

Evergreen Application

ACCESSIBILITY EVALUATION

The goal of this initial evaluation is to convey a general overview of Web Accessibility and the primary aspects of the website(s) that do not conform to WCAG 2.1 A and AA Success Criteria.

Date of Evaluation: June 14 - 27, 2019

Table of Contents

Evergreen Application	1
Evaluation Process and Deliverables	4
Evaluation Environment.....	4
Background Summary.....	5
Pages Reviewed	5
Executive Summary	6
Video Demonstration of the Georgia Evergreen Application	6
Overall Conformance	6
Remediation Needed	8
Reviewer's Report	10
Detailed Feedback & Recommendations.....	10
1. WCAG 2.0 1.1.1 A: Appropriate use of alternative text.....	10
2. WCAG 2.0 1.3.1 A: Information and Relationships	11
3. WCAG 2.0 1.3.1 A: Information and Relationships	11
4. WCAG 2.0 1.3.1 A: Information and Relationships	12
5. WCAG 2.0 1.4.1 A: Use of Color	12
6. WCAG 2.0 1.4.3 AA: Color Contrast	13
7. WCAG 2.0 1.4.5 AA: Images of Text.....	14
8. WCAG 2.0 2.1.1 A: Keyboard Access	14
9. WCAG 2.0 2.4.3 A: Focus Order.....	14
10. WCAG 2.0 2.4.4 A: Link Purpose in Context	15
11. WCAG 2.0 2.4.7 AA: Focus Visible - Focus States & Indicators	15
12. WCAG 2.0 3.3.1 A: Error Identification	16
13. WCAG 2.0 4.1.1 A: Validate HTML	17
Appendix A: Commonly Used Code Snippets.....	17
Links that open new windows, dialogs or PDFs	17
Hiding Content from both visual and screen reader users	19
HTML and WAI-ARIA Landmarks	19
Add Skip Navigation.....	19
Content for screen reader users but not sighted users	20
OPTGROUP to group OPTION elements inside a SELECT	20

Tab Panel	20
Example:	23
Keyboard	23
Accessibility	24
Tables	24
Data Tables	25
Layout Tables.....	25
Scalable Font and Element Widths (Sizes).....	26
EM and REM	26
Approaches & Responsive Design	26
Setting <body> font-size to ="62.5%"	26
Using button elements without the default formatting	27
Hiding and showing content with JavaScript DOM Coding.....	27
ARIA, Dynamic Content, alerts, and feedback	30
ARIA and Dynamic Content	30
WAI-ARIA	30
Live Regions	31
ARIA Menus	31
Stack Overflow Flyout Menu Summary.....	31
Appendix B: Screenshots.....	32
Appendix C: Useful Resources.....	36
W3C's Web Accessibility Initiative (WAI)	36
Company Information	37
Reviewers.....	37

Evaluation Process and Deliverables

The evaluation process includes a combination of assistive technology solutions, detailed code inspection, and manual analysis of the selected URLs. A brief mp4 video is also being included under the heading “Video Demonstration of the Georgia Evergreen Application” that highlights some of the aspects documented in this evaluation using assistive technology solutions commonly used by people with disabilities.

The evaluation process is a systematic testing of WCAG 2.1 A and AA Success Criteria using both automated tools and manual checks. Automated tools used may include any combination of browser Extensions, Addons or toolbars: the WebAIM WAVE extension, Deque aXe extension, HTML_CodeSniffer, Paciello Group Web Accessibility Toolbar and Colour Contrast Analyser, NCSU Color Contrast Analyzer, SSA ANDI, and Firebug. Other tools may be included as they become available.

This report provides a detailed analysis of specific URLs selected for this review, and their level of conformance with WCAG 2.1 Level A and AA. Solutions for remediation to meet WCAG 2.1 Level A and AA is also provided. The selected URLs and issues identified are within a specific point in time, representing a snapshot of the site’s current state. Issues identified in this report may therefore become modified, obsolete, or removed with future website iterations and releases.

Accessibility issues detailed in this report are a representation of the site in general. Many of the recommendations, outlined in this review, can be applied to the rest of the website to improve accessibility conformance overall.

Incorporating and maintaining accessibility requires ongoing testing and monitoring throughout the lifecycle of a given website, including its design and development stages. Doing so, will provide greater assurance that new or additional accessibility barriers will not be introduced.

The latter part of this document, beginning with the section “Detailed Feedback & Recommendations,” provides specific code remediation examples and solutions intended for developers. The accompanying Excel spreadsheet, also intended for developers, serves as a companion reference and provides both usability and programmatic information with specific references and code remediation solutions related to the Georgia Evergreen website.

Evaluation Environment

Software/Configuration Used for Testing

Browsers Used for Testing:

Firefox- 67.0.1

Chrome - 75.0.3770.80

Screen Readers Used for Testing:

JAWS (Version 18) for Windows
NVDA (2019) for Windows

Operating Systems Used for Testing:

Windows 10

Screen Resolutions Used for Testing:

1920 x 1080

Background Summary

To provide the most robust sampling of testing for the evaluation process of the Georgia Evergreen application, the URLs selected for this review were based on the following criteria:

Pages with the greatest amount of traffic, as determined by analytics

- Most frequently visited pages, which could be determined by your internal analytics tools.
- Critical pages of importance. For example, essential forms or processes that need to be completed for customers to receive essential information and services.
- Pages containing unique elements such as multimedia, dynamic elements, tables, etc.

Pages Reviewed

URLs include the following (Refer to [Appendix B](#) for Screenshots)

1. Login - <https://terran-testbox.gapines.org/eg/staff/>
2. Home page - <https://terran-testbox.gapines.org/eg/staff/>
3. Patron Search - <https://terran-testbox.gapines.org/eg/staff/circ/patron/search>
4. Patron Book Check Out - <https://terran-testbox.gapines.org/eg/staff/circ/patron/87/checkout?card=99999303411>
5. Patron Book Holds - <https://terran-testbox.gapines.org/eg/staff/circ/patron/87/holds>
6. Register patron - <https://terran-testbox.gapines.org/eg/staff/circ/patron/register>
7. Book check in - <https://terran-testbox.gapines.org/eg/staff/circ/checkin/checkin>

Primary contact for the project: John Rempel - jrempel3@gatech.edu

Executive Summary

This accessibility report is an in-depth evaluation of a predetermined sampling of URLs of the Evergreen application that documents the types of accessibility issues and conformance violations in accordance with the W3C's international set of guidelines known as Web Content Accessibility Guidelines (WCAG 2.1 Level AA). For additional information related to WCAG, visit: [Web Content Accessibility Guidelines \(WCAG\) Overview](#). Recently updated Section 508 standards, also known as ICT Refresh, is a federally recognized set of standards that has been harmonized with WCAG 2.1 (Level A and AA). For additional information on the ICT Refresh, visit: [About the ICT Refresh](#). It should also be noted that with recent cases in which the Department of Justice intervened, conformance to WCAG 2.1 (Level A and AA) were required when applicable. For additional information, visit: [ADA.gov](#)

The Evergreen application is fairly accessible. Most of the issues found were minor. The most prevalent issue found was the lack of accessible labels for the various form fields within the site. Another noticeable issue was the sound made when users activate the submit button for certain forms. This sound is not expected by users and can take them by surprise.

Many of the user impacts are highlighted in the video demonstration. Additional information is also available in the “Detailed Findings & Recommendations” section below and the accompanying spreadsheet. Resources listed for follow-up study are also provided in the “Results and Recommended Actions” section below. Feedback on this evaluation is welcome.

Video Demonstration of the Georgia Evergreen Application

A brief mp4 video has been created of the Georgia PINES website that gives a basic understanding of how people with disabilities access the website, and highlights some of the barriers that may be encountered. The video is to be considered a supplemental component to the Georgia PINES web accessibility review and accompanying Excel spreadsheet, which provides additional detail and remediation solutions. The video in its current state does not contain captions, but AMAC would be happy to provide captions at no additional cost upon request if anyone on your team would benefit. Because the current video does not contain captions, we request that it not be posted online or shared publicly. To access the video, visit: [Georgia Evergreen Application Accessibility Video](#).

Overall Conformance

The website meets partial conformance with WCAG 2.1 Level A and AA Success Criteria (SC). The Success Criteria listed here pass conformance for the pages reviewed. Conformance with an SC may be because no feature was identified to test (Conforms - not applicable), while the other SCs pass testing (Pass - Commendable). This conformance is for the period of time when the review was performed and does not apply to future releases of the website.

WCAG 2.1 Success Criteria	Notes
1.2.1 A: Audio-only and Video-only (Prerecorded)	Conforms - Not Applicable
1.2.2 A: Captions (Prerecorded)	Conforms - Not Applicable
1.2.3 A: Audio Description or Media Alternative (Prerecorded)	Conforms - Not Applicable
1.2.4 AA: Captions (Live)	Conforms - not applicable
1.2.5 AA: Audio Description (Prerecorded)	Conforms - not applicable
1.3.3 A: Sensory Characteristics	Conforms - not applicable
1.3.4 AA: Orientation	PASS - Commendable
1.4.2 A: Audio Control	Conforms - not applicable
1.4.4 AA: Resize text	PASS - Commendable
1.4.10 AA: Reflow	PASS - Commendable
1.4.11 AA: Non-text Contrast	PASS - Commendable
1.4.12 AA: Text Spacing	PASS - Commendable
1.4.13 AA: Content on Hover or Focus	PASS - Commendable
2.1.4 A: Character Key Shortcuts	Conforms - not applicable
2.2.1 A: Timing Adjustable	Conforms - not applicable
2.2.2 A: Pause, Stop, Hide	Conforms - not applicable
2.3.1 A: Three Flashes or Below Threshold	Conforms - not applicable
2.4.5 AA: Multiple Ways	PASS - Commendable
2.5.1 A: Pointer Gestures	PASS - Commendable
2.5.2 A: Pointer Cancellation	PASS - Commendable
2.5.4 A: Motion Actuation	Conforms - not applicable
3.1.1 A: Language of Page	PASS - Commendable
3.1.2 AA: Language of Parts	PASS - Commendable
3.2.1 A: On Focus	PASS - Commendable
3.2.2 A: On Input	PASS - Commendable
3.2.3 AA: Consistent Navigation	PASS - Commendable
3.2.4 AA: Consistent Identification	PASS - Commendable
3.3.2 A: Labels or Instructions	PASS - Commendable
3.3.3 AA: Error Suggestion	PASS - Commendable
3.3.4 AA: Error Prevention (Legal, Financial, Data)	PASS - Commendable

Remediation Needed

The table below is organized by the WCAG 2.0 A and AA Success Criteria where issues have been identified that need remediation to conform to the criteria. The issues include the impact to users on a scale defined in the legend.

Also included, is the estimated level of effort for remediation. The level of effort to remediate outstanding accessibility issues is an estimation of the level of complexity, cost and time involved with remediation based on the number of instances and a general understanding of the implementation required.

The impact and level of effort are provided to assist with determining the priority for issues to be remediated. Critical impact issues should be addressed first, and the level of effort is helpful in sizing the task for sprint/task planning.

Legend:

1. **WCAG 2.0 A and AA Success Criteria - The three part numbers (e.g. 1.1.1)** refer to the WCAG 2.0 checkpoints (<http://www.w3.org/TR/WCAG20/>).
2. **Success Criteria Non-conformance** - The specific violation(s) identified in the web page or document.
3. **Impact** - The accessibility impact is an indication of the severity to people with disabilities and the distribution of the issue(s) across the sampled pages.
 - **Critical** - Points out the issues that will block access for a person with a disability or would make the content very difficult to understand. The issue will cause the page to fail compliance.
 - **High** - Affected users will have major difficulties accessing components of the site or application and may be unable to complete tasks independently. They may be unable to complete forms; they may find content difficult to locate and navigate to; they may be unable to access some content at all. If there are enough barriers, users may abandon the site and not return. User experience will be low, and users may need additional assistance or accommodation. The issue may cause the page to fail compliance.
 - **Medium** - Affected users will have moderate difficulties accessing components of the site or application. They may have difficulties understanding site content, navigating the site, and interacting with content; they may find the site cumbersome and their usage slow; they may be unable to locate and use some information. User experience will be low.
 - **Low** - Affected users will have minor difficulties accessing components of the site or application. They may find the site annoying; they may feel that they are not the target audience for the site; they may have difficulties locating some information or interacting with the site. User may become annoyed with the site and be reluctant to return.

4. **Level of Effort** - Based on the reviewer's experience, the estimated effort that would need to be exerted to remediate the issue throughout the documents.

Issue Summary by Success Criteria

WCAG 2.0	Success Criteria Non-conformance	Impact	Level of Effort
1.1.1 A: Non-text Content	Alt text is provided for decorative images.	High	Low
1.3.1 A: Info and Relationships	There were several instances of elements that do not have the correct semantic markup. There were also instances of form fields that did not have programmatically associated labels.	Critical	Low
1.3.2 A: Meaningful Sequence	On some of the elements, reading and tab order is illogical.	Critical	Low
1.4.1 A: Use of Color	There is an instance of color being used to convey information to users.	Critical	Low
1.4.3 AA: Contrast (Minimum)	A few instances of insufficient color contrast were found.	High	Low
1.4.5 AA: Images of Text	The alt text for the PINES logo is inadequate.	Medium	Low
2.1.1 A: Keyboard Access	A link was found on the patron registration page that is unable to receive keyboard focus.	High	Low
2.1.2 A: No Keyboard Trap	Form fields are causing minor keyboard traps.	Medium	Medium
2.4.1 A: Bypass Blocks	The heading structure of the site could be improved. The site is also missing a skip to main content link.	High	Low
2.4.2 A: Page Titled	Each page's title is read in a confusing manner via screen readers.	Medium	Low
2.4.3 A: Focus Order	On some of the elements, reading and tab order is illogical.	Critical	Low
2.4.4 A: Link Purpose (In Context)	Link text of some links could be improved.	High	Low
2.4.6 AA: Headings and Labels	Various form fields are missing labels.	Critical	Low
2.4.7 AA: Focus Visible	The links in the navigation bar lack visible focus.	Critical	Low
3.2.2 A: On Input	Placeholder text is used as a label form a few of the form fields on the site.	Critical	Low
3.3.1 A: Error Identification	There error identification method for the login form is inadequate.	Critical	Low

4.1.1 A: Parsing	There were some parsing issues found when running the source code through W3C's HTML Validator.	<i>Medium</i>	<i>Medium</i>
4.1.2 A: Name, Role, Value	Various name, role, and value issue were found across the site.	Critical	<i>Low</i>
4.1.3 AA: Status Messages	One status message was found on the patron registration form.	<i>High</i>	<i>Low</i>

Reviewer's Report

Detailed Feedback & Recommendations

The list below is organized by WCAG 2.1 A and AA Success Criteria. Each Success Criteria contains an overall explanation of how an issue does not conform to the criteria. A detailed example of an issue is provided along with recommendations for remediation. Detailed test results have been captured on a page-by-page basis and are provided in a separate Excel Spreadsheet.

Recommendations are informational as there may be other methods available to fix the issues within the agency environment or that have been newly developed given new technology and best practices. It is the intention of these findings to provide an understanding of the accessibility issue and allow agencies and development teams the freedom to remediate according to the technology, talent and imaginations available. Meeting the intention of the Success Criteria can be innovative and move ICT (Information and Communication Technology) toward better solutions that provide the best user experience to all people.

1. WCAG 2.0 1.1.1 A: Appropriate use of alternative text

Applicable to:

Home Page

Impact:

[High]

Explanation:

Decorative

Redundant images or images used for spacing convey no information and should be coded to be ignored by assistive technology. In HTML, set alt-text to null, or empty: alt="" (no spaces).

Example: The link icons on the home page are missing alt text attributes.

Current Code:

```

```

Recommended Code:

```

```

2. WCAG 2.0 1.3.1 A: Information and Relationships

Applicable to:

Register Patron Page

Impact:

[High]

Explanation:

[Programmatic Labels](#)

Assistive Technology relies on the programmatic association of form controls to labels to accurately communicate labels when a form control gets focus.

Additional Resources:

[W3C Tutorial Labels](#)

Example: The form fields on the Register Patron page have explicit labels, but those labels are not programmatically associated with the corresponding fields.

Current Code:

```
<label class="ng-binding">Barcode</label>

<input type="text" name="barcode" ...>
```

Recommended Code:

```
<label class="ng-binding" for="barcode">Barcode</label>

<input type="text" name="barcode" id="barcode"...>
```

3. WCAG 2.0 1.3.1 A: Information and Relationships

Applicable to:

GLOBAL

Impact:

[High]

Explanation:

[Heading Structure](#)

Headings are useful for all users and are essential for accessible webpages. For all users, Headings semantically identify and group content. For screen reader users, one of the most common navigation methods is to use the heading structure. Individuals with attention or cognitive impairments benefit from having consistent and sectioned information that is clearly identified. Headings make content easy to parse and understand the intended meaning. For more information, visit: [W3C-Headings](#)

4. WCAG 2.0 1.3.1 A: Information and Relationships

Applicable to:

GLOBAL

Impact:

[Critical]

Explanation:

Data Tables

Data tables are a meaningful way to provide tabular data. Proper semantic markup should be used to define data tables including the summary attribute and <th> with scope attribute to define. Summary information and headers should be concise and provide the purpose of the table or column.

Example: There are several tables, across the entirety of the site, that presented visually as such, but do not contain table markup.

Recommendation:

To ensure that all elements are able to be accessed as intended, provide the correct semantic markup. For elements that are presented as tables, provide the correct markup so that they are able to be accessed via screen reader and other assistive technology as intended.

5. WCAG 2.0 1.4.1 A: Use of Color

Applicable to:

Patron Page

Impact:

[Critical]

Explanation:

Color used to provide information must also have a non-color indicator that provides the same information. For more information, visit: [W3C-Use of Color](#)

Example: On a patron's information summary, the color red is used to profile information about a certain field. For instance, if the birth date is missing, the text is colored red to indicate that this is an issue. Screen reader users will not be able to decipher that this is an issue.

Screenshot:

Profile	Local Administrator
Home Library	BR1
Net Access	Filtered
Date of Birth	
Parent/Guardian	
Last Activity	

Recommendation:
 Ensure that when color differences are used to convey information, the information conveyed by the color differences are also conveyed explicitly in text.

6. WCAG 2.0 1.4.3 AA: Color Contrast

Applicable to:

Register Patron

Check In

Impact:


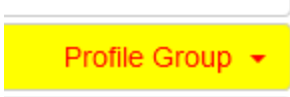

[High]

Explanation

Text, including images of text, should have sufficient color contrast with the background to make the text clearly visible to all users. When color contrast is low, people with low vision or color impairment are unable to see the text clearly. Target ratios are:

- Regular text a color contrast of 4.5:1 is the minimum ratio.
- Large text a color contrast of 3:1 is the minimum ratio.
- Visible focus color with no additional non-color indicator a color contrast of 3:1 between the normal view and hover or focus view is the minimum contrast. IS it recommended as a best practice to have a non-color indicator for links when they receive focus.

For more information, visit: [W3C - Color Contrast Minimum](#)

Contrast Ratio	Screenshot	Page
Contrast Ratio: 3.6:1		Register Patron
Contrast Ratio: 3.72:1		
Contrast Ratio: 3.1:1		Check In

7. WCAG 2.0 1.4.5 AA: Images of Text

Applicable to:

Home page

Impact:

[*Medium*]

Explanation:

Text should be used in place of images of text where the technology can present the visual presentation. Exceptions include logos. For more information visit [W3C - Images of Text](#)

Example: The alt text for the PINES logo includes the word logo which could add unnecessary redundancy for screen reader users.

Current Code:

```

```

Recommended Code:

```

```

8. WCAG 2.0 2.1.1 A: Keyboard Access

Applicable to:

Patron Page

Impact:

[*Medium*]

Explanation:

[Missing href on <a> element](#)

The href attribute is needed on <a> elements to ensure keyboard users can navigate to and activate the element.

Example: Several anchor elements are missing the href attributes.

Current Code:

```
<a ng-click="incrementHours()" ng-class="{disabled: noIncrementHours()}" class="btn btn-link disabled" ng-disabled="noIncrementHours()" tabindex="-1" disabled="disabled"><span class="glyphicon glyphico...</a>
```

Recommendation:

Those these links work and function intended for a standard keyboard, this may not be the case for other assistive technologies. Ensure that all elements of the site are accessible to all users.

9. WCAG 2.0 2.4.3 A: Focus Order

Applicable to:

Register Patron Page

Impact:

[Critical]

Explanation:

Keyboard focus should be visible and follow standard patterns: top to bottom, left to right. For more information, visit: [W3C-Focus Order](#)

Example: When a user completes the patron registration form, the next logical step is to submit or save their entry. Unfortunately, the focus moves from the last form field to the very top of the page and causes users to navigate through the various page elements before reaching the save/submit button.

Recommendation:

Provide a method for users to tab from the last form field to the Print, Save and Save & Clone buttons.

10. WCAG 2.0 2.4.4 A: Link Purpose in Context

Applicable to:

Register Patron

Impact:

[High]

Explanation:

Link text or link text with programmatically determinable context needs to provide the purpose or destination of the link. For more information, visit: [W3C-Link purpose in Context](#)

Example: The required fields, suggested fields, and all fields link text should be improved. The links act as filters for the content, but this could be lost on screen reader users who might assume that the links would take them to a different page that would explain or provide more information about required, suggested or all fields.

Current Code:

```
<a href="" ng-class="{disabled : edit_passthru.vis_level == 2}" ng-click="edit_passthru.vis_level=2">Required Fields</a>
```

Recommended Code:

```
<a href="" ng-class="{disabled : edit_passthru.vis_level == 2}" ng-click="edit_passthru.vis_level=2">Show Required Fields</a>
```

11. WCAG 2.0 2.4.7 AA: Focus Visible - Focus States & Indicators

Applicable to:

GLOBAL

Impact:

[Critical]

Explanation:

For anyone who relies on the keyboard to access web pages, visually determining the component or section of the page that has the focus of attention is essential. Users with attention limitations, short-term memory limitations, or limitations in executive processes also benefit by being able to recognize visually where the focus is located on a web page or application. For more information, visit: [W3C-Focus Visible](#)

Example: The navigation bar lack visible focus. The rest of the site has sufficient visible focus indicators, but it could be improved.

Recommendation:

See spreadsheet for recommendation.

12. WCAG 2.0 3.3.1 A: Error Identification

Applicable to:

Login

Impact:

[Critical]

Explanation: If an input error is automatically detected, the item that is in error is identified and the error is described to the user in text.

Example: The error message provided when a user enters incorrect login information is inadequate. The message is located beneath the login form which could cause it to be missed by keyboard only users or users who use screen magnification.

Also, if a user accidentally skips a form field, an error message is not provided to make them aware of this mistake. This could cause users to grow confused or assume that the submit button is broken.

Current Code:

```
<span ng-show="loginFailed" class="label label-warning">Login  
Failed</span>
```

Recommended Code:

Provide adequate error messages so that all users are made aware of any input mistakes that may be made.

See the following link for more information on accessible error messages: <https://www.w3.org/WAI/WCAG21/Understanding/error-identification.html>

13. WCAG 2.0 4.1.1 A: Validate HTML

Applicable to:

GLOBAL

Impact:

[Medium]

Explanation:

Inspect website for validation errors with non-standard HTML. For reference see: <https://validator.w3.org/>

Appendix A: Commonly Used Code Snippets

Links that open new windows, dialogs or PDFs

Give users advanced warning when before automatically opening a new window, dialog or PDF by providing a warning that allows the user to decide if they want to leave the current window, and the warning will help them find their way back if they do decide they would like to go to the new window. It will help them understand that the "back" button will not work and that they wish to return to the last window they had open, in order to find their previous location.

```
<a href="#">Webcredible (opens new window)</a>
```

Table 1: The name or label that describes a control can include the warning about opening in a new window.

[All about AMAC \(Opens new window\)](#)

```
<a href="http://www.amacusg.org/" target=" blank">All about AMAC
(Opens new window)</a>
```

Table 2: Using CSS to provide a warning before opening a new window

```
<html>
<head>
<title>Pop-Up Warning</title>
  <style type="text/css">
    body {
      margin-left:2em;
```

```
        margin-right:2em;
    }
    :focus { outline: 0; }
    a.info {
    position:relative;
    z-index:24;
    background-color:#ccc;
    color:#000;
    text-decoration:none
    }
    a.info:hover, a.info:focus, a.info:active
{
    z-index:25;
    background-color:#ff0
    }
    a.info span {
    position: absolute;
    left: -9000px;
    width: 0;
    overflow: hidden;
    }
    a.info:hover span, a.info:focus span,
a.info:active
    span {
    display:block;
    position:absolute;
    top:1em; left:1em; width:12em;
    border:1px solid #0cf;
    background-color:#cff;
    color:#000;
    text-align: center
    }
    div.example {
    margin-left: 5em;
    }
</style>
</head>
<body>
<h1>Pop-Up Warning</h1>
<p> This is an example of an <a
class="info"
    href="popup_advisory_technique.html"
target="_blank">
<strong>External link</strong><span>Opens
a new
    window</span></a>
```

```
        </p>
    </body>
</html>
```

A [working example of Using CSS to provide a warning before opening a new window](#) is available.

Hiding Content from both visual and screen reader users

In some cases, it is necessary to hide content from either visual users, or screen reader users. This would apply to content "expandable/collapsible" content, and other similar situations. To hide content from both visual and screen reader users, use the CSS instructions `display:none;` and `visibility:hidden;`

HTML and WAI-ARIA Landmarks

Defining all content within landmark regions is a good practice and provides a method for AT users to navigate quickly. To better understand landmarks in relation to HTML5 and WAI-ARIA, visit: [WAI-ARIA Overview](#). Follow [WAI-ARIA Authoring Practices for Landmark Regions](#) including providing an accessible name when defining multiple landmarks of the same type such as navigation. The best practice is to define Banner, Main and Contentinfo once per page.

New HTML 5 elements and corresponding ARIA roles:

HTML 5	WAI-ARIA Role
<header>	role="banner"
<nav>	role="navigation"
<main>	role="main"
<footer>	role="contentinfo"
<aside>	role="complementary"
none	role="search"
<form>	role="form"
<section>	role="region" provide accessible name using aria-labelledby or aria-label

The most useful of these for most websites are header/banner, nav/navigation, main, and footer/contentinfo. The others are useful as well, but are not as widely applicable across most websites.

Add Skip Navigation

Provide a method to skip navigation elements for keyboard users. Large header menus can be tedious and unnecessary when browsing multiple pages on the same website. For additional information, visit: [W3C-Adding a link at the top of each page that goes directly to the main content area](#)

Content for screen reader users but not sighted users

In some cases, extra content can be provided for the benefit of screen reader users, which is not useful for visual browsers. Examples include headings to identify parts of the page, where the visual design makes these divisions clear already. It is desirable to include such information without changing the visual layout of the site. However, most assistive technologies ignore content that is hidden using the technique given above. For content that should be announced to assistive technology users, but not appear visually, use the following CSS class:

CSS used to hide content visually, but make it available to assistive technology users

```
.visually-hidden
{
  position: absolute;
  clip: rect(1px 1px 1px 1px); /* for Internet Explorer */
  clip: rect(1px, 1px, 1px, 1px);
  padding: 0;
  border: 0;
  height: 1px;
  width: 1px;
  overflow: hidden;
}
```

HTML demonstrating use of the class

```
<p class="visually-hidden">This paragraph is present in the DOM
and accessible to assistive technologies, but is visually
hidden.</p>
```

OPTGROUP to group OPTION elements inside a SELECT

Group information in a select dropdown using optgroup. For more information, visit: [W3C Optgroup](#)

Tab Panel

WAI-ARIA-enabled tabbed interface has three components: the `tablist`, which contains a collection of tabs, each of which is a control for loading its associated content tabpanel. Depending on the size of the window and length of tab labels, you can generally create a tab view that contains between two and eight tabs. `Tablist` strips should never wrap to multiple rows or insert horizontal scrollers for the tabs because of the poor visual connection between the tab and the pane visually displayed below. Additionally, tooltips should not be used directly on the tabs as this causes screen readers to enter Browse Mode when Application Mode is needed to navigate and announce a tab's state and properties.

If a `tabpanel` does not contain regular text content, there is nothing to set focus to using the Tab key and exits the tab component. If the regular text content includes

one or more links, then pressing the Tab key will set focus to the first link, skipping over any text content that might come before it. Unless the screen reader uses Browse Mode with the Virtual Cursor on, it remains in Application Mode, and cannot read any regular text content using the normal reading commands. If the `tabpanel` had focus, the user could manually turn the Virtual Cursor back on.

The `tabpanel` itself is set with `tabindex="0"` to place it in the Tab order. In this case, screen readers automatically exit Application Mode and enter Browse Mode in both when using the Tab key to move focus to the `tabpanel`. This, then, allows these screen readers easy access to the `tabpanel` content through the normal reading commands. It is worth noting that setting focus to a non-form or non-application element can accomplish the same thing in some instances so this suggests a possible workaround, namely including such an element (for example, a heading) in the Tab order at the beginning of the `tabpanel` content. The main heading in each `tabpanel` has `tabindex="0"` to place it in the Tab order. This allows the use of the Tab key to move focus from the active tab control into the `tabpanel`'s regular text content. This causes screen readers to automatically exit Application Mode and permit the use of the screen reader's normal reading commands for accessing the `tabpanel` content.

1. Add the role of `tablist` to the `ul` element, indicating that the children are tabs.
2. Add the role `presentation` to each of the `li` elements, indicating that the screen reader should ignore the list items themselves.
3. Add role of `tab` to each link, re-mapping their roles to the intended screen-reader recognizable element type.
4. Add `aria-selected` to each of the tabs. When you switch tabs in your JS code, update these to reflect the new state of each. Only one may be selectable at any given time, so the values of two should be false, and only one should be true.
5. Add `aria-controls` to each tab, indicating which panel the tab references.
6. Add a role of `tabpanel` to each of the div containers and include `tabindex="0"`.
7. Add `aria-labelledby` referencing the actual tab's name given to the anchor elements by the inner text above as labels for the panels.

```
<div class="tabs3 tabst1">
<ul>
<li role="presentation" tabindex="-1" aria-selected="false" aria-
controls="tabs-1">
<a href="#tabs-1" role="tab" id="ui-id-8">
Double Header <br />
Section 1</a></li>
<li role="presentation" tabindex="-1" aria-selected="false" aria-
controls="tabs-2">
<a href="#tabs-2" role="tab" tabindex="-1" id="ui-id-9">
Double Header <br />
Section 2</a></li>
```

```
<li role="presentation" tabindex="-1" aria-selected="true" aria-
controls="tabs-3"><a href="#tabs-3" role="tab" tabindex="-1" id="ui-
id-10">
Double Header <br />
Section 3</a></li>
</ul>
<div id="tabs-1" tabindex="0" role="tabpanel" aria-labelledby="ui-id-
8">
<p>Content for Tab 1. </p>
<p>This is the content area of tabpanel 1. </p>
</div><div id="tabs-2" tabindex="0" role="tabpanel" aria-
labelledby="ui-id-9"><p>Content for Tab Two.</p><p>This is the content
area of tabpanel 2. </p></div><div id="tabs-3" tabindex="0"
role="tabpanel" aria-labelledby="ui-id-10"><p>Content for Tab
Three.</p><p>This is the content area of tabpanel 3. .
</p></div></div>
```

Lastly, each tabpanel needs to be addressed by adding `id="ui-id-8"`, `role="tabpanel"`, `tabindex="0"` and `aria-labelledby` or use `aria-label` to call the link text of the tablist above.

There are 3-possibilities to set the initially selected tab:

1. active option (highest priority)
2. url fragment identifier matching a tab `href`'s fragment identifier:
`href="#fragment-identifier"`
3. The `ui-tabs-active` class attribute is already specified in HTML source:
`<li class="ui-tabs-active">` (lowest priority)

It is possible to initialize an empty tab set and add tabs thereafter.

1. After collapsing the active tab, it cannot be activated a second time unless the collapsible option is set to true.
2. The type of tab (in-page vs. Ajax) is determined automatically from the `href` attribute of the contained anchor elements.
3. Restructuring of the ``'s `listitem(s)` is required.
 - a) Add the role `presentation` to each of the `li` elements, indicating that the screen reader should ignore the list items themselves.
 - b) Move role of `tab` to each link, re-mapping their roles to the anchor as the intended screen-reader recognizable element type.
 - c) Move both `aria-selected` and `aria-controls` from each of the ``'s re-mapping these properties to the anchor.

Note: It is important to remove all `tabindex` attributes with a value higher than "0".

Example:

```
<ul id="tabs" role="tablist">
  <li role="presentation" tabindex="-1"><a id="tab1" href="#"
  onclick="showTab(1);" role="tab" aria-controls="panel1" aria-
  selected="true">Tab 1</a></li>
  <li role="presentation" tabindex="-1"><a id="tab2" href="#"
  onclick="showTab(2);" role="tab" aria-controls="panel2" aria-
  selected="false">Tab 2</a></li>
  <li role="presentation" tabindex="-1"><a id="tab3" href="#"
  onclick="showTab(3);" role="tab" aria-controls="panel3" aria-
  selected="false">Tab 3</a></li>
</ul>
...
<div id="panel1" role="tabpanel" aria-labelledby="tab1">
  ...
</div>
<div id="panel2" role="tabpanel" aria-labelledby="tab2">
  ...
</div>
<div id="panel3" role="tabpanel" aria-labelledby="tab3">
  ...
</div>
```

Keyboard

When focus is on a tab:

- UP/LEFT - Move focus to the previous tab. If on first tab, moves focus to last tab. Activate focused tab after a short delay.
- DOWN/RIGHT - Move focus to the next tab. If on last tab, moves focus to first tab. Activate focused tab after a short delay.
- HOME - Move focus to the first tab. Activate focused tab after a short delay.
- END - Move focus to the last tab. Activate focused tab after a short delay.
- SPACE - Activate panel associated with focused tab.
- ENTER - Activate or toggle panel associated with focused tab.
- ALT+PAGE UP - Move focus to the previous tab and immediately activate.
- ALT+PAGE DOWN - Move focus to the next tab and immediately activate.

When focus is in a panel:

- CTRL+UP - Move focus to associated tab.
- ALT+PAGE UP - Move focus to the previous tab and immediately activate.
- ALT+PAGE DOWN - Move focus to the next tab and immediately activate.

Accessibility

Follows WAI-ARIA best practices with the following exceptions:

- Arrow keys do not immediately activate panels. This allows users to navigate tabs without activating them, which may be important for remote tabs.
- Updated the `aria-selected` state immediately so that assistive technology solutions announce the tab as `selected`, even though there is a delay in activating it.
- We implement `ALT+PAGE UP/DOWN` instead of `CTRL+PAGE UP/DOWN` because of interference with browser shortcuts, as mentioned in the best practices.
- Also works when a tab has focus. This allows consistent navigation regardless of whether focus is on a `tab` or in a `tabpanel`.
- Tabs and panels have `aria-labelledby` pointing to the main anchor in the tab. This allows additional controls, such as buttons, added to the tab without affecting the label. The main tab has a role of `presentation`.
- Ajax tabs should have `aria-live=polite` on the content panel. When the activated tab's the content is loading `aria-busy` is set to `true`; after the content loads, then we'll remove `aria-busy`.
- Add `aria-controls` on the tab's anchor pointing at the associated panels. This allows AT to announce the relationship. For example, JAWS implements a feature where you can jump from the tab to the panel because of the `aria-controls` relationship.
- As the user navigates with the arrow keys when the tab first gains focus, since the panel remains hidden this is not announced because we use delayed activation.

Navigating through the tabs should not immediately activate the panels; there needs to be a short delay so that users can navigate through the tabs without activating each tab along the way.

Navigating while holding `CTRL`, should prevent the automatic activation. This makes it possible for users of assistive technologies such as screen readers, to have as much time as they need while navigating without activating each tab.

`ALT+PAGE UP` and `ALT+PAGE DOWN` will move focus to the previous or next tab, respectively, and immediately activate the tab (this works regardless of whether focus is on the tabs or in a panel). When focus is in a tab panel, `CTRL+UP` will move focus to the associated tab.

Tables

There are two distinct types of tables frequently used on web pages: data tables and layout tables. Use data tables when row and column headers provide contextual information within the table. In the past, we commonly used, layout tables to overcome limitations in visual presentation and layout using HTML. With the more common use of cascading style sheets (CSS) for general layout purposes, layout tables

are less frequently used. For a user who visually accesses a web page, most of the time it will be irrelevant whether a layout table or CSS is being used or not. Most screen reading programs, however, will indicate whether a table exists, along with the exact number of columns and rows of the layout table. For users who are blind, using too many layout tables for information can cause confusion and information overload, not to mention weighting down your web pages significantly. Using CSS accomplishes this just as easily. For more information, visit: [Info and Relationships: USC 1.3.1](#)

The WCAG 2.0 Guideline 1.3.1 for Info and Relationships recommends:

- Tables used for tabular data.
- Headings used to associate data cells with headers.
- Data table captions and summaries used where appropriate.

In HTML, this means that a properly coded, accessible table containing tabular data would look similar to the following table:

```
<table summary="Phone Contact List">
  <thead>
    <tr>
      <th>Name</th>
      <th>Phone</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>John Doe</td>
      <td>404-555-1212</td>
    </tr>
    <tr>
      <td>Jane Zoe</td>
      <td>404-555-2121</td>
    </tr>
  </tbody>
</table>
```

The `<thead>` tag contains the header row that defines the columns for the table; these column labels are contained within the `<th>` tag. The data for the table is contained within the `<tbody>` tag and the cells contained with the `<td>` tag. The summary attribute for the `<table>` tag describes the content or purpose of the table. The presence of a `<thead>` tag with column labels contained within `<th>` tags aids a screen reader user in table orientation.

Data Tables

Tables used to display data in a grid. To make tables accessible, header cells must be marked up with `<th>`, and data cells with `<td>`. Explicit associations needed for more complex tables, use `scope`, `id`, and `headers` attributes. For more information, visit: [Info and Relationships: USC 1.3.1](#)

Layout Tables

Using tables to dictate the layout can keep pages from rendering and re-flowing properly. This is particularly true on pages, which use very complex layouts.

Whenever possible, developers should utilize stylesheet markup to define the layout of the page instead of tables.

- Ensure absolute sizing in tables is avoided
- Ensure layout tables do not contain structural markup
- Ensure layout tables indicate their use for presentation purposes

Ensure layout tables linearize properly

Add WAI-ARIA `role="presentation"` to prevent screen readers from announcing table structure.

Scalable Font and Element Widths (Sizes)

Affects: All visual users, especially screen magnification, mobile, & other viewports.

EM and REM

Traditionally web design has used Pixel (px) sizes for font-sizes and layout purposes. While this worked well when computer monitors were very standard in sizes and resolutions, this is no longer the case. Relative units are a commonly used method to display scalable font on multiple viewport sizes. EMs scale according to the font-size of their parent elements, REMs (root EM) scale according to the root (body) element font-size.

Approaches & Responsive Design

There are many approaches to properly addressing issues related to layout and sizing. Ultimately the most robust method is a full responsive design approach with media queries for viewport sizes. This, however, is the most time-consuming method and would only be appropriate for a redesign, not a remediation. There are a few other approaches that are effective for improving font and layout resizing.

Setting <body> font-size to =”62.5%”

Setting the font-size to 62.5% in the body element is a common and easy approach to modifying pixel widths. By setting the font-size in the body element, this attribute is then inherited throughout every other element. Changing the font-size to this value makes the math of converting px to em/rem very easy. 14px at 100% == 1.4 em/rem at 62.5%. This is a very human readable method for converting sizes. An effective combination is to use EMs for “margin, padding, width, height, and line-height” and REM for most other PX values, specifically “font-size.” There are other approaches, such as setting the body font-size in the body to a pixel value then scaling based on that, that may work better for your purposes. For more information, visit: [Font size: the font size property](#)

Using button elements without the default formatting

Button elements are often avoided as often their default appearance does not meet design goals. However, `button` elements are what we call "natively active elements". This means that you can use `onClick` with these elements and keyboard access will work automatically. If instead you start with, say, a `div` element, additional coding is needed to implement keyboard access.

The CSS below shows how to remove aspects of default browser formatting for buttons. The comments explain what each line does, so that you can choose just what you need for your design.

CSS used to clear default formatting

```
button.clearformatting {  
  
    /* remove browser-provided 3d edge effect */  
  
    border: 0;  
  
    /* remove extra spacing, and animated effect */  
  
    padding: 0;  
  
    /* remove color assigned by browser */  
  
    background: transparent;  
  
    /* add any formatting you like */  
  
    background: url('button_texture.gif') repeat  
  
}
```

Hiding and showing content with JavaScript DOM Coding

When changing the page using JavaScript, a common mistake is to add a newly created element to the end of the document object model (to the end of the Web page) even if that is not a logical location for the new element. Often, then CSS is used to move the item to a logical location visually. However, even with the CSS, to the screen reader user, that element is still at the bottom of the page.

This is code that allows you to place your new element anywhere within the Document Object Model, so that you can choose the best location for all users.

Placing a new element before another element

In this example, a new paragraph is created and is placed just before an ordered list with id L:

```
/* Create paragraph */  
  
var newP = document.createElement("p");  
  
var PText = document.createTextNode("This list will help with the  
questions you missed!");  
  
newP.appendChild(PText);  
  
/* Get existing ordered list */  
  
var existingL = document.getElementById("L");  
  
/* Get parent of existing ordered list */  
  
var parentObj = existingL.parentNode  
  
/* Place new paragraph inside the parent of the ordered list,  
just before the ordered list */  
  
parentObj.insertBefore(newP, existingL);
```

Placing a new element after another element

JavaScript doesn't offer a command for insert after, but there's a reliable way to do this. In this example, a new paragraph is created and is placed just after an ordered list with id L:

```
/* Create paragraph */  
  
var newP = document.createElement("p");  
  
var PText = document.createTextNode("Good job with the list; now  
take a break!");  
  
newP.appendChild(PText);  
  
/* Get existing ordered list */  
  
var existingL = document.getElementById("L");  
  
/* Get parent of existing ordered list */  
  
var parentObj = existingL.parentNode
```

```
/* Get the next sibling of the existing ordered list */  
  
var nextS = existingL.nextSibling  
  
/* don't worry if there's a chance nextS will be null - see below  
*/  
  
/* Place new paragraph inside the parent of the ordered list,  
just after the ordered list */  
  
parentObj.insertBefore(newP, nextS);  
  
/* if nextS is null, insertBefore will place newP at the end of  
parentObj,  
which would be just after the existing ordered list, in that case.  
*/  
  
/* Create paragraph */  
  
var newP = document.createElement("p");  
  
newP.appendChild(document.createTextNode("Good job with the list;  
now take a break!"));  
  
/* Get existing ordered list */  
  
var existingL = document.getElementById("L");  
  
/* Add to page after list */  
  
existingL.parentNode.insertBefore(newP,existingL.nextSibling);
```

Making a new element the first child of another element

In this example, a new list item is created and becomes the first list item in the ordered list with id L:

```
/* Create new list item */  
var newLi = document.createElement("li");  
var PText = document.createTextNode("Start by making  
sure you have a good place to study.");  
newLi.appendChild(PText);  
/* Get existing ordered list */  
var existingL = document.getElementById("L");  
/* Place new list item inside the ordered list,  
just before the first child of the ordered list */  
existingL.insertBefore(newLi,existingL.firstChild);
```

Making a new element the last child of another element

In this example, a new list item is created and becomes the last list item in the ordered list with id L:

```
/* Create new list item */
var newLi = document.createElement("li");
var PText = document.createTextNode("Double check
your work.");
newLi.appendChild(PText);
/* Get existing ordered list */
var existingL = document.getElementById("L");
/* Place new list item inside the ordered list,
as the last item in the ordered list */
existingL.appendChild(newLi);
```

ARIA, Dynamic Content, alerts, and feedback

Provide user feedback using ARIA for states and live regions on webpages. For more information, visit: [Live Region Attributes](#) and [aria-busy \(state\)](#)

ARIA and Dynamic Content

WAI-ARIA provides a framework for adding attributes to identify features for user interaction, how they relate to each other, and their current state. WAI-ARIA describes new navigation techniques to mark regions and common Web structures as menus, primary content, secondary content, banner information, and other types of Web structures. Follow [WAI-ARIA Authoring Practices](#).

- Avoid forced focus changes that are not user-initiated
- Ensure auto-updating dynamic content can be paused, stopped, or hidden
- Ensure content updates define focus updates appropriately
- Ensure that dynamic content is rendered in-line with the controls that change it
- Ensure that textual equivalent information is updated appropriately when an element's state changes
- Inform assistive technologies of changes in content
- Provide an accessible alert method for content changes that occur without explicit user knowledge

WAI-ARIA

Follow [WAI-ARIA Authoring Practices](#).

- Ensure appropriate use of ARIA roles, states, and properties are provided
- Ensure ARIA regions, landmarks and HTML sections are identifiable
- Ensure ARIA roles, states, and properties are valid

- Ensure elements that use ARIA provide non-ARIA fallback accessible content when not accessibility supported

Live Regions

- Ensure live regions define atomic-ness unless they are not atomic
- Ensure live regions define controlling elements when present
- Ensure non-uniformly updated live regions use the ARIA busy attribute
- Ensure relevant changes for live regions are explicitly defined if the change is not text or a node addition
- Ensure live regions for dynamically changing content are provided
- Provide explicit labels for live regions
- Ensure long descriptions for complex live regions are provided

ARIA Menus

When implementing WAI-ARIA menus and sub-menus be sure to follow the [W3C best practices for menus](#). Another resource for menus is the [W3C Menu Tutorial](#). The Stack Overflow Flyout Menu Summary is a concise outline of the requirements for flyout/sub-menus. Follow [WAI-ARIA Authoring Practices](#) for Menu, Menu bar and Menu button.

Stack Overflow Flyout Menu Summary

HTML structure:

```
<div> <!-- Outer wrapper -->
  <ul> <!-- Main navigation bar container -->
    <li> <!-- First-level item without submenu -->
      <a> <!-- Destination URL -->
      </a>
    </li>
    <li> <!-- First-level item with submenu -->
      <a> <!-- Destination URL -->
      </a>
      <ul> <!-- Second-level menu container -->
        <li> <!-- Second-level item -->
          <a>
          </a> <!-- Destination URL -->
        </li>
      </ul>
    </li>
  </ul>
</div>
```

Roles:

- role="navigation" for outer wrapper <div>
- role="menubar" for navigation bar container
- role="menu" for second-level containers
- role="presentation" for first- and second-level menu items (they are not needed in the exposed accessible menubar structure)

- role="menuitem" for first- and second-level <a> menu items

Properties:

- aria-haspopup="true" for first-level <a> menu items having a submenu
- aria-labelledby="ID of previous <a> menu item" for second-level containers

States:

- aria-selected="true" on currently visited first- or second-level <a> item; aria-selected="false" on the other <a> items. That is to enforce the concept "selected <==> current page"
- aria-expanded="true/false" for second-level containers
- aria-hidden="true/false" for second-level containers
- aria-activedescendant="" for main navigation bar container. This is an alternative to working with tabindex
- tabindex=0 on currently visited <a> item; tabindex=-1 on the other <a> items. That is in order to first focus on the current page when tabbing to the navigation bar. It is an alternative to working with aria-activedescendant

Keyboard:

1. Tab: Move focus in/out of the menu from other points in the web application.
2. Shift+Tab: Move focus in/out of the menu from other points in the web application, in the reversed order.
3. Right arrow: Next navigation bar item
4. Left arrow: Previous navigation bar item
5. Enter: Activate currently focused item (i.e. navigate to corresponding URL)
6. Space: Activate currently focused item (i.e. navigate to corresponding URL)

Appendix B: Screenshots

URLs include the following:

1. Login - <https://terran-testbox.gapines.org/eg/staff/login>

Sign In

Username

Password

Workstation

2. Home - <https://terran-testbox.gapines.org/eg/staff/>

Search

APINES

Circulation and Patrons

- Check Out Items
- Check In Items
- Register New Patron
- Search For Patron By Name
- Pull List for Hold Requests

Item Search and Cataloging

- Advanced Search
- Item Status / Display
- MARC Batch Import / Export
- z39.50 Import
- Record Buckets
- Item Buckets

Administration and Documentation

- GPLS Help Desk
- PINES Documentation
- Reports
- PINES Quick Reports
- Workstation Administration

3. Patron Search - <https://terran-testbox.gapines.org/eg/staff/circ/patron/search>

4. Patron Book Check Out - <https://terran-testbox.gapines.org/eg/staff/circ/patron/87/checkout?card=99999303411>

5. Patron Book Holds - <https://terran-testbox.gapines.org/eg/staff/circ/patron/87/holds>

6. Register patron - <https://terran-testbox.gapines.org/eg/staff/circ/patron/register>

7. Book check in - <https://terran-testbox.gapines.org/eg/staff/circ/checkin/checkin>

The screenshot shows a library system interface with a green navigation bar at the top containing 'Search', 'Circulation', 'Cataloging', 'Acquisitions', and 'Administration'. The main area is titled 'Checkin Items' and includes a search bar with a 'Barcode' field and a 'Submit' button. An 'Effective Date' field is set to '2019-06-24'. Below this is a table titled 'Items Checked In' with columns: #, Balance Owed, Barcode, Bill #, Checkin Date, Family Name, Finish, Location, Route To, Start, Title, Circulation Modifier, and Circulation Library. The table contains one row with the following data: # 1, Balance Owed (empty), Barcode FIC610001612, Bill # (empty), Checkin Date (empty), Family Name (empty), Finish (empty), Location Display, Route To Hoks Shelf, Start (empty), Title Coraline, Circulation Modifier (empty), and Circulation Library BR3. Below the table are buttons for 'Print Receipt', 'Trim List (20 Rows)', 'Strict Barcode', and 'Checkin Modifiers'.

Appendix C: Useful Resources

W3C’s Web Accessibility Initiative (WAI)

- [Web Content Accessibility Guidelines 2.0 \(WCAG\)](#)
- [How to Meet WCAG 2.0 \(Quick Reference\)](#)
- [Accessible Rich Internet Applications \(WAI-ARIA\) Suite Overview](#)
- [WAI-ARIA 1.0 Authoring Practices](#)
- [WAI-ARIA 1.0 | The Roles Model](#)

U.S. Federal Government

[Section 508 Refresh Guidelines](#)

PDF

[Adobe’s Accessibility Resource Center for PDFs and Flash](#)

Adobe TV | [Accessibility @ Adobe tutorials, demos and techniques](#)

Flash

Adobe | [Flash Accessibility site](#)

Company Information



512 Means Street NW
Suite 250
Atlanta, Georgia 30318

Phone: 404-894-8000
Toll-Free: 866-279-2964
Fax: 404-894-8323

Customer Support Hours:
8:30am - 4:30pm EST

For customer support or technical assistance, please call 1-866-418-2750 or send an email to cidi-support@design.gatech.edu.

Reviewers

The following AMAC staff members carried out the Georgia PINES web review:

Rayianna Daniels, Digital Accessibility Specialist, has a BS of Science in Computer Science, along with a background in web accessibility training, web accessibility evaluations, web development and IT support. She also provides training on a variety of applications and assistive technology solutions to Higher Ed institutions across the country.

John Rempel, QA Accessibility Specialist, has more than 20 years of training experience, with certifications and extensive experience as an AT Specialist, Vision Rehabilitation Therapist, Orientation & Mobility Specialist and Certified Professional in Accessibility Core Competencies (CPACC) through IAAP. Due to his own visual impairment, he relies on some of the same AT solutions and techniques used for testing web accessibility. For additional information, [read John Rempel's bio](#).