

Developing Accessibility Compliant Surveys in Blaise 5: Tools, Techniques, Challenges and Best Practices

Michael Bunitsky and Todd Flannery, Westat

Presenter: Mangal Subramanian

1. Introduction

To ensure that the content of web surveys is accessible to people who use Assistive Technologies (AT) to navigate the internet, we must customize controls in Blaise 5 to meet the standards of Section 508 of the Rehabilitation Act of 1973¹ and WCAG2.1². While Blaise software provides several tools to aid in meeting these standards, Blaise coders often need to customize templates or other controls to ensure compliance across various user devices and settings. This paper will describe the challenges and tools involved in making instruments accessible to AT users, along with several approaches that allow Westat to meet these requirements.

2. Some Initial Considerations

Although Blaise coders do not have access to all the screen reader devices that may be available to users who require AT, there are several options available to be able to meet the accessibility requirements. Blaise offers settings like disabling auto focus and use of skip links to allow greater options as we build instruments and will send warnings to us when exceptions are noticed via validating the layouts and allowing visibility of accessibility errors. To be able to fully satisfy the demands of compliance we may use tools and techniques that can be customized to specifications.

3. Tools

Our 508 testing and development teams rely on a variety of tools to verify that our instruments meet accessibility requirements.

Our standard screen reader and accessibility development tool is the free and open-source Non-Visual Desktop Access (NVDA)³ application. In addition to audibly reading the text as any screen reader would, of particular use to our developers is NVDA's tool for visually displaying the screen reader output as text, rather than playing it as audio. [Fig.1]

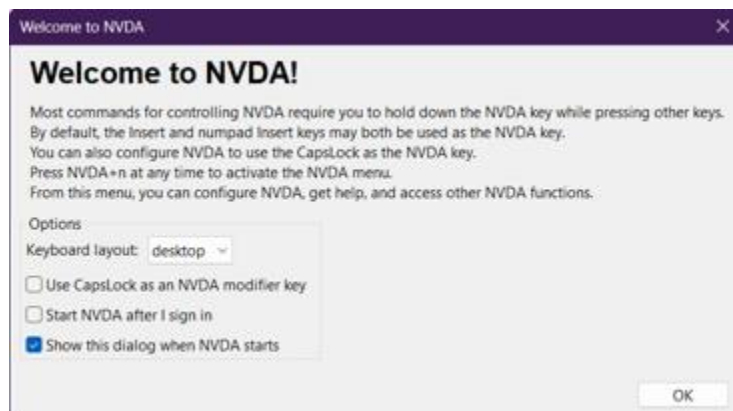


Figure 1

We also use TPGi's free Colour Contrast Analyzer (CCA)⁴ application to verify that our layout design colors meet accessibility requirements for low-vision and color-blind respondents. [Fig.2]



Figure 2

But by far our most valuable tool for 508 development and testing, we use the Accessible Name and Description Inspector (ANDI)⁵ “bookmarklet” hosted by the U.S. Social Security Administration website to access a wide variety of valuable information right in our web browsers, as we are running our instruments in real-time. ANDI provides data about behind-the-scenes values used by screen readers for every focusable element on the page; information about graphics and color contrast; accessibility data on

links and table structures; and any hidden content on the page that might be visible to screen readers. It also looks for errors and other accessibility issues with the page and will point them out, making it a bit easier to understand what works and what doesn't. [Fig.3]

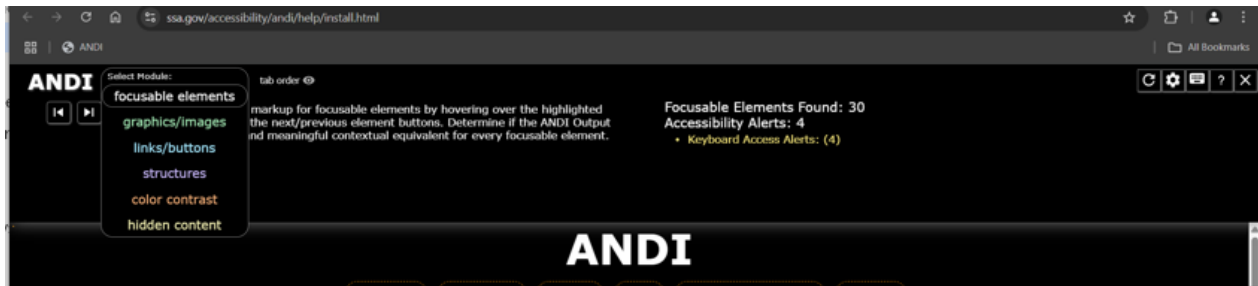


Figure 3

It's important to note that ANDI requires an update to the Blaise Content Security Policy file (.bcsp) to work with Blaise web instruments. Add this line to bcsp file to use ANDI with a Blaise web instrument. [Fig.4]

```
<ContentSecurityPolicySource SourceURI="www.ssa.gov/accessibility/andi/" UseForScripts="true" UseForStyles="true" UseForImages="true" UseForConnect="false" UseForDefault="false" />
```

Figure 4

4. Techniques

To assure accessibility compliance we often need to customize controls and apply specialized techniques. Here are a couple examples of these techniques that have proven useful in our development.

4.1 OS textbox screen reader text: working with fills and building the correct string

In an OtherSpecify group with a main enumerated/set field and a SPECIFY string field, using the standard ScreenReaderText expression for the StringTextBox template [Fig.5], we discovered that any fills included in the string field's question text would be empty when first read by the screen reader, until such time as the SPECIFY textbox received the focus on-screen, at which point the fill text would suddenly appear. Our testing indicated this was because when first landing on the OS group page, the main enumerated/set field receives the initial focus, and the SPECIFY string field hasn't yet been reached on the route, meaning its fills haven't been loaded yet, either. (This is also true for OS Category texts from the main field that include fills, as they take their question text from the string field in the OS group, rather than the enumerated/set field.)

```

IF Field.RoleTextExists('Question') THEN
    IF Field.RoleTextExists('Watermark') THEN
        Field.GetRoleText('Question') + ' ' + Field.GetRoleText('Watermark')
    ELSE
        Field.GetRoleText('Question') + ' <text key=txtAnswerStringInfo>'
    ENDIF
ELSE
    IF Field.RoleTextExists('Watermark') THEN
        Field.Name + ' ' + Field.GetRoleText('Watermark')
    ELSE
        Field.Name + ' <text key=txtAnswerStringInfo>'
    ENDIF
ENDIF

```

Figure 5 – Standard StringTextBox ScreenReaderText expression (from VisuallyImpaired sample)

Our solution to this problem was to update the ScreenReaderText expression for the StringTextBox template, to create a new string only when the text box is being used as a SPECIFY field in an OS group. [Fig.6 - bold] We identify these instances by checking the datatype of the first field on the page; in the case of our OS groups, this will always be an enumerated or set field. We then use the Question roletext from the first field on the page, rather than the Question roletext associated with the string field, because we know any fills in the main enumerated/set field will have already been loaded.

However, in our survey the two question texts in our OS groups differ from one another in a few key ways. For instance, any “Choose one response” or “Choose all correct answers” text that might be included in the main enumerated/set field’s question text, would not be included in the SPECIFY string field’s question text. Similarly, the string field’s question text included additional instruction text required for 508 compliance, along with further text added by project methodologists. To satisfy all of these competing requirements, we constructed our SPECIFY StringTextBox ScreenReaderText expression by first taking a substring of the main enumerated/set field’s question text, starting from the beginning of the string, and going forward until the first appearance of a question mark [Fig.6 - yellow]; this returns the main question text stem, while ignoring any additional “Choose one response” text or similar, that wouldn’t apply to the SPECIFY field.

To that string we then added the Category text from the OS category of the main enumerated/set field [Fig.6 - aqua], identified as the first category that includes the word “SPECIFY.” Including this text is a 508 requirement.

Finally, for the additional and instructional texts required by 508 and/or project methodologists, we created Text controls in the resource database and added them to our expression string. [Fig.6 – green, grey]

BOLD is our updates to the standard expression above;

YELLOW grabs a substring of the question text, from the beginning of the text to the first question mark; it uses the Question roletext of the first field on the page (the main enumerated field for the OS group) rather than the field associated with the StringTextBox, so that any fills in the question text will appear in the screen reader text, as well;

AQUA adds the text of the first Category that contains the word 'specify';

GREEN adds the project-specified OS text;

GREY adds the 508 required OS text.

```
IF Field.RoleTextExists('Question') THEN
  IF Field.RoleTextExists('Watermark') THEN
    Field.GetRoleText('Question') + ' ' + Field.GetRoleText('Watermark')
  ELSE
    IF Page.Fields.First().DataType = Enumeration OR Page.Fields.First().DataType = Set THEN
      SUBSTRING(Page.Fields.First().GetRoleText('Question'), 1, POSITION('?', Page.Fields.First().GetRoleText('Question'))
      + ' '
      + Page.Fields.First().Categories.First(POSITION('SPECIFY', item.text) <> 0).Text
      + ' <text key=txtOS_Project>'
      + ' <text key=txtOS_508>'
    ELSE
      Field.GetRoleText('Question') + ' <text key=txtAnswerStringInfo>'
    ENDIF
  ENDIF
ELSE
  IF Field.RoleTextExists('Watermark') THEN
    Field.Name + ' ' + Field.GetRoleText('Watermark')
  ELSE
    Field.Name + ' <text key=txtAnswerStringInfo>'
  ENDIF
ENDIF
```

Figure 6 – Updated StringTextBox ScreenReaderText expression for OS fields

4.2 Using roletexts to limit repetition of long field texts for first category

For the screen reader text of Categories in an Enumerated list, it is considered best practice to repeat the question text before the first category text. [Fig.7] In some cases, however, this can present an impediment to the AT-user.

```

IF Category.code = 1 THEN
    Field.GetRoleText('Question') + '' + Category.GetRoleText('Category')
ELSE
    Category.GetRoleText('category')
ENDIF

```

Figure 7 - Standard Enumeration CategoryRadioButton ScreenReaderText expression (from VisuallyImpaired sample)

Consider the example of a survey consent page, that could have several paragraphs of introductory and explanatory text, before a simple Yes/No question. After the screen reader application reads out the question text on-screen, having to listen to it be read out all over again a second time before the “Yes” category is finally read would be frustrating and time-consuming.

A more common example might be the first in a series of pages of enumerated questions, where the question text for the first field includes an introductory paragraph explaining the series, before finally asking the first question. In this case, it wouldn’t be useful for the screen reader application to read the introductory paragraph a second time when it reads out the first category. Instead, we would only want to include the actual question part of the text and omit the opening paragraph.

Our solution to this problem was to add a roletext to fields with over-sized or introductory texts. The roletext is initialized to the limited substring of the field text that should be repeated for the first category. Finally, in our ScreenReaderText expressions, we simply look for the existence of that roletext in the field definition, or else we use the standard question text. [Fig.8]

[BOLD = our update to the standard expression above]

```

IF Category.code = 1 THEN
    IF Field.RoleTextExists('QTXTSub') THEN
        Field.GetRoleText('QTXTSub') + '' + Category.GetRoleText('Category')
    ELSE
        Field.GetRoleText('Question') + '' + Category.GetRoleText('Category')
    ENDIF
ELSE
    Category.GetRoleText('category')
ENDIF

```

Figure 8 – Using a roletext to avoid repetition of long field texts by screen readers

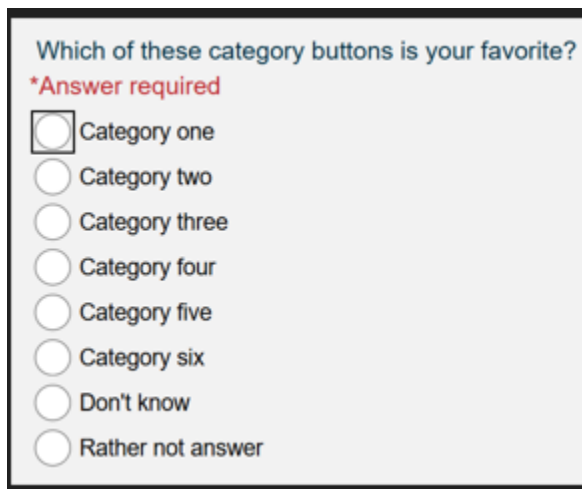
5. Challenges

Development of customized solutions for accessibility controls sometimes presents challenges in terms of methodology or limitations of the existing tools. Sometimes compromise or creativity can produce the desired results. Here we describe some examples of these challenges.

5.1 The Focus Problem

One of the requirements for any 508 accessible instrument is to visually indicate the current focus on-screen at any given time, so that keyboard-only users can know which control on the page they are interacting with. The standard method for accomplishing this is to wrap the control with the focus in a 2pt-width, solid black outline. Methodologists voice concerns that this could potentially cause data bias in certain circumstances.

When a respondent tries to move forward on the route before responding to the current question, the error state is triggered, and an error message appears on-screen indicating that the question must be answered before moving on. In the context of that error message being on the screen, having the first category highlighted with a thick, black border could possibly be interpreted as indicating that the highlighted response is the correct response, or the response that the instrument is asking for, leading to a higher prevalence of that response being chosen by respondents. [Fig.9]



Which of these category buttons is your favorite?
*Answer required

- Category one
- Category two
- Category three
- Category four
- Category five
- Category six
- Don't know
- Rather not answer

The image shows a survey question with a list of radio button options. The first option, 'Category one', is selected and has a thick, solid black border around its radio button. The other options are 'Category two', 'Category three', 'Category four', 'Category five', 'Category six', 'Don't know', and 'Rather not answer'. The question text is 'Which of these category buttons is your favorite?' and there is a red asterisk indicating that an answer is required.

Figure 9

To resolve this conflict between data integrity and instrument accessibility, we made some key changes to our focus and error-handling in our resource database. The first was to soften the appearance of the control outlines by changing them from 2pt-width, solid black to 1pt-width, dashed black.

We then added a panel around the DataValueArea on the FieldPane template, and added expressions to the BorderWidth and Color properties, to show a 2pt-width solid red border around the DataValueArea whenever the correct error state was thrown.

As a result of these changes, when a respondent tries to move forward on the route without answering, the entire category list is now surrounded by a red outline, in the same color red as the error message, to indicate that all categories are available, and no one category is being singled out; thereby satisfying the methodological requirements. At the same time, the focus around the control is still indicated by the dashed outline, satisfying 508 requirements. [Fig.10]

Which of these category buttons is your favorite?
***Answer required**

- Category one
- Category two
- Category three
- Category four
- Category five
- Category six
- Don't know
- Rather not answer

Figure 10

5.2 Lookup Textboxes vs. Screen Readers

It's very important to set the Accessibility Role property correctly for 508-compliant controls. The Accessibility Role is used to set values that tell the screen readers how to interpret and describe the controls to AT-users. If the Accessibility Role is set incorrectly, then respondents who can't see the screen won't be able to understand how to interact with the instrument controls. However, determining the correct Accessibility Role to use isn't always straightforward, and sometimes there is no correct value.

For example, in the case of the Lookup Textbox control. A Lookup Textbox functions almost like a cross between a textbox and a combo box. It looks like a textbox for all intents and purposes, but as the respondent begins to type a response into the control, a menu will pop up with a limited list of response options, pulled from the larger, complete list in the background, and based on the characters the respondent has typed so far. Crucially, unlike regular textboxes, if the respondent simply types their response into the box and then hits TAB to move to the next control, their response will disappear from the textbox and not be saved; the only way to enter a response into a Lookup Textbox, is to first begin typing a response to bring up the list, and then select one of the responses from the list, either by using the mouse or the ENTER key.

There does not appear to be any Accessibility Role for "lookup textbox," not only in Blaise, but in the wider world of HTML-accessibility. The closest options we were able to find were "textbox" or "combobox," but neither of those correctly communicates the correct method of using the control.

If we set the Accessibility Role to "textbox," then visually impaired users will expect to type their response into the box and then use TAB to move to the next control, as they would for any other textbox; they will have no way of knowing about the pop-up menu or have any way to figure out how to use it.

On the other hand, if we set the Accessibility Role to "combobox," then the result ultimately remains the same. In this case, visually impaired users will expect to use the Down arrow on their keyboard to access the drop-down menu. But unlike a combobox, there is no menu in a Lookup Textbox until the respondent has typed something into the box first.

This is a critical problem for AT-users, as it leaves no way for them to know how to correctly interact with the control. To get around this problem, we created a Text control in the resource database, that listed the instructions for how to use the keyboard to enter a response into the Lookup Textbox. We then added that Text to the ScreenReaderText expression for the control on the LookupTextbox response value template.

As a result, the screen reader text now provides the AT-user direct instructions for the control, without having to rely on interpretation of the Accessibility Role property.

6. Best Practices

Through experience in developing some customized controls and methods to address the accessibility challenges, we have developed a set of best practices that allow us to do the job more efficiently. The following sections describe some of these tips for applying these methods.

6.1 Images

Every image in a 508 compliant instrument needs to have an alt-text description of the image that can be communicated to the screen reader. This description should be as accurate as possible and must explicitly include any language that is shown in the image.

To satisfy this requirement, we added a roletext to fields that include images, initialized to the alt text string for the specified image. We then updated the ScreenReaderText expression on our image controls to look for this roletext.

6.2 Categories

Category texts should not be included in the tab order (i.e., their TabIndex property should be set to -1), even if they have OnClick events that allow them to be clicked to select the response. For keyboard-only users, it can be frustrating to have to TAB twice to move between categories. But more importantly, screen readers will often interpret these category texts as buttons when they are included in the tab order, which can make the screen confusing and hard to understand for low- or no-vision users who can't see the layout on the page.

6.3 Color Contrast

We tend to take for granted that everyone sees the same colors all the time, but that's not true at all. We each see colors a little differently, so a layout that looks pleasing to our own eyes could also be entirely impossible for another person to see at all.

When selecting colors for your instrument layout design, it's important to remember to check their color contrasts first to ensure they are accessible. You wouldn't want to settle on a logo and color scheme, only to find out later that people with a certain type of color-blindness won't be able to see it.

7. Where to Go from Here

Use of mobile tools and customizations to increase accessibility via mobile devices will be required as new technologies are introduced. Improved automation of tools and tests to increase efficiency of the test cycle will allow for more agile development. Enhanced adaptation to newer technologies and continued

collaboration on best practices to adapt to changing requirements for compliance will allow us to meet increased demand in delivering fully accessible Blaise instruments.

8. Conclusions

As we adapt to new technologies, our plans and implementation of the tools for accessibility must continue to meet new requirements and address challenges related to newer devices. We expect to continually customize our controls and the tools and methods we as coders use to test whether our surveys are fully accessible. Collaboration among the coding community to share our experiences and challenges will continue to enhance our knowledge and successes in meeting these demands.

9. Notes

¹U.S. General Services Administration. *Section508.gov*. Accessed March 23, 2025. <https://www.section508.gov/create/documents/>.

²World Wide Web Consortium (W3C). *Web Content Accessibility Guidelines (WCAG) 2.1*. Accessed March 23, 2025. <https://www.w3.org/WAI/GL/WCAG21/>.

³NV Access. *Non-Visual Desktop Access (NVDA)*. Accessed March 23, 2025. <https://www.nvaccess.org/>.

⁴TPGi. *Colour Contrast Analyser (CCA)*. Accessed March 23, 2025. <https://www.tpgi.com/color-contrast-checker/>.

⁵U.S. Social Security Administration. *Accessible Name and Description Inspector (ANDI)*. Accessed March 23, 2025. <https://www.ssa.gov/accessibility/andi/>.