Microsoft Position Paper on Annotations

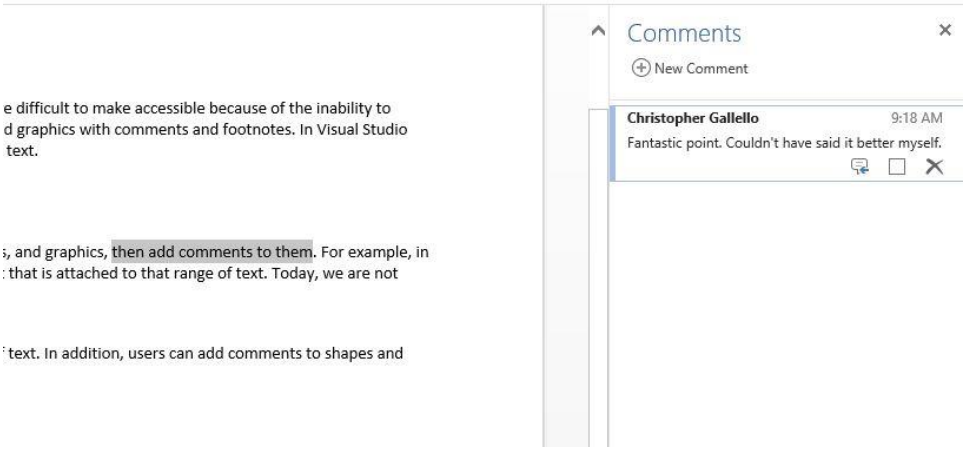
Chris Gallello, Program Manager, Office Online, Microsoft

Across various web applications built by Microsoft, there are features that are difficult to make accessible to blind users because of the inability to mark up annotations on the web. In Office Online, users are able to annotate text and graphics with comments and footnotes. In Visual Studio Online, code errors allow ranges of code to be highlighted and marked up with error text. The following document outlines a proposal for extending ARIA in order to allow for screen readers to provide a better annotation experience to end users.

Feature Examples

Office Online Comments

Word Online and PowerPoint Online allow users to select text, images, and shapes, and add comments to them. For example, in Word Online, a user can highlight text, then click on the "New Comment" button to add a comment about that specific range of text. Today, we are not able to provide that semantic relationship to the screen reader, even though sighted users see that relationship in a visual way.



In addition to text, users can select other elements that can be found across the Office Online suite, such as images, shapes, and charts. This is done in a similar way - a user first selects the element that they would like to comment on, then clicks on the "New Comment" button.

Word Online Footnotes and Endnotes

Soon, Word Online will allow users to add footnotes and endnotes to their documents, which are useful when writing a document using outside sources that should be referenced, or when additional explanatory text would be distracting to the main document. As with comments, we are not able to provide a semantic relationship between the number representing a footnote and the actual footnote text itself.

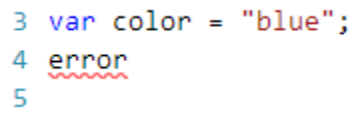
Word Online spelling errors

Word Online currently manages spelling errors separately from the browser in order to incorporate the dictionary that Microsoft has built for the installed version of Word. Thus, any red underlines are drawn

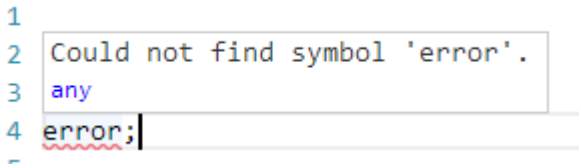
using CSS as opposed to letting the browser handle them. Because of this, the browser does not natively recognize these as misspelled regions, and we have a desire to mark these up separately using ARIA in order to ensure that blind and low vision users can understand them. This scenario is a bit different than the ones above, because the annotation does not refer to a separate piece of text - it's merely a visual annotation that we would like to provide to accessibility users.

Visual Studio Online code errors

Visual Studio Online annotates errors inside of source code with colored underlines, using red for errors and green for warnings. The accompanying message is accessible by either hovering over the underlined region or using a keyboard shortcut to expand an in-line warning.



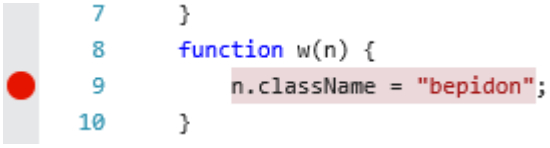
When hovering over a code error, an annotation comes up.



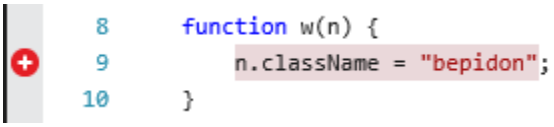
It would be desirable to express these annotations in ARIA so that assistive technology can announce their presence during code authoring and reading.

Visual Studio Online breakpoints

Another scenario of note is the presence of breakpoints, which can be toggled on a given statement and have an accompanying icon in the glyph margin:



These breakpoints may also be conditional and only activate in some cases:



Being able to announce the presence and type of breakpoint through markup would greatly assist in achieving parity with the visual display of this information provided by the icon.

User Experience of Annotations

The user experience for a blind or low vision user should be very similar and follow some key interactions that sighted users already get:

- While the user is reading through the main text, they are able to notice that text is annotated based off of a visual cue.

- The reader has the option to read the annotation or continue reading the main content.
- If the reader decides to read the annotation, they are able to find the annotation because it is visually linked to the annotation.
- If the reader decides to continue reading the main content after reading the annotation, they are able to resume reading quickly thanks to the visual link.
- If the reader decides to interact with the annotation, they may do so using any relevant UI associated with the annotation.

In order to provide a consistent user experience, the user should be able to make use of a screen reader to interact with annotations:

- As the reader reads through some text, the fact that an annotation is present can be announced by the screen reader.
- The reader places their view on the annotated text, at which point they can use a keyboard shortcut or gesture to hear the annotation.
- The reader may decide to explore the UI around the annotation. They may use a keyboard shortcut or gesture to move their focus to the annotated text.
- The reader may decide to jump back to the text in the same position where they left off. They may use a keyboard shortcut or gesture to resume reading.

Proposal

ARIA should be extended to support annotating DOM elements in order to provide the screen reader with two new pieces of information:

- The type of annotation
- A semantic relationship between the annotated DOM element and a separate DOM element which may contain additional information

The following is an example set of new ARIA tags in order to properly mark up annotations. **This example serves as a starting point and is not a final design.**

Proposed tag	Value	Description
`aria-annotationtype`	A localized name of the annotation type that applies to the marked up region.	A developer should put this around any element(s) that is being annotated. The value should be a human readable name that might get read out by the screen reader - such as "Comment" or "Footnote"
`aria-annotatedby`	A space delimited list of element IDs of regions which hold the annotation text that applies to the marked up region.	This tag works similar to describedby - a developer can use this to point to the DOM element where the annotation resides.
`aria-annotationfor`	A space delimited list of element IDs of regions for which this	This tag is the opposed of aria-annotatedby; rather than pointing to the annotation, this

	marked up region is the annotation.	is placed on the annotation, and points back to the originating text.

Basic example

For the commenting feature in Word Online, here is one potential implementation of the proposal:

Main editing area

```
<div contenteditable="true">
      <p>This is some text and <span aria-annotationtype="comment" aria-
      annotatedby="comment1" id="commentedRange">this region is
      commented,</span> but this region is not.</p>
</div>
```

Comment in the Comments Pane

```
<div id="comment1" aria-annotationfor="commentedRange">
      <div>This is the comment text.</div>
      <button>Delete</button>
</div>
```

Overlapping annotations example

The nature of generic annotations presents a problem when being expressed in HTML, because of HTML's tree structure. In order to express overlapped ranges of annotated text, app developers should separate out the overlapped regions. In the example below, pretend that the underlined text is a range of one annotation type, and the italicized text is a range of another annotation type.

The big brown duck *lazily jumped* *over the dog*.

In order to express this using the proposed ARIA annotation tags, the following can be used:

```
<span aria-annotationtype="comment" aria-annotationfor="comment1"
id="commentSegment1">The big brown duck</span>
<span aria-annotationtype="comment" aria-annotationfor="comment1"
id="commentSegment2">
      <span aria-annotationtype="reference">lazily jumped</span>
</span>
<span aria-annotationtype="reference">over the dog</span>
```

In this case, the comment grouping in the DOM might be as follows:

```
<div id="comment1" aria-annotationfor="commentSegment1 commentSegment2">
      <div>This is the comment text.</div>
      <button>Delete</button>
</div>
```

Example screen reader features

The following are example screen reader features that make use of the above ARIA annotation tags.

- **Announce when entering/exiting annotation:** When moving the cursor into and out of an annotated region of text, the screen reader could announce the annotation type. For example "Entering comment" or "Exiting comment" (where "comment" is the value of aria-annotationtype.
- **Keyboard shortcut to hear the annotation:** When the cursor is within the annotated range of text, a keyboard shortcut will read out the annotation. The screen reader can choose to not read out any buttons that fall within the annotation.
- **Keyboard shortcut to hear the annotated text range:** If a user's screen reader focus is in an annotation, a keyboard shortcut will read out the original annotated text range.
- **Keyboard shortcut to move focus between an annotated text region and the annotation:** When the cursor is within the annotated range of text, a keyboard shortcut will move the user's focus to the first focusable control within the annotation.
- **Keyboard shortcut to "resume reading":** After the user jumps from an annotated text region to the annotation in order to interact with the annotation and other relevant UI, they may want to resume reading in the main text area. Once the original keyboard shortcut to jump to the annotation is activated, the screen reader remembers where it left off, and using this keyboard shortcut will move focus back to the original text, even if the user has interacted with a variety of different UI.

Discussion topics

There are a few open issues that should be discussed in reference to the above proposal:

- **Verbosity/importance control** - Annotations are almost always ancillary pieces of information to the main text, but there are some cases where they are important and helpful to the user. Allowing users to have control over when they hear annotations would be very helpful. More specifically, there could be a list of common Annotation Types that are outlined in ARIA that are recognized. Screen readers could then implement custom switches for these outlined types, and beyond that, provide a best faith effort of plumbing them through to the user. Some examples of recognized Annotation Types include:
    - Comment
    - Footnote
    - Endnote
    - Spelling error
    - Code error
    - Warning
    - Breakpoint
    - Reference
- **Hidden annotations:** What if the annotation is currently hidden? For example, text is annotated and highlighted visually, but the comment is not visible until clicking on it. Until clicking on it, the comment and associated UI is in the DOM, but visually hidden.
    - Should screen readers be smart enough to know that the UI is hidden, and therefore focus should not move to it?
    - Should it be a 'best practice' for web developers to remove comments from the DOM when they are visually hidden?