure 4.2a is shown in Listing 4.4

```
rails@41638854b053:~/a11y$ rspec spec/features/demos_spec.rb
F

Failures:

  1) Demo a11y tests demo feature
     Got 0 failures and 2 other errors:

     1.1) Failure/Error: visit '/demos'

          Capybara::Accessible::InaccessibleError:
            Found at http://127.0.0.1:37215/demos

            *** Begin accessibility audit results ***
            An accessibility audit found
            Errors:
            Error: AX_TEXT_01 (Controls and media elements should have labels) failed on
                the following element:
            TABLE > TBODY > TR > TD > TABLE > TBODY > TR:nth-of-type(2) > TD:nth-of-type
                (2) > TABLE > TBODY > TR > TD:nth-of-type(5) > SELECT
            See https://github.com/GoogleChrome/accessibility-developer-tools/wiki/Audit-
                Rules#-ax_text_01--controls-and-media-elements-should-have-labels for
                more information.

            Warnings:
            Warning: AX_TEXT_02 (Images should have an alt attribute) failed on the
                following elements (1 - 5 of 33):
            [...]
            TABLE > TBODY > TR > TD > TABLE > TBODY > TR:nth-of-type(2) > TD:nth-of-type
                (2) > TABLE > TBODY > TR > TD:nth-of-type(2) > IMG
            See https://github.com/GoogleChrome/accessibility-developer-tools/wiki/Audit-
                Rules#-ax_text_02--images-should-have-an-alt-attribute-unless-they-have-
                an-aria-role-of-presentation for more information.

            Warning: AX_TITLE_01 (The purpose of each link should be clear from the link
                text) failed on the following elements (1 - 2 of 2):
            #content > P:nth-of-type(2) > SPAN > A
            #content > P:nth-of-type(2) > SPAN > A:nth-of-type(2)

            Warning: AX_COLOR_01 (Text elements should have a reasonable contrast ratio)
                failed on the following elements (1 - 4 of 4):
            [...]
            TABLE > TBODY > TR > TD > TABLE > TBODY > TR:nth-of-type(2) > TD:nth-of-type
                (2) > TABLE:nth-of-type(2) > TBODY > TR > TD:nth-of-type(5) > TABLE >
                TBODY > TR:nth-of-type(7) > TD > FONT > B
            See https://github.com/GoogleChrome/accessibility-developer-tools/wiki/Audit-
                Rules#-ax_color_01--text-elements-should-have-a-reasonable-contrast-ratio
                 for more information.

            Warning: AX_IMAGE_01 (Meaningful images should not be used in element
                backgrounds) failed on the following elements (1 - 4 of 4):
            [...]
            TABLE > TBODY > TR > TD > TABLE > TBODY > TR:nth-of-type(3) > TD:nth-of-type
                (3)
            See https://github.com/GoogleChrome/accessibility-developer-tools/wiki/Audit-
                Rules#-ax_image_01--meaningful-images-should-not-be-used-in-element-
                backgrounds for more information.

            *** End accessibility audit results ***

Finished in 1.19 seconds (files took 1.9 seconds to load)
1 example, 1 failure
```

Listing 4.4: Output of capybara-accessible

Appendix A

About The Authors

Christoph Lipautz

Christoph is a student in computer science at the University of Technology in Graz and a full-stack developer that has worked for the last decade with focus on the web. He is the author of chapter WAI-ARIA.

Christian Mayer

Christian has more than 12 years of professional experience in developing and managing large web projects. He is the author of chapter Accessibility testing.

Stephanie Reich

Stephanie is currently in the 7th semester of Mathematics and Informatics for a teaching degree. Beside her colleague Mario Windpessl, she is one of the author of the chapters Motivation and Design Patterns. Primarily, she was writing about the motivation and general aspects of doing accessible web design and also about the visual design of accessible web pages.

Mario Windpessl

Mario is in the 7th semester of Mathematics and Informatics for a teaching degree at Karl Franzens University and University of Technology in Graz. He is author of chapter Design Patterns, primarily Navigation and Non-text-content.

Bibliography

Bootstrap v2.3.2 [2013]. 2013. `http://getbootstrap.com/2.3.2/components.html#alerts` (cited on page 13).

Bootstrap v3.3.7 [2016]. 2016. `http://getbootstrap.com/components/#alerts` (cited on page 14).

Borodin, Yevgen, Jeffrey P. Bigham, Glenn Dausch and I. V. Ramakrishnan [2010]. “More Than Meets the Eye: A Survey of Screen-reader Browsing Strategies”. In: *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A) - W4A ’10*. W4A ’10. Raleigh, North Carolina: Association for Computing Machinery (ACM), 2010, 13:1–13:10. ISBN 978-1-4503-0045-2. doi:10.1145/1805986.1806005. `http://dx.doi.org/10.1145/1805986.1806005` (cited on page 20).

Brajnik, Giorgio [2008]. “A comparative test of web accessibility evaluation methods”. In: *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility - Assets ’08*. Association for Computing Machinery (ACM), 2008. doi:10.1145/1414471.1414494. `http://dx.doi.org/10.1145/1414471.1414494` (cited on page 19).

Case Commons [2016]. *capybara-accessible. Capybara driver that makes accessibility assertions in RSpec feature tests*. 2016. `https://github.com/Casecommons/capybara-accessible/` (cited on page 23).

Claws [2016]. 2016. `https://github.com/chmutoff/claws` (cited on page 22).

Google [2016]. *Accessibility Developer Tools*. 2016. `https://github.com/GoogleChrome/accessibility-developer-tools` (cited on page 23).

Google Accessibility [2015]. *Accessibility Developer Tools. Chrome Extension*. 2015. `https://chrome.google.com/webstore/detail/accessibility-developer-t/fpkknkljclfencbdbgkenhalefipecmb?hl=en` (cited on page 23).

Griffin, Emily [2016]. *Tips for making web video and audio accessible*. 2016. `http://www.3playmedia.com/2015/07/13/tips-for-making-web-video-audio-accessible/` (cited on pages 7, 9).

Lazar, Jonathan, Aaron Allen, Jason Kleinman and Chris Malarkey [2007]. “What Frustrates Screen Reader Users on the Web: A Study of 100 Blind Users”. *International Journal of Human-Computer Interaction* 22.3 [May 2007], pages 247–269. doi:10.1080/10447310709336964. `http://dx.doi.org/10.1080/10447310709336964` (cited on page 20).

Lazar, Jonathan, Alfreda Dudley-Sponaugle and Kisha-Dawn Greenidge [2004]. “Improving web accessibility: a study of webmaster perceptions”. In: volume 20. 2. Elsevier BV, Mar 2004, pages 269–288. doi:10.1016/j.chb.2003.10.018. `http://dx.doi.org/10.1016/j.chb.2003.10.018` (cited on page 19).

Lopes, Rui, Daniel Gomes and Luis Carrico [2010]. “Web not for all”. In: *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A) - W4A ’10*. ACM. Association for Computing Machinery (ACM), 2010, page 10. doi:10.1145/1805986.1806001. `http://dx.doi.org/10.1145/1805986.1806001` (cited on page 19).

Osmani, Addy [2016]. *A11y. Accessibility audit tooling for the web*. 2016. `https://addyosmani.com/a11y/` (cited on page 23).

Pickering, Heydon [2014]. *Apps For All: Coding Accessible Web Applications*. Smashing Magazine GmbH, 2014. ISBN 978-3-94454079-5 (cited on page 17).

Pickering, Heydon [2016]. *Inclusive Design Patterns*. Smashing Magazine GmbH, 2016. ISBN 978-3-945749-43-2 (cited on pages 3–4, 12).

Power, Christopher, André Freire, Helen Petrie and David Swallow [2012]. "Guidelines are only half of the story". In: *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. Association for Computing Machinery (ACM), 2012. doi:10.1145/2207676.2207736. http://dx.doi.org/10.1145/2207676.2207736 (cited on page 19).

Sloan, David, Andy Heath, Fraser Hamilton, Brian Kelly, Helen Petrie and Lawrie Phipps [2006]. "Contextual web accessibility - maximizing the benefit of accessibility guidelines". In: *Proceedings of the 2006 international cross-disciplinary workshop on Web accessibility (W4A) Building the mobile web: rediscovering accessibility? - W4A*. Association for Computing Machinery (ACM), 2006. doi:10.1145/1133219.1133242. http://dx.doi.org/10.1145/1133219.1133242 (cited on page 19).

Storey, David [2016]. *Permanent or situational dissabilitys*. 2016. https://twitter.com/dstorey/status/649636741033279488 (cited on page 2).

Vigo, Markel, Justin Brown and Vivienne Conway [2013]. "Benchmarking web accessibility evaluation tools". In: *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility - W4A '13*. ACM. Association for Computing Machinery (ACM), 2013, page 1. doi:10.1145/2461121.2461124. http://dx.doi.org/10.1145/2461121.2461124 (cited on pages 19, 22).

W3C [2008]. *Web content accessibility guidelines (WCAG) 2.0*. 2008. https://www.w3.org/TR/WCAG20/ (cited on page 19).

W3C [2012]. *Inaccessible Home Page - Before and After Demonstration*. 2012. https://www.w3.org/WAI/demos/bad/before/home.html (cited on page 22).

W3C [2014]. *Text Alternative Computation*. 2014. https://www.w3.org/TR/wai-aria/roles#textalternativecomputation (cited on page 23).

W3C [2016a]. *WAI-ARIA Authoring Practices 1.1*. 2016. https://www.w3.org/TR/wai-aria-practices-1.1/ (cited on pages 11, 13, 20).

W3C [2016b]. *Web Accessibility Evaluation Tools List*. 2016. https://www.w3.org/WAI/ER/tools/ (cited on pages 19, 22).

W3Schools [2016a]. *HTML Language Code Reference*. 2016. https://www.w3.org/TR/wai-aria/roles#role_definitions (cited on pages 3, 5).

W3Schools [2016b]. *HTML main tag*. 2016. http://www.w3schools.com/tags/tag_main.asp (cited on page 3).

WAI [2016a]. *Definition of Roles*. 2016. https://www.w3.org/TR/wai-aria/roles#role_definitions (cited on page 11).

WAI [2016b]. *Images Concepts*. 2016. https://www.w3.org/WAI/tutorials/images/ (cited on page 6).

WAI [2016c]. *Making the Web Accessible*. 2016. https://www.w3.org/WAI/intro/accessibility.php (cited on pages 1, 3).

WAI [2016d]. *Supported States and Properties*. 2016. https://www.w3.org/TR/wai-aria/states_and_properties (cited on page 12).

WAI [2016e]. *WAI-ARIA Overview*. 2016. https://www.w3.org/WAI/intro/aria (cited on pages 11, 18).

WebAIM [2016a]. *Keyboard Accessibility*. 2016. http://webaim.org/techniques/keyboard/ (cited on page 19).

WebAIM [2016b]. *Screen Reader User Survey #6 Results*. 2016. `http://webaim.org/projects/screenreadersurvey6/` (cited on pages 20, 22).