Table of Contents

Introduction 2

Support for assistive technologies through accessibility services. 2

Accessibility Gap Analysis of SVG 4

Filling the Gaps – Detailed discussion points 6

Semantic Structure and Containment Hierarchy 6

The Role Attribute 7

The Role Value Taxonomy and Use Case Development 7

Develop New ARIA States and Properties for Graphics 8

Keyboard Navigation 8

Follow system settings for font and color settings 9

Panning and Zooming 9

Caret and Selection Tracking 9

Use Cases 10

SVG 2.0 Accessibility

***Draft***

# Introduction

Every major browser now supports Scalable Vector Graphics (SVG) making it an open supported retained mode, two-dimensional graphics technology capable of supporting clear rendering of graphics even when zoomed. Furthermore, the growth of big data on the Web and the limited number of mobile platforms on which Flash is supported make SVG and HTML5 Canvas the essential 2-D drawing engines for complex visualizations on the Web. The drawback is that it is not accessible, restricting the use of some rich visualization solutions that make use of it. The purpose of this document is to define a common strategy among the major browser manufacturers to fill accessibility gaps in SVG 2.0.

# Support for assistive technologies through accessibility services.

Most major browsers support assistive technologies, like desktop applications, by mapping content to platform accessibility API. This API is used by assistive technologies to process web content. The following graphic shows how SVG content could map to platform accessibility APIs.



**Figure 1.0 SVG Markup to Platform Accessibility API**

Figure 1.0 shows how SVG markup, with WAI-ARIA markup could get mapped to an SVG DOM node that in turn is mapped to an accessible API object in the platform accessibility API. An Accessible object in the platform accessibility API exposes:

* The role or type of object that semantically has meaning to the user
* The accessible state or properties associated with the object (selected, checked, etc.)
* Name of the object
* Event notification to convey changes in state, DOM structure changes, etc.
* A means to walk the tree or containment hierarchy
* Special interfaces to text, tables, control patterns, etc.

The section raises a number of issues that need to be addressed in order for SVG to be accessible which will be covered in the following sections.

Of the items needed, SVG only provides the <title> element and the <description> element which are defined in the document [Accessibility Features of SVG](http://www.w3.org/TR/SVG-access/) . These features allowed an author to define a name and help text for an SVG drawing object that was the extent of the requirements that one might expect from WCAG 1 accessibility guidelines. These features do not provide for full interoperability with assistive technologies and in that they do not convey the intent of the author and in the case of title may create unwanted side effects such as a tooltip to pop up when hovering over the drawing object.

# Accessibility Gap Analysis of SVG

This table shows a preliminary accessibility gap analysis of SVG and it is a summary of the detailed

|  |  |  |
| --- | --- | --- |
| Feature |  | Questions? |
| Role attribute | Limited to XLink | * Do we have it take a CURIE and namespace reference ARIA roles?
* Do we simply take non-prefixed ARIA values from the ARIA specification?
 |
| Role value taxonomy | Work to use the existing ARIA specification as much as possible. If we need to expand on it we will do the work here and work with the ARIA working group to integrate into ARIA 2.0. This allows Canvas to benefit.**Unless a role is applied to an element it should be considered presentational for performance reasons on behalf of the browser and AT** |  Note: Apple and Mozilla already map ARIA roles on existing SVG elements to platform accessibility API so that may not be practical. |
| ARIA states and properties | Work to use the existing ARIA specification as much as possible. If we need to expand on it we will do the work here and work with the ARIA working group to integrate into ARIA 2.0. This allows Canvas to benefit. |  |
| Keyboard Navigation | Use tabindex from HTML5 | * Do we also use SVG Tiny’s keyboard navigation model?
* Should we include navigation vehicles from other document formats?
* Do we use a combination of these?
 |
| Label capability with tooltip | Yes <title> |  |
| Label capability without tooltip | Use ARIA labeling  |  |
| element description support | Limited to hidden <desc> text | * I suggest we use what ARIA 1.1 provides as it should include an aria-describedat that will take a URI. ARIA 1.0 already provides an aria-describedby that can reference visible content
 |
| Semantic structureContainment hierarchy | Make use of SVG DOM and* <g> element
* aria-owns
* aria-flowto
 | * Will these mechanisms be sufficient?
* How do we manage the amount of information exposed

 |
| Rich Text Caret and selection support |  | Do we support any rich text editing in SVG?* ARIA 1.1 is looking at a Caret role
* ARIA 1.1 has aria-selected
* No vehicle to provide selection location
* Do we do this for SVG 2.0?
 |
| Device Independent Event support |  | Recommend we look at Indie UI |
| Follow System settings for for font and color |  | SVG provides no interactive UI components per say but it would be good to be able to follow system settings for font and color information. To do this we need to be able to expose the settings to an SVG application. * Should we simply expose high contrast mode that could be adequate itself?
* Should we defer this to Indie UI so that what we use would be applicable to all web application technology.

I recommend we look to Indie UI context awareness APIs for this but we need consensus. |
| Pan and Zoom document areas for all users (not just those considered with low vision) |  | To what extent should we go with this?* Pan and Zoom within a selected area
* Pan with and Zoom at the pointing device location
* Zoom and pan around an object with focus
* Directed Pan and Zoom based on Indie UI events
 |
| Animation |  | Do we do something special for animation? |

# Filling the Gaps – Detailed discussion points

This section covers some of the detailed discussion from the gap analysis above.

## Semantic Structure and Containment Hierarchy

Up until now the limited accessibility work on SVG has been dependent on the use of the SVG DOM to apply accessibility information. The SVG DOM is created from a parsing of XML declarative drawing directives that in no way convey semantic structure such as the following:

* A bar within a bar chart
* An option within a listbox that could be used to represent a pie chart

In discussions with Google, Microsoft, and Mozilla it was agreed that we should follow the current declarative model defined by ARIA and use the <g> element to produce a structured, semantic containment hierarchy and to use WAI-ARIA to apply accessibility semantics on top of the that structure. Currently, both Safari and Firefox map role information to platform accessibility API in SVG so this strategy is consistent with best practices today. How well the mapping works and how consistent the mappings are is yet to be determined.

WAI-ARIA provides additional properties that can facilitate providing structural information to the assistive technology:

* aria-owns – states that an element owns another element as a child
* aria-flowto – states that an object flows to another object (such as a rendered network topology)
* aria-setsize – indicates the size of a set
* aria-posinset – indicates the position in a set
* aria-level – indicates how deeply nested an element is in a structure

Al challenge we will have stems from the fact that SVG is a retained mode graphics engine. All drawing objects are in the DOM even though they may not be perceivable at the current magnification level. An example would be a house drawing where you zoom in for more detail on an object. A assistive technology, when accessing the accessibility services layer could be exposed to all the drawing objects and this could create performance and memory consumption issues for the browser who is maintaining an accessibility tree of the SVG document or the assistive technology who maintains a virtual model of the SVG document. We must take this into consideration when making SVG accessible.

## The Role Attribute

The role defines the type of an object in a UI. Given that SVG will largely be used within an HTML document it would be best to not use the namespace feature of XML and have role simply refer to role values in the ARIA taxonomy negating the need for namespaces.

## The Role Value Taxonomy and Use Case Development

The only Role Value Taxonomy is only defined by WAI-ARIA and it encompasses:

* Interactive Widgets such as checkboxes, list boxes, grids, and sliders
* Non-abstract structural semantics roles such as region, landmark regions, and document
* Abstract roles that define a class of semantic roles but are not directly applicable to markup.
* Restrictions on the states and properties that can be applied
* An inheritance model for Roles to inherit from their superclasses.
* Required child elements for roles

***If new ARIA roles are needed we should subclass portions of the WAI-ARIA taxonomy for graphics. This may also require us to work with platform vendors to extend their platform accessibility API.***

***ARIA 2.0 has an issue to address new roles for SVG. ARIA 2.0 work has not begun. We should seed new values in the SVG working group and then coordinate integration of them into ARIA 2 so that canvas may also make use of them.***

Regardless of the approach here we must produce a set of use cases on which to build our SVG accessibility strategy.

## Develop New ARIA States and Properties for Graphics

***If new ARIA states and properties we should extend WAI-ARIA for graphics. This may also require us to work with platform vendors to extend their platform accessibility API.***

***ARIA 2.0 has an issue to address new states and properties for SVG. ARIA 2.0 work has not begun. We should seed new values in the SVG working group and then coordinate integration of them into ARIA 2 so that canvas may also make use of them.***

If we do not wish to limit ourselves to the existing ARIA role, states, and properties and expand on what ARIA provides we may need to create new states and properties. If this is done we will need to define accessibility API mappings for states and properties for the platforms. We should

## Keyboard Navigation

SVG 1.1 has no keyboard navigation and SVG Tiny 1.2 keyboard navigation is inconsistent with HTML keyboard navigation. In meetings between Google, Mozilla, IBM, and Microsoft it was generally agreed that HTML5 tabindex should be used for keyboard navigation. There are also plans to integrate tabindex in a future version of WebKit. This allows for a consistent DOM keyboard navigation model. Microsoft currently supports focusable but had no issue with supporting HTML5’s tabindex instead.

HTML5 supports a tabindex property on all its elements. If the tabindex is set to “0” the element is placed in the tab order based on the document order. A value greater than zero can change the order in which the element appears in the tab order to override the default DOM navigation order. A value of “-1” allows the author to programmatically set focus on the element with the tabindex having this value but it does not place the element in the tab navigation order. All interactive controls, such as form controls are placed in the tab navigation sequence similar to having a tabindex=”0”.

## Follow system settings for font and color settings

It is a 508 requirement to meet system settings for font and color. Since SVG has no UI controls at this point we could defer this to Indie UI to expose the information through their Context Awareness API settings or provide a new API for SVG to do this. I recommend we defer this to Indie UI and let them handle it.

## Panning and Zooming

Today, Web browsers support page level zooming using the cntrl+/- keys for zooming but no panning. Unlike HTML content, SVG uses scalable vector graphics such that when zooming we don’t experience stair stepping that we typically see with bitmap graphics. Also, for complex graphics and charts it may be advantageous to be able to zoom to parts of those graphs and pan around those portions of the graph. This would also be of value to mobile devices that have a limited display real estate. This is a benefit to all low vision users – including those who don’t rely on a screen magnifier. With the changes we are making for accessibility we should consider adding:

* Pan and Zoom within a selected area (drag your mouse and chose a box around which to pan or zoom)
* Pan with and Zoom at the pointing device location
* Zoom and pan around an object with focus
* Directed Pan and Zoom based on Indie UI events based on the current point of regard defined the pointer position or the object having keyboard focus. IndieUI is intended to supply higher level events such as ZoomIn, ZoomOut, DrillDown, etc. without worrying about the input device that provided the command.

## Caret and Selection Tracking

It is important for a screen reader or screen magnifier to be able to track when a edits or selects text. In any drawing engine there is a provision to be able to draw and there will be times in SVG where we want to be able to edit or select text. In ARIA 1.1 we are looking at a new ARIA caret and selection tracking markup that has yet to be defined but the essence of the change is to take a drawing object, give it a role of caret and bind it to an element with text in markup using an ID reference. We could then provide an offset location within the text associate with it. The bounds of the caret can be determined from the path associated with the drawing object.

# Use Cases

Although not yet covered here it will be important that we create use cases that we need to address during the course of specification and implementation to ensure we have met the SVG 2.0 accessibility goals. IBM’s main interest in SVG accessibility is for producing accessible visualized data analytics and eventually drawing objects in web-based office suites. Here are some examples:

* Scatter plots
* Bar charts
* Pie Charts
* Rich text embedded drawing in a cloud based office offering
* Drawings in presentation tools in a cloud based office offering