Map Markup Language

Draft Community Group Report 06 October 2020

Latest editor's draft:
https://maps4html.org/MapML/spec/

Editors:
Peter Rushforth (Natural Resources Canada)
Robert Linder

Authors:
Robert Linder
Joan Masó
Elf Pavlik

Participate:
GitHub Maps4HTML/MapML
File a bug
Commit history
Pull requests

Join the Community Group:
Maps for HTML Community Group

Abstract

Map Markup Language is a text format for encoding map information for the World Wide Web.

The objective of MapML is to allow Web-based user agent software (browsers and others) to display and edit maps and map data without necessary customization.

The standard HTML `map` element allows the HTML author to designate sub-areas of an image to be used as hyperlinks, creating a two-dimensional "image map". This functionality, while useful, is

limited to the simplest of mapping applications. A more generally useful `map` element would enable (functions like) zooming and panning of the map in a traditional Web mapping way, while simultaneously being simple and declarative for an HTML author to include on their Web page. This specification proposes the syntax and semantics of an extended `map` element.

The applications of Web maps are diverse. The wide scope of use of Web maps appears similarly broad to that of other Web media types, such as video or audio. In other words, there are a multitude of reasons for wanting to include a map on your Web page. The HTML `map` element proposed by this document should be provider-agnostic, and should only depend on Web standards, including Uniform Resource Identifiers, Document Object Model, Cascading Style Sheets and media types.

Status of This Document

This specification was published by the Maps for HTML Community Group. It is not a W3C Standard nor is it on the W3C Standards Track. Please note that under the W3C Community Contributor License Agreement (CLA) there is a limited opt-out and other conditions apply. Learn more about W3C Community and Business Groups.

Intent of this Specification §

This specification describes Map Markup Language (MapML), which is a hypertext document format for maps. This specification is also intended as a proposal to the HTML Working Group to extend the semantics of the standard HTML `map` element.

MapML is needed because while Web browsers implement HTML and SVG, including the `<map>` element, those implementations do not meet the requirements of the broader Web mapping community. On the one hand, the semantics of maps are quite different from those of Scalable Vector Graphics, while on the other hand, the semantics of the HTML `map` element are incomplete or insufficient relative to modern Web maps and mapping in general. Robust web maps are implemented by a variety of non-standard technologies. Web maps do not work without script support, making their creation a job beyond the realm of beginners' skill sets, while making them potentially inaccessible due to developer inattention. In order to improve collaboration and integration of the mapping and Web communities, it is desirable to enhance or augment the functionality of the <map> (or a similar <new>) element in HTML to include the accessible user interface functions of modern web maps (e.g. panning, zooming, searching for, and zooming to, styling, identifying feature properties, etc.), while maintaining a simple, declarative, accessible interface for HTML authors. At the same time, the DOM interface to the new or improved elements should provide low-level programmable mapping hooks in the style of the Web.

To achieve this it is necessary to define a new hypertext format (MapML) and media type which encodes map semantics.

This document is an evolving proposal to the Web and Mapping communities. Collaborators and implementation experience is requested. The intention is to define a new hypertext format (MapML) which encodes map layer semantics directly, but which leverages existing standards where possible and desirable, such as Cascading Style Sheets and OGC Simple Features, for example. MapML will provide an essential part of the contract between Web user agents and Web servers when map features are exchanged, in a manner based on the architectural style of the Web, in a similar way to how HTML provides (part of) the contract for documents.

Implementation Experience §

MapML has been implemented using custom elements by Natural Resources Canada, which participates in the W3C Maps for HTML Community Group, as well as the Open Geospatial Consortium. It is hoped that other organizations will also implement MapML and contribute their experience back to the community via the Community Group and Github.

Map Markup Language services are provided by Natural Resources Canada as part of the Canadian Geospatial Data Infrastructure.

The reader is requested to contact the editor with any known implementations of Map Markup Language or other implemented support for this proposal.

The customized built-in HTML `<map>` element and `<mapml-viewer>` autonomous custom element were built and are maintained by the Maps for HTML Community Group. [Web-Map-Custom-Element]

▸ Changes Since the Previous Draft §

Level of Endorsement by the Community §

Please join the W3C Maps for HTML Community Group if you would like to participate in the development and implementation of this specification.

You can also star us on Github.

Patent Information §

Development of this specification together with implementations is done under the terms of the W3C Software Notice and License, in accordance with the rules of the W3C Community Groups program.

How to provide feedback §

Please send comments on this specification to public-maps4html@w3.org, or WICG, or open an issue.

If you wish to make comments regarding this document, please send them to public-maps4html@w3.org (subscribe, archives).

Table of Contents

C.2 Informative references

1. Introduction §

This section is non-normative.

Map Markup Language is a text-based format which is allows map authors to encode map information as hypertext documents exchanged over the Uniform Interface of the Web. The format is defined using some characteristics of the *HTML Standard*, *MicroXML*, and *The GeoJSON Format* and other standards. [HTML] [MicroXML] [rfc7946]

This specification describes a proposed enhancement to the `map` element syntax and processing model. The semantics of the existing standard `map` element allow an HTML author to create `area` elements over a standard HTML `img`, which then act as hyperlinks. The existing standard `map` element is a perfect extension point in HTML for modern Web maps, since it shares common (ancestor) abstract semantics (2D area on which authors can draw hyperlinks), which can be relied on as widely implemented and stable fallback markup processing by HTML authors targeting newer or future user agents which implement this proposal. This proposal to update the `map` element enhances the core functionality of the `map` element to include cartographic Web applications. The intention of this specification is to describe the `map` element independent of underlying implementation - whether that be native browser or custom elements. Details specific to how to author a Web page using the Custom Elements implementation can be found in the [Web-Map-Custom-Element] project.

2. Conformance §

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words *MAY*, *MUST*, *MUST NOT*, and *SHALL* in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1 Conforming documents §

Conforming documents are documents that comply with all the conformance criteria for MapML documents. For readability, some of these conformance requirements are phrased as conformance requirements on authors; such requirements are implicitly requirements on documents: by definition,

all documents are assumed to have had an author. (In some cases, that author may itself be a user agent — such user agents are subject to additional rules, as explained below.)

User agents fall into several (overlapping) categories with different conformance requirements.

Web browsers and other interactive user agents

Web browsers are one of the primary client technologies anticipated for MapML, however other categories of interactive client could also be developed, for example as extensions to / plugins for traditional Geographic Information Systems software.

Non-interactive presentation user agents

User agents that process MapML documents purely to render non-interactive versions of them must comply to the same conformance criteria as map browsers, except that they are exempt from requirements regarding user interaction.

While MapML documents are thought to describe map semantics in a standard way, the scope of this specification is intended to not overlap that of HTML documents themselves. As such, this conformance class may not be relevant, except when consuming MapML within the context of an HTML instance's user agent. For example, the legend and other supporting map related information is out of scope for MapML, whereas it is conceivably in-scope for HTML documents.

> NOTE
>
> Typical examples of non-interactive presentation user agents are printers.

User agents with no scripting support

Scripting can form an integral part of a Web application. User agents that do not support scripting, or that have scripting disabled, however should still be able to display and enable map-user interaction via the affordances described in this document.

Conformance checkers

Conformance checkers must verify that a document conforms to the applicable conformance criteria described in this specification.

The term "MapML validator" can be used to refer to a conformance checker that conforms to the applicable requirements of this specification.

> NOTE
>
> Schemas cannot express all the conformance requirements of this specification. Therefore, a validating XML processor and a DTD cannot constitute a conformance checker.
>
> To put it another way, there are three types of conformance criteria:
>
> 1. Criteria that can be expressed in a schema (RelaxNG, XML Schema etc).
> 2. Criteria that cannot be expressed by a DTD, but can still be checked by a machine.
> 3. Criteria that can only be checked by a human.
>
> A conformance checker must check for the first two. A simple RelaxNG-based validator only checks for the first class of errors and is therefore not a conforming conformance checker according to this specification.

Data mining tools

Applications and tools that process HTML and MapML documents for reasons other than to either render the documents or check them for conformance should act in accordance with the semantics of the documents that they process.

Authoring tools and markup generators

Authoring tools and markup generators must generate conforming documents. Conformance criteria that apply to authors also apply to authoring tools, where appropriate.

> NOTE
>
> For example, a markup generator must ensure that polygons have three or more vertices, the first and last of which have the same location.

> NOTE
>
> In terms of conformance checking, an editor has to output documents that conform to the same extent that a conformance checker will verify.

When an authoring tool is used to edit a non-conforming document, it may preserve the conformance errors in sections of the document that were not edited during the editing session (i.e. an editing tool is allowed to round-trip erroneous content). However, an authoring tool must not claim that the output is conformant if errors have been so preserved.

Authoring tools are expected to come in two broad varieties: tools that work from structured

databases, and tools that work on an interactive What-You-See-Is-What-You-Get media-specific editing basis (WYSIWYG).

All authoring tools, whether WYSIWYG or not, should make a best effort attempt at enabling users to create accessible, well-structured, and efficient content.

This document contains explicit conformance criteria that overlap with some RNG definitions in requirements. If there is any conflict between the two, the explicit conformance criteria are the definitive reference.

3. Use Cases and Requirements §

This section is non-normative.

3.1 Use Cases §

The following usage scenarios illustrate some of the ways in which **Map Markup Language** might be used for various applications:

- **Tiled maps**

  Modern map services are dominantly created by composing adjacent map 'tiles' (often bitmapped images) into a coherent and unified image of a map.

- **Image-based maps**

  Web services typically produce a single geo-referenced image which constitutes a map. MapML encodes URL references to such images, together with appropriate georeferencing metadata.

- **Vector-based maps**

  Geospatial databases, for example PostGIS, are able to serve and manipulate vector representations of features which can be symbolized according to their properties.

- **Multi-layer maps**

  Maps are often required to overlay information in such a way as to enable both visual and automatated spatial relationship processing. MapML will support the layering of information using the painters model, wherein elements are drawn in document order overtop of earlier elements, together with CSS styling (transparency) techniques.

- **Use CSS for map styles**

  MapML elements should be style-able with CSS rules supplied by the MapML author. For example, the MapML author should be able to link to a stylesheet from a MapML document.

- **Location search**

  A MapML service might want to allow service consumers to search within the contents of a service for a place name or an address, over which the map is repositioned when a selection is made. MapML should provide (markup) facilities to enable such searches without forcing the client download large amounts of geospatial data.

- **Hyperlinks between and within services**

  MapML should allow service-level links such that service providers can link together / federate to provide apparently seamless spatial coverage.

- **Feature identification**

  If a MapML document includes features, the MapML author should be able to markup any or all of those features in a way which lends itself to feature identification and property display.

- **Attribution**

  MapML documents encode citation information on a per-request basis, making it available as markup in the response.

The following use cases illustrate some of the ways in which **the `map` element** might be used in Web mapping applications:

- **Multiple, user-controllable map layers**

  The `map` element allows HTML authors to add zero or more map layers to the map, with optional controls enabled declaratively.

- **Selection / identification / description of map features**

  The user should be able to interact with individual map features as individual elements, with or without hyperlinks.

- **Rescaling to expose greater detail (zooming)**

  The user should be able to zoom in or out on the map to expose more or less detail as desired / possible.

- **Repositioning to examine relationships between places (panning)**

  The user should be able to pan the map so as to examine locations other than the default or initial location displayed, as desired / possible.

- **Using Cascading Style Sheets to enable HTML authors to control the style of their own map content.**

  The HTML author should be able to use Cascading Style Sheets on the `map` element and its children so as to create the user experience desired, with the caveat that the style of features in map sources / services should be under the control of the exclusive control of the map content author, per MapML requirements. In brief, the HTML author should not be able to style content that does not belong to them, but should be able to create and style features displayed on the map from their own domain.

- **Map links of various shapes, including points, lines, polygons, circles and rectangles**

  HTML authors should be able to place and style links on a web map, allowing navigation of the browser through user selection.

- **Links within and between services to enable a "Web of maps"**

  A map layer could be "distributed" over many URLs, and the user agent would seamlessly present the contents of those URLs to the user, possibly although not necessarily as though they were a single end point. For example, a response document should be able to communicate a link to an adjacent zoom level or geographic area in the response entity and the user agent could conditionally follow that link depending on the gestures of the user.

- **API-level access to the map, its layers, features and events**

  Scripting has an important place in the presentation of map information. Declarative web maps should be able to be progressively enhanced through the use of script. Furthermore, web maps should emit map- and control-specific events which support progressive enhancement.

- **Crawler-generated location-enabled, searchable indexes of Web content**

  Web maps should serve to geocode content in a Web page, such that crawlers may be able to augment their indexes with location information, leading to enhanced discovery possibilities for Web search users.

3.2 Requirements §

- **Uniform interface**

  No URL recipes should be required to be supported by clients - support for simple opaque URLs and standardized media types should be all that is required clients. Application state / state transitions should be conveyed in markup.

- **Implementation commitments**

  Simple, lightweight server software should be available to allow authors to serve spatial resources as MapML.

- **Accessible custom element**

  One or more accessible Custom Element clients should be available to demonstrate client functionality.

- **Ease of authoring**

  A range of authoring situations should be supported, from simple single file services to robust MapML services over large volumes of framework data.

4. Web Maps §

The `map` element, in conjunction with any `layer` and `area` element descendants, defines an interactive `map` media element.

4.1 Web mapping elements and API §

4.1.1 The `<map>` element §

> **Categories**:
>
> Flow content.
> Phrasing content.
> Palpable content.
> Embedded content.
> Interactive content.
>
> **Contexts in which this element can be used**:

Where embedded content is expected.

Content model:

If `lat`, `lon`, and `zoom` are NOT present: Transparent; otherwise: Zero or more `layer` or `area` elements.

Content attributes:

Global attributes.
`zoom` — A positive integer zoom (scale) value.
`lat` — A decimal WGS84 latitude value for the map center.
`lon` — A decimal WGS84 longitude value for the map center.
`projection` — A case-sensitive string identifier of the MapML coordinate reference system used by the map. Default is `OSMTILE`.
`controls` — Show user agent controls.
`controlslist` — Show/hide specific user agent controls.
`width` — Horizontal dimension.
`height` — Vertical dimension.

Tag omission in text/html:

Neither tag is omissible.

Allowed ARIA role attribute values:

`application`.

Allowed ARIA state and property attributes:

Global aria-* attributes.

DOM interface:

WebIDL

```
[Exposed=Window, LegacyFactoryFunction=Map(unsigned long width, unsigned
long height, double lat, double lon, unsigned short zoom, optional
DOMString projection, boolean controls)]
interface HTMLMapElement : HTMLElement {
  [HTMLConstructor] constructor();

  [SameObject] readonly attribute HTMLCollection layers;
  [SameObject] readonly attribute HTMLCollection areas;
  [CEReactions] attribute unsigned short zoom;
  [CEReactions] attribute double lat;
  [CEReactions] attribute double lon;
  readonly attribute DOMString projection;
```

```
    [CEReactions] attribute boolean controls;
    [SameObject, PutForwards=value] readonly attribute
        DOMTokenList controlsList;
    [CEReactions] attribute unsigned long width;
    [CEReactions] attribute unsigned long height;
    undefined zoomTo(double latitude, double longitude, optional unsigned
  short zoom);
  };
```

The map element, in conjunction with child layer and optionally area elements, defines a map media element.

***map*.areas**

Returns an HTMLCollection of the area elements in the map.

***map*.layers**

Returns an HTMLCollection of the layer element children of the map.

The areas attribute must return an HTMLCollection rooted at the map element, whose filter matches only area child elements.

The layers attribute must return an HTMLCollection rooted at the map node, whose filter matches only layer child elements of this map element.

The ***zoom*** attribute indirectly identifies an initial scale of the map.

The zoom IDL attribute must reflect the content attribute of the same name.

The ***lat*** and ***lon*** attributes locate the initial center of the map.

The lat and lon IDL attributes must reflect the content attribute of the same name.

The ***controls*** attribute is a boolean attribute. If present, it indicates that the author has not provided scripted controls and would like the user agent to provide its own set of controls.

If the attribute is present, or if scripting is disabled for the media element, then the user agent should ***expose a user interface to the user***. This user interface should include features to zoom in, zoom out, pan to an arbitrary position in the content, and show the media content in manners more suitable to the user (e.g. fullscreen map or in an independent resizable window). Other controls may also be made available.

The `controls` IDL attribute must reflect the content attribute of the same name.

The ***`controlslist`*** attribute, when specified, helps the user agent select what controls to show on the media element whenever the user agent shows its own set of controls. Its value must be an unordered set of unique space-separated tokens that are ASCII case-insensitive. The allowed values are ***`nofullscreen`***, ***`nolayer`*** and ***`nozoom`***.

The `nofullscreen` keyword hints that the fullscreen mode control should be hidden when using the user agent's own set of controls for the media element.

The `nolayer` keyword hints that the layer control should be hidden when using the user agent's own set of controls for the media element.

The `nozoom` keyword hints that the zoom controls should be hidden when using the user agent's own set of controls for the media element.

> NOTE
>
> *Hiding these aspects of the user agent's own controls does not necessarily disable the related functionality. For example, the user agent might present the same functionality through a context menu or keyboard shortcut.*

The `controlsList` IDL attribute must reflect the value of the `controlslist` content attribute.

The supported tokens for `controlsList`'s DOMTokenList are the allowed values defined in the `controlslist` attribute and supported by the user agent.

A user agent *MAY* ignore the author's preference when it makes sense.

> NOTE
>
> A user agent might ignore the `nofullscreen` keyword if the content area containing the map is small, such as on a mobile device.

The ***`projection`*** attribute value identifies the coordinate system for the map and all the `layer` element children, with the exception of `layer` children in the WGS84 projection. WGS84 serves as a "wild-card" projection; features encoded in longitude, latitude values according to this projection can often be re-projected for use on maps encoded in other projections. As such, `layer` children do not have a projection attribute, in that they are verified to share the projection declared by their parent `map` element. Once established, the projection of the `map` is read-only. Unless otherwise specified, units are based on pixels projected and transformed from the units of the underlying projection system. The

complete definition of these coordinate reference systems can be found in the Tiled Coordinate Reference Systems table.

The `map` element supports the ***`width`*** and ***`height`*** dimension attributes. [HTML]

The default object size is a width of 300 CSS pixels and a height of 150 CSS pixels. [css-images-3]

4.1.2 The `<layer>` element §

> **Categories:**
>
> Flow content.
> Phrasing content.
> Palpable content.
> Embedded content.
> Interactive content.
>
> **Contexts in which this element can be used:**
>
> As a child of the `map` element.
>
> **Content model:**
>
> If the `src` attribute is present: Nothing. If no `src` attribute is present: Metadata content describing nested Map Markup Language content.
>
> **Content attributes:**
>
> Global attributes.
> `src` — Address of the resource.
> `label` — String label for the layer in the layer control, if displayed.
> `checked` — Layer on/off status.
> `disabled` — Sets the disabled state of the layer in the layer control.
> `hidden` — Visibility status of the layer in the layer control.
> `referrerpolicy` — Referrer policy for fetches initiated by the element.
> `crossorigin` — A CORS settings attribute, specifies how the element handles crossorigin requests.
>
> **Tag omission in text/html:**
>
> Neither tag is omissible.
>
> **Allowed ARIA role attribute values:**
>
> None.
>
> **Allowed ARIA state and property attributes:**

Global aria-* attributes.

DOM interface:

```
[Exposed=Window]
interface HTMLLayerElement : HTMLElement {
  [HTMLConstructor] constructor();

  readonly attribute DOMString src;
           attribute DOMString label;
           attribute boolean checked;
  readonly attribute boolean disabled;
           attribute boolean hidden;
  readonly attribute LegendLink[] legendLinks;
           attribute DOMString referrerPolicy;
           attribute DOMString? crossOrigin;
};

[Exposed=Window]
interface LegendLink {
  [HTMLConstructor] constructor();

  readonly attribute DOMString type;
  readonly attribute DOMString rel;
  readonly attribute DOMString href;
  readonly attribute DOMString title;
  readonly attribute DOMString lang;
  readonly attribute DOMString hreflang;
  readonly attribute DOMString sizes;
};
```

The `map` element displays its child `layer` elements similarly to "acetate" layers on a (paper) map. The layers follow the painters model, whereby `layer` elements are displayed on top of earlier siblings according to their document order. `layer` transparency is controlled by the CSS opacity property. [SVG2] [css-color-3]

In contrast to the HTML media elements' `source` child element where a single `source` element is selected by the user agent for play, the `map` element's child `layer` elements are visually combined with sibling `layer` elements and presented by the parent `map` element as a pan-able, zoom-able two-dimensional image. The display state (on / off) of the content represented by the `layer` element is controlled by the `checked` boolean attribute and corresponding property. If checked, the `layer` is

drawn on the map in the sequence it is found in the parent element; if not `checked` the `layer` is not drawn but should remain visible in the `map` element's representation of `controls` in an "unchecked" state.

The `src` attribute value is the URL of a web resource encoded in Map Markup Language.

The `hidden` boolean attribute can be set to remove the `layer` from the *map layer control, but it will remain displayed on the map*. In order to remove the `layer` from the map display, it can be have its `checked` property toggled, or be removed from the DOM.

The `disabled` property is a read-only boolean indicator of the visibility of the layer on the map. If the layer is not visible due to errors, including projection, zoom or extent mismatch, the property will be true and if the layer is present in the layer control (i.e. `hidden` is false), it will be disabled in that control i.e. not checkable.

4.1.3 The `<area>` element §

Categories:

Flow content.
Phrasing content.

Contexts in which this element can be used:

As a child of the `map` element.

Content model:

Nothing.

Content attributes:

Global attributes.
`href` — Address of the hyperlink.
`alt` — User-visible label.
`coords` — Coordinates for the shape to be created on the map.
`shape` — The kind of shape to be created on the map.

Tag omission in text/html:

Neither tag is omissible.

Allowed ARIA role attribute values:

`link` role (default - do not set).

Allowed ARIA state and property attributes:

Global aria-* attributes.

Any aria-* attributes applicable to the allowed roles.

DOM interface:

WebIDL

```
[Exposed=Window]
interface HTMLAreaElement : HTMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString href;
  attribute DOMString alt;
  attribute DOMString coords;
  attribute DOMString shape;
};
```

The `area` element allows the HTML author to encode and style a limited set of spatial feature types on its parent `map` element, using familiar [HTML] image map coordinates. Such features can be optionally designated as hyperlinks to be selected by the user.

If present, the `href` attribute identifies the designated coordinate area as a hyperlink. If no `href` attribute exists, the area is designated as a non-interactive area. The `alt` attribute specifies a label which can be used by the user agent for ephemeral and non-visual labelling of the area.

If the `area` element has no `href` attribute, then the area represented by the element cannot be selected, and the `alt` attribute must be omitted.

The `coords` attribute is a comma-separated list of integer coordinate values which delimit the geometry of the (area) feature. The `shape` attribute identifies the type of geometry which is encoded as coordinates in the `coords` attribute.

The shape attribute is an enumerated attribute. The following table lists the keywords defined for this attribute. The states given in the first cell of the rows with keywords give the states to which those keywords map.

State	Keywords	Notes
Circle state	`circle`	

State	Keywords	Notes
Default state	`default`	
Polygon state	`poly`	
Rectangle state	`rect`	
Marker state	`marker`	new
Line state	`line`	new

The `shape` attribute may be omitted. The missing value default is the rectangle state.

The `coords` attribute must, if specified, contain a valid list of integers. This attribute gives the coordinates for the shape described by the `shape` attribute. The processing for this attribute is described as part of the image map processing model.

In the ***circle state***, `area` elements must have a `coords` attribute present, with three integers, the last of which must be non-negative. The first integer must be the distance in CSS pixels from the left edge of the image to the center of the circle, the second integer must be the distance in CSS pixels from the top edge of the image to the center of the circle, and the third integer must be the radius of the circle, again in CSS pixels.

In the ***default state***, `area` elements must not have a `coords` attribute. (The area is the whole area of the map on the page.)

In the ***polygon state***, `area` elements must have a `coords` attribute with at least six integers, and the number of integers must be even. Each pair of integers must represent a coordinate given as the distances from the left and the top of the map in CSS pixels respectively, and all the coordinates together must represent the points of the polygon, in order.

In the ***rectangle state***, `area` elements must have a `coords` attribute with exactly four integers, the first of which must be less than the third, and the second of which must be less than the fourth. The four values must represent, respectively, the distance from the left edge of the map to the left side of the rectangle, the distance from the top edge to the top side, the distance from the left edge to the right side, and the distance from the top edge to the bottom side, all in CSS pixels.

In the ***marker state***, `area` elements must have a `coords` attribute with exactly two integers. The first integer must be the distance in CSS pixels from the left edge of the map to the top left corner of the marker, the second integer must be the distance in CSS pixels from the top edge of the map to the top left corner of the marker.

In the ***line state***, `area` elements must have a `coords` attribute with at least four integers, and the number of integers must be even. Each pair of integers must represent a coordinate given as the distances from the left and the top of the map in CSS pixels respectively, and all the coordinates together must represent the points of the line, in order.

The intention of the `area` element is to extend the behavior of the existing [HTML] `area` element. For information pertaining to the processing of various link types created by the `area` element, please refer to the description of hyperlink processing for the [HTML] `area` element.

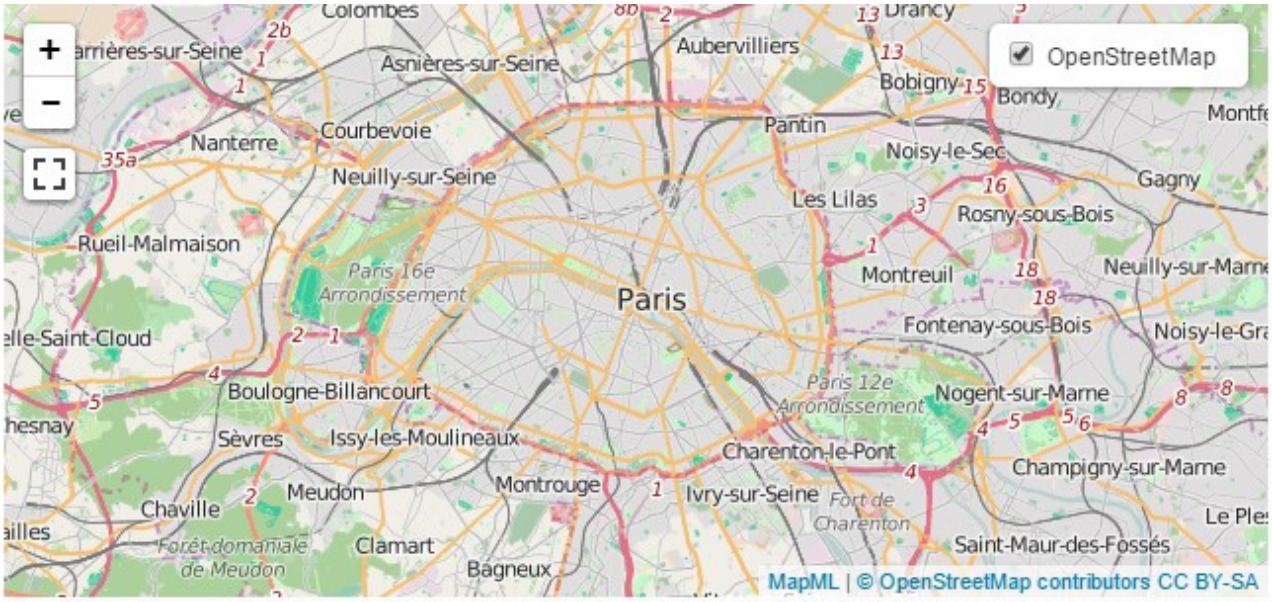
4.2 Authoring §

A ***Web map*** is an embedded media element which enables dynamic cartographic Web applications in HTML documents.

A Web map is a rectangular page area, represented in HTML in the form of a `map` element, containing a set of zero or more `layer` elements. A Web map represents a region of the earth's surface in two dimensions. A map's initial location (`lat`, `lon`), extent (`width` and `height`) and scale (`zoom`) are defined by its attributes.

EXAMPLE 1

Consider a map of Paris:



This map can be created with the following markup:

```
<map zoom="11" lat="48.85591" lon="2.3469543" width="640" height="300" controls
   <layer label="OpenStreetMap" src="https://example.com/mapml/osm/" checked cr
</map>
```

A Web map can have zero or more layers. Map layers are drawn in document order of the `layer` elements, following the painters model. If the user (or author) enables more than one layer, all the layers will be painted, in document order. Features in layers drawn earlier will be painted over by later layers. Opacity of layers can be managed with CSS, if necessary. [SVG2] [css-color-3]

EXAMPLE 2

Here is a map of Ottawa, with two layers:

This map can be created with the following markup:

```
 <style>
   .transparency {
     opacity: 0.2;
   }
 </style>
<map zoom="15" lat="45.398043" lon="-75.70683" width="640" height="300" projecti
  <layer label="Canada Base Map" src="https://example.com/mapml/cbmt/" checked
  <layer label="CanVec+ 031G" src="https://example.com/mapml/canvec/50k/features
</map>
```

4.3 Syntax §

The `map` element has a different content model defined by this specification, compared to that defined by the *HTML Standard*.

At such a moment as is warranted, the standard definition of the [HTML] `map` and `area` elements will be merged into this specification, but it is probably confusing to merge the definitions at this point in time. The intention is, however, that the existing standard map element behaviour and markup can be used by authors as a widely supported fallback mechanism for non-compliant browsers as this

specification is incrementally implemented by the platform browsers.

5. Semantics, Structure and Styling of MapML Documents §

5.1 Semantics §

5.1.1 Coordinate Systems §

In this document, all ***Coordinate Systems*** (CS) are 2D. A 2D CS has two axes, one horizontal (commonly related to west-east direction), and one vertical (commonly related to north-south direction; do not confuse it with "elevation"). This document defines the axis order of coordinate pairs serialized in MapML documents which the axis-specific values are not distinguished by markup, such as within the `coordinates` element. In documents conforming to this specification, we override any externally defined axis order. Coordinates are serialized as *horizontal axis followed by vertical axis*, separated by whitespace. Where applicable, coordinate pairs are separated by whitespace. The axis names of a coordinate system are identified by this document.

5.1.2 Coordinate Reference Systems §

A ***Coordinate Reference System*** (CRS) is a Coordinate System that is referenced to locations on the Earth. Maps are graphics which follow mathematical rules (called projections) established to transform locations on Earth to locations on a plane (formerly paper, more recently a device screen). Projections are mathematical equations designed to conserve particular properties of the source location, and often have parameters that can be used to induce desired properties for particular locations.

> NOTE
>
> To facilitate the sharing of CRS definitions, the *International Association of Oil ans Gas Producers* (IOGP) (formerly known as the *European Petroleum Survey Group* (EPSG)) mantains a list of CRS definitions and codes in their *EPSG Geodetic Parameter Dataset*. Some EPSG codes are used in the following table. In an effort to bring some clarity about the axis order for the longitude-latitude based CRSs, the OGC defined some "CRS codes". CRS:83 is defined equivalent to EPSG:4269 but with horizontal-then-vertical axis order, and CRS:84 is defined equivalent to EPSG:4326 except in its axes order. [EPSG-REGISTRY]

5.1.3 Tiled Coordinate Reference Systems §

A ***Tiled Coordinate Reference System*** is a set of Coordinate Reference Systems, which share a common origin in space, but which differ by the size of their smallest unit at defined locations, e.g. meters/pixel along the equator.

This document defines the following coordinate reference system types:

Coordinate Reference System type codes and descriptions §

Code	Name	Description
`tcrs`	The root coordinate system(s) in a Tiled CRS	A set of CRS that, for each of a member of a set of resolutions, associates pixel coordinates with an origin at the upper left corner of the pixel space with axis values increasing right (x, horizontal) and downwards (y, vertical). The pixel size in the associated PCRS units is a function

Code	Name	Description
		of the associated resolution value.
`gcrs`	Geodetic CRS	A CRS that models the Earth as an ellipsoid or a sphere and expresses locations in longitude and latitude. The origin is at the intersection point of the prime meridian of Greenwich (longitude, horizontal axis) and the equator (latitude, vertical axis).
`pcrs`	Projected CRS	A CRS that transforms the GCRS space into a planar system using a projection method, and

Code	Name	Description
		expresses locations in Easting (horizontal) and Northing (vertical) coordinates. The PCRS is generated on top of a GCRS and both together are commonly identified with a CRS code (EPSG codes being the most common codes).
`tilematrix`	Tile matrix CRS	In a TCRS, for each resolution, a tilematrix coordinate system groups underlying TCRS pixels into square tiles and counts tiles with the origin at

Code	Name	Description
		the upper left corner of the tiled space and increasing right (column axis, horizontal) and downwards (row axis, vertical) respectively.
`tile`	Tile CRS	A CRS associated to each tile in a tilematrix, measured in pixels with the origin at the upper left corner of the tile and increasing right (i axis, horizontal) and downwards (j axis, vertical), respectively.
`map`	Map CRS	A CRS that is

Code	Name	Description
		associated to the state of the map (the actual view port in the computer screen) measured in pixels, with the origin at the upper left corner of the map and increasing (i axis, horizontal) and downwards (j axis, vertical), respectively.

MapML documents are similar to other spatial information with regard to definition of projection and coordinate system requirements. However, MapML defines a system of communicating coordinate system information across the uniform interface of HTTP in order to eliminate assumptions in shared coordinate systems: the coordinate systems requested by the client and available from the server are represented as simple defined string identifiers defined in the table below, and exchanged in markup defined by this specification.

Tiled Coordinate Reference Systems §

TCRS identifier	Description	PCRS	GCRS*	Projection	Origin (x,y)	T

TCRS identifier	Description	PCRS	GCRS*	Projection	Origin (x,y)	T
OSMTILE	Web Mercator-based tiled coordinate reference system. Applied by many global map applications, for areas excluding polar latitudes.	EPSG::3857 / WGS 84 - Pseudo-Mercator	CRS:84	Spherical Mercator	-20037508.342787, 20037508.342787	
CBMTILE	Lambert Conformal Conic-based tiled coordinate reference system for Canada.	EPSG::3978 / NAD83 - Canada Atlas Lambert	CRS:83	Lambert Conic Conformal (2SP)	-34655800, 39310000	

TCRS identifier	Description	PCRS	GCRS*	Projection	Origin (x,y)	T
APSTILE	Alaska Polar Stereographic-based tiled coordinate reference system for the Arctic region.	EPSG::5936 / WGS 84 - EPSG Alaska Polar Stereographic	CRS:84	Polar Stereographic (variant A)	-28567784.109255, 32567784.109255	
WGS84	World Geodetic System 1984.	CRS:84	CRS:84	Pseudo Plate carrée	LatLng(90,-180)	

TCRS identifier	Description	PCRS	GCRS*	Projection	Origin (x,y)	T

* All coordinate reference systems' coordinate pairs, where not explicitly marked up or identified by axis (for example, in the `coordinates` element) are defined by this specification to be in "horizontal axis followed by vertical axis" order. Where axes order is defined differently by external definition, this specification *MUST* be taken to be correct **for MapML documents**.

OSMTILE §

This specification defines the string "OSMTILE" to be the identifier of a tiled coordinate reference system projected (in the Web Mercator system) and scaled into 19 zoom levels (0-18) at defined resolutions. The OSMTILE coordinate reference system has become a de facto interoperable standard for 'slippy' Web maps. The OSMTILE coordinate reference system is suitable for small scale mapping of the Earth, exclusive of north and south polar latitudes, where distortion is extreme.

Some of the major defining characteristics of the OSMTILE coordinate reference system include the fact that a large portion of the surface of the earth is represented by a single 256px by 256px tile (image) at zoom level 0, with successive zoom levels' tiles dividing the area represented by the parent tile into four equal quadrants, and 'nesting' perfectly within the parent tile such that each successive level of tiles contains 2^{zoom} individual tiles. The geo-registration of tiles at all zoom levels is defined by the coordinate system definition, and they can thus be 'mashed up' with other spatial data in a defined manner.

CBMTILE §

The "CBMTILE" tiled coordinate reference system is suitable for small to medium scale mapping of Canada.

APSTILE §

The "APSTILE" tiled coordinate reference system is based on the Stereographic family of map projections, with the North polar aspect, per the EPSG::5936 coordinate reference system. This projection system is suitable for displaying areas in (north) polar latitudes.

WGS84 §

This specification defines the string "WGS84" to be the identifier of a tiled coordinate reference system projected in the Equirectangular, or *Plate carrée* system.

The WGS84 tiled coordinate reference system is suitable for small to medium scale mapping of the entire world. It is also useful for vector data provided that the inter-vertex distance is no less than a single pixel in size at any particular zoom level.

> EDITOR'S NOTE
>
> It is believed that the definition of WGS84 according to this document makes it a suitable choice for non-georeferenced mapping applications, such as ultra high resolution graphics and technical drawings are one of the their use cases. The editor of this document requests specific implementations to contribute implementation experience in using this TCRS for technical and other drawings.

Other coordinate reference systems §

Many other projection systems exist, and could in principle be used by map clients and servers. As such projections and associated defined tiled coordinate reference systems come into common usage, it is expected that their definitions for use in MapML and on the Web will be imported here. The contents of this document constitutes a registry of such projections.

5.1.4 Scale / Resolution / Zoom levels §

Scale is an essential characteristic of all geospatial information. The scale at which information is used

dictates the model used for its representation. Geospatial information cannot be meaningfully combined without consideration given to its scale. Entire cities are represented on 'small scale' maps as dots, whereas on another large scale map, a single city could have many thousands of individual features in its depiction.

Not only is scale important in the portrayal of maps, it is also essential when defining the model of underlying feature data.

Map projections distort the surface of the earth in ways which suit the objectives of the projection definition. As a result of that distortion, the scale and the resolution of maps in that projection vary as a function of location. The resolutions specified by this document are only valid in defined locations. For example, in WGS84 and OSMTILE the defined resolutions are valid only along the equator. "Zoom" levels are integer values corresponding to the index of the resolution value defined above, and represent numeric proxy for a scale, at the defined location(s). For example, the OSMTILE TCRS, being based internally on the Spherical Web Mercator projection (EPSG::3857), allows us to reasonably portray a large portion of the Earth's surface, requiring relatively simple and fast math to convert to and from spherical latitude/longitude coordinates. Additionally, it enables the simple tiling system known as OSMTILE in this specification, that has become a *de facto* standard. As such, it was deemed to have appropriate properties for a globally useful projection on the Web.

In MapML documents, scale is indirectly represented by zoom level, and varies across any rendered map extent. Two MapML documents in the same TCRS and zoom level should be interoperable to the degree that they can be automatically and visually related (overlayed). The zoom level of a MapML document is often represented by the value of the `value` attribute of the `input[@type=zoom]`. By associating a specific zoom `value` to the input, we are able to compute the spatial bounds of a `link` template by means of `min`, `max` and `axis` values of other `input[@type=location]` elements associated to the same `link` template element with which the `input[@type=zoom]` is associated. In any case, the zoom level of any MapML document *MAY* be present in document metadata as a `meta` element e.g. <meta name="zoom" content="0"/>. In the case where a MapML document contains an `extent` element, an `input@type=zoom` *MUST* be present which controls how the zoom is transmitted from client to server. Where that input has a non-empty `input@value` attribute, in case of discrepancy between the `meta[@name=zoom]/@content` and the `input[@type=zoom]/@value`, the latter zoom value *SHALL* be taken to be correct, EXCEPT in the case where there may exist more than one `input[@type=zoom]` and their `value` attributes do not agree, in which case the value of `meta[@name=zoom]/@content` shall be taken to be correct.

5.1.5 Extents §

A MapML document is a representation of a defined portion of a two dimensional map area. In

MapML, this is called an 'extent' (see below), the extremes of which can be described in terms of locations coordinates specified in the TCRS (pcrs, gcrs, tilematrix, etc.). The `extent/@units` value *SHALL* be taken to be the authoritative declaration of the TCRS for a MapML document.

5.1.6 Fragment identifiers §

> EDITOR'S NOTE
>
> TBD

5.2 Structure §

5.2.1 The `Document` object §

A MapML document is a [MicroXML] document. As such, it *MUST* have a single root element named `mapml`. It *MUST NOT* have a `DOCTYPE` declaration; attributes named `xmlns` and names containing a colon character ':' (U+003A) are not permitted. Other restrictions relative to [XML] syntax are enumerated in [MicroXML].

5.2.2 The `<mapml>` element §

> **Categories**:
>
> N/A.
>
> **Contexts in which this element can be used**:
>
> The root of a MapML document.
>
> **Content model**:
>
> One `head` element, followed by one `body` element.
>
> **Content attributes**:
>
> `lang` — the language of the document, expressed as a BCP 47 language tag. [BCP47]
>
> **DOM interface**:
>
> **WebIDL**

```
[Exposed=Window]
interface MAPMLMapMLElement : MAPMLElement {
  [HTMLConstructor] constructor();
};
```

The mapml element is the the root element of a MapML document, and must contain one head element followed by one body element.

The mapml element may carry a lang attribute, as defined by [HTML].

5.2.3 The <head> element §

Categories:

Metadata content.

Contexts in which this element can be used:

As the first child element of the mapml element.

Content model:

One or more elements of metadata content, of which exactly one is a title element and no more than one is a base element.

Content attributes:
DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLHeadElement : MAPMLElement {
  [HTMLConstructor] constructor();
};
```

The head element is for MapML document metadata content. The content may include: one title element, one base element, zero or more link elements, and zero or more meta elements.

5.2.4 The <title> element §

Categories:

Metadata content.

Contexts in which this element can be used:

As a child of the `head` element.

Content model:

Text.

Content attributes:
DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLTitleElement : MAPMLElement {
  [HTMLConstructor] constructor();
};
```

The `title` element should exist as the one and only `title` element in the `head` element. Its content should be a text string describing the document. It is conceivably used as a bookmark title.

5.2.5 The `<base>` element §

Categories:

Metadata content.

Contexts in which this element can be used:

As a child element of the `head` element.

Content model:

Nothing.

Content attributes:

`href` — The absolute URL to be used to resolve relative URLs in the document.

DOM interface:

WebIDL

```
[Exposed=Window]
```

```
interface MAPMLBaseElement : MAPMLElement {
  [HTMLConstructor] constructor();
};
```

The `base` element is used to identify a URL to be used to act as a base URL in order to resolve relative URLs later in the document.

There must be only one `base` element in a MapML document, and it must be in the content of the `head` element, before any MapML elements which potentially carry a URL for resolution, notably the `link` element.

5.2.6 The `<meta>` element §

Categories:

Metadata content.

Contexts in which this element can be used:

In the `head` element.

Content model:

Nothing.

Content attributes:

`name` — Metadata name.
`http-equiv` — Pragma directive.
`content` — Value of the element.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLMetaElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString name;
  attribute DOMString httpEquiv;
  attribute DOMString content;
};
```

5.2.7 The `<link>` element §

Categories:

Metadata content.

Contexts in which this element can be used:

If the element represents a hyperlink (has a `href` attribute): as a child of the `head` or `body` element.
If the element represents a link template (has a `tref` attribute): as a child of the `extent` element.

Content model:

Nothing.

Content attributes:

`href` — Address of the hyperlink.
`tref` — URL Template.
`rel` — Relationship between the document containing the element and the destination resource.
`projection` — The TCRS of the linked resource.
`hreflang` — Language of the linked resource.
`type` — Hint for the type of the referenced resource.
`title` — Title of the link.

DOM interface:

```
[Exposed=Window]
interface MAPMLLinkElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString href;
  attribute DOMString tref;
  attribute DOMString rel;
  attribute DOMString projection;
  attribute DOMString hreflang;
  attribute DOMString type;
  attribute DOMString title;
};
MAPMLLinkElement implements LinkStyle;
```

The `link` element allows authors to link their document to other resources.

The destination of the link(s) is given by the ***`href`*** attribute, which must be present and must contain a valid non-empty URL potentially surrounded by spaces. If the `href` attribute is absent, then the element does not define a link.

A `link` element must have a `rel` attribute. If the `rel` attribute is absent, then the element does not define a link.

The types of link are given by the value of the ***`rel`*** attribute, which generally has a value that is a single token, except in the case of the link@rel='self style' value described below. The allowed keywords and their meanings are defined below. If the `rel` attribute is absent, has no keywords, or if none of the keywords used are allowed according to the definitions in this specification, then the

element does not create any links.

The `projection` attribute value identifies the TCRS of the linked resource, with a value from the defined set of TCRS identifiers. When `rel=alternate` is used together with the `projection` attribute, clients may be able to perform agent-driven content negotiation to provide a better user experience. For example, if an HTML author mistakenly enters the URL of an OSMTILE resource in their HTML `layer@src` attribute, but the `map` in which the layer takes part is declared to be `CBMTILE`, the MapML author can ease the potential for resultant confusion by providing appropriate `rel=alternate` links to equivalent MapML resources in other projections.

The `link/@title` value can be used in conjunction with `link[@rel=license]` as a string description of the link to the terms under which the MapML content is made available, while the `link/@href` value is used to link to an HTML document of the license terms. Note that there may be more than a single contributor to the information presented in a single extent, and it is therefore possible to have several licenses terms for a single extent, each of which will have its own `link` element in a single MapML document.

The `link@rel='self style'` rel value can be used to designate one of a set of alternative named style links as being the style of the current document. As such, the value of the `link@title` attribute will be used by the user agent to label the current style from among the list of alternate styles.

The `link[@tref]` element represents a URL template for a resource (e.g. tiles) which can be used by the client make requests for resources to fill its extent at zoom levels and bounds indicated by sibling `input[/@type=location]` elements' `min` and `max` attributes, or to (partially) identify URL targets suitable to enable query of server resources. The `rel` attribute contains a keyword string which identifies the role of the resource(s) for which the `tref` attribute value represents a URL template, which may contain zero or more `input/@name` variable references. Such variable references must be a case-insensitive match of sibling `input` elements' `name` attribute.

The content represented by `link[@tref]` elements should be painted in document order of the appearance of the `link[@tref]` elements.

`@rel` values §

Link type	Brief description
`alternate`	Gives alternate representations of the current document.

Link type	Brief description
`legend`	Indicates the link to a legend resource.
`license`	Indicates that some or all of the content of the current document is covered by the copyright license described by the referenced document.
`stylesheet`	Imports a stylesheet.
`style`	Indicates a map resource that is equivalent to the current resource, in the same TCRS but with a different map style.
`self`	Indicates that the linked resource is the current resource. When used in combination with the style

Link type	Brief description
	rel value, indicates that the linked resource is the current selected style from a group of alternative styles for the current layer. There should only be one such link that has both rel values 'self' and 'style' in a document.
`zoomin`	Indicates a resource that may be used at a zoom level greater than the maximum zoom of the current extent.
`zoomout`	Indicates a resource that may be used at a zoom level less than the minimum zoom of the current extent.
`tile`	The templated resource

Link type	Brief description
	reference identifies a tile
`image`	The templated resource reference identifies an image that covers the map extent
`features`	The templated resource reference identifies a feature collection
`query`	The templated resource reference identifies a query resource at a location

5.2.8 The `<body>` element §

> **Categories**:
>
> N/A.
>
> **Contexts in which this element can be used**:
>
> As the second child of the `mapml` element.
>
> **Content model**:
>
> Features and metadata.
>
> **Content attributes**:

N/A.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLBodyElement : MAPMLElement {
  [HTMLConstructor] constructor();
};
```

The `body` element represents the content of the document.

5.2.9 The `<extent>` element §

Categories:

Metadata content.

Contexts in which this element can be used:

Required to be a child of the `body` element.

Content model:

A set of multiple `input` and one or more `link` elements with their `rel` attribute in either the "`tile`", "`image`" or "`features`" state, and zero or one `link` element with its `rel` attribute in the "`query`" state.

Content attributes:

`units` — The name of the coordinate reference system whose measurement units are to be used by values submitted by child `input` elements.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLExtentElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString units;
  readonly attribute MAPMLExtentControlsCollection elements;
  readonly attribute long length;
```

```
    getter Element (unsigned long index);
    getter (RadioNodeList or Element) (DOMString name);
  };
```

The `extent` element is a map-associated affordance, which contains input and link elements, whose job it is to serialize location event properties that can be submitted to a server for processing.

The `units` attribute indicates the parent TCRS that location events shall be generated for, and serialized as requested by the `extent`'s contents.

5.2.10 The `<input>` element §

Categories:

N/A.

Contexts in which this element can be used:

Required to be a child of the `extent` element.

Content model:

Nothing.

Content attributes:

`type` — Enumerated keyword string identifies the type of input.
`name` — The token to be used as a variable name in form serialization.
`value` — The current or default value of the name/value pair represented by the `input`.
`shard` — A boolean attribute indicating associated `datalist` values be used to shard map tile requests across subdomains (the values).
`list` — The id of an associated `datalist` element to supply values for the input. Useful only with `shard` at this time.
`min` — The minimum value for the extent parameter acceptable by the server.
`max` — The maximum value for the extent parameter acceptable by the server.
`step` — The increment by which a value may vary between the `min` and `max` values.
`units` — When `type` attribute is `location`, enumerated keyword string identifies the associated coordinate system for this input.
`axis` — When `type` attribute is `location`, enumerated keyword string identifies the axis of the associated OR related coordinate system from the `units` attribute, for which an associated axis value will be returned. The `min` and `max` attribute values, if present, will be interpreted in the defined units of this axis.

`rel` — When `type` attribute is `location`, `tile` or `image` (the default if not specified) identifies the relation of this location to either a tile or an image corresponding to the map extent in which the location is found. Establishes the meaning of the `position` attribute to identify the location relative to the tile in question or to the map extent (the default).
`position` — When `type` attribute is `location`, enumerated keyword string identifies the relative position of a location.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLInputElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString type;
  attribute DOMString name;
  attribute boolean shard;
  readonly attribute MAPMLElement? list;
  attribute DOMString min;
  attribute DOMString max;
  attribute DOMString step;
  attribute DOMString units;
  attribute DOMString axis;
  attribute DOMString rel;
  attribute DOMString position;
  attribute DOMString value;
};
```

The `input` element represents a typed data field, associated with a map extent form, to allow the user agent to request documents for a specific map area.

The `type` attribute controls the data type (and associated control) of the element. It is an enumerated attribute. The following table lists the keywords and states for the attribute — the keywords in the left column map to the states in the cell in the second column on the same row as the keyword.

`input@type` values §

Keyword	State	Data type	Control type

Keyword	State	Data type	Control type
`zoom`	zoom	An integer value	A derived property of a displayed map
`location`	location	A string value with no line breaks	A location, the units and axis of which are specified by sibling attributes.
`width`	width of the extent	A string value with no line breaks	.
`height`	height of the extent	A string value with no line breaks	.
`hidden`	hidden	An arbitrary string	n/a

When the `type` attribute is in the "location" state, the `input` may have three associated attributes: `units`, `axis` and `position`. `units` identifies the associated coordinate system that the `location` is referred to. Each Tiled Coordinate Reference System (TCRS) instance may have one or more associated coordinate systems that are coordinate reference system by virtue of their defined association to the TCRS. For instance, the OSMTILE TCRS is associated to the underlying coordinate reference system known commonly as Web Mercator, by virtue of an origin location, defined in meters in the EPSG:3857 CRS plus a set of zoom level resolutions, and a transformation.

`input@units` values §

Keyword	State	OGC equivalent
`tcrs`	For each zoom level, locations are expressed in pixel coordinates with the origin at the top-left corner of the pixel space.	OGC Tile Matrix Set "tile matrix (i',j')".
`pcrs`	The location is expressed in projected coordinates. Units are meters except for WGS84 where pcrs AND gcrs coordinates are in longitude-latitude. E.g. meters for OSMTILE, which has an associated	Commonly represented as (x,y)

Keyword	State	OGC equivalent
	projected coordinate reference system of EPSG:3857.	
`gcrs`	The location is expressed in the geodetic coordinate system associated to the TCRS, e.g. decimal degrees for the OSMTILE TCRS.	Commonly represented as (long,lat)
`map`	The map coordinate system is defined by the state of the map system. At each pan and / or zoom, the map coordinate origin is reset within the Location is expressed in	WMS GetFeatureInfo (i,j)

Keyword	State	OGC equivalent
	pixel coordinates with the origin starting at the upper left corner of the extent increasing right and downwards, respectively.	
`tilematrix`	For each zoom level, locations are expressed in tile row and column index (0-based) values, with the origin at the top-left corner of the tiled space, with column index values increasing right and row index values increasing downwards.	WMTS GetTile (TileCol, TileRow).

Keyword	State	OGC equivalent
`tile`	For each tile within every zoom level, a location is expressed in pixel coordinates with the origin at the upper left corner of the tile increasing right and downwards, respectively and ending at 256. The combination of tilematrix and tile coordinates yields a location reference with pixel-sized precision.	WMTS GetFeatureInfo (i,j)

The meaning of the `position` attribute value (keyword) depends upon the presence and value of the associated `rel` attribute. When the `rel` attribute is not present or has the value `image`, the `position` attribute keyword value describes the input location relative to the ancestor `extent`. When `rel=tile` (only applicable when the `units` value equals `tilematrix`), the `position` attribute values describe the input location relative to the tile at the location in question.

`input@position` values §

Keyword	State
`top-left`	Identifies a location relative to a tile or extent.
`top-right`	Identifies a location relative to a tile or extent.
`bottom-left`	Identifies a location relative to a tile or extent.
`bottom-right`	Identifies a location relative to a tile or extent.
`center-left`	Identifies a location relative to a tile or extent.

Keyword	State
`center-right`	Identifies a location relative to a tile or extent.
`top-center`	Identifies a location relative to a tile or extent.
`bottom-center`	Identifies a location relative to a tile or extent.
`center`	Identifies a location relative to a tile or extent.

The `axis` keyword identifies the axis of the coordinate that the `input` variable represents. The coordinate system should be identified by the `units` attribute.

`input`@`axis` values §

Keyword	State
`x`	TCRS x axis.

Keyword	State
`y`	TCRS y axis.
`row`	TileMatrix row axis (parallel to y axis of TCRS).
`column`	TileMatrix column axis (parallel to x axis of TCRS).
`i`	Map or tile CS i axis (parallel to x axis of TCRS).
`j`	Map or tile CS j axis (parallel to y axis of TCRS).
`easting`	Projected coordinate reference system axis (parallel to x axis of TCRS).

Keyword	State
`northing`	Projected coordinate reference system axis (parallel to y axis of TCRS).
`latitude`	Geodetic coordinate reference system axis (parallel to y axis of TCRS).
`longitude`	Geodetic coordinate reference system axis (parallel to x axis of TCRS).

5.2.11 The `<datalist>` element §

> **Categories**:
>
> N/A.
>
> **Contexts in which this element can be used**:
>
> As a child of the `extent` element, associated to an `input` element via that element's `list` attribute.
>
> **Content model**:

One or more `option` elements.

Content attributes:

`id` — Identifies the list within the current document.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLDatalistElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString id;
};
```

5.2.12 The `<label>` element §

Categories:

N/A.

Contexts in which this element can be used:

As a child of the `extent` element.

Content model:

Phrasing content.

Content attributes:

`for` — Identifies by id reference the select for which the content of this element is the label.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLLabelElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString for;
};
```

5.2.13 The `<select>` element §

Categories:

N/A.

Contexts in which this element can be used:

As a child of the `extent` element.

Content model:

One or more `option` elements.

Content attributes:

`id` — Identifies the select within the current document.
`name` — The token to be used as a variable name in form serialization.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLSelectElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString id;
  attribute DOMString name;
};
```

5.2.14 The `<option>` element §

Categories:

N/A.

Contexts in which this element can be used:

As a child of the `select` element or the `datalist` element.

Content model:

Nothing.

Content attributes:

`value` — The value to be used in form submission.
`label` — The label to be presented to the user, if applicable.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLOptionElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString value;
  attribute DOMString label;
};
```

5.2.15 The `<tile>` element §

Categories:

Feature data.

Contexts in which this element can be used:

Child of the `body` element.

Content model:

Nothing.

Content attributes:

`row` — An integer (a Y axis value / the tile size) in the range of the tile coordinate reference system.
`col` — An integer (a X axis value / the tile size) in the domain of the tile coordinate reference system.
`src` — A URL from which the tile may be obtained.
`type` — A hint about the MIME type of the resource obtainable at the `src` URL.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLTileElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute long row;
  attribute long col;
  attribute DOMString src;
  attribute DOMString type;
};
```

The `tile` element is a type of feature which is associated with a tiled coordinate reference system that subdivides or 'tiles' 2D space in a recursively repeating grid pattern, where the origin of both the tiled coordinate reference system and the grid at all zoom levels is defined in coordinates of an underlying projected coordinate reference system.

A defining characteristic of tiled coordinate reference systems is that they rely on integer grid row/col coordinates and zoom values. The use of these values in URLs can yield highly cacheable resources, which can lead to high-performance map services.

The "zoom" value is a global integer property of a MapML document whose coordinate system is defined by this specification. All MapML documents have a defined zoom value. The "zoom" value is equal to the `zoom@value` child of the `extent` element. Hence the zoom value is not a direct attribute of the `tile` element.

The main example of such a tiled coordinate reference system is `OSMTILE`, although others exist.

5.2.16 The `<image>` element §

Categories:

Feature data.

Contexts in which this element can be used:

A child of the `feature` element, which has a sibling `bbox` element.

Content model:

Nothing.

Content attributes:

`src` — Address of the hyperlink.
`type` — A hint as to the MIME type of the resource.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLImageElement : MAPMLElement {
  [HTMLConstructor] constructor();
};
```

5.2.17 The `<feature>` element §

Categories:

Feature data.

Contexts in which this element can be used:

Child of the `body` element.

Content model:

An optional `geometry` element and an optional `properties` element.

Content attributes:

Global attributes

`zoom` — the 'native' zoom level of the feature geometry.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLFeatureElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString zoom;
};
```

A `feature` element represents a geographic feature. A `feature` element has an optional `properties` child element, and a required child `geometry` element.

5.2.18 The `<properties>` element §

Categories:

Feature data.

Contexts in which this element can be used:

A child of the `feature` element, containing elements representing the properties of the feature.

Content model:

One or more unknown elements with text values.

> EDITOR'S NOTE
> TODO: Allow HTML content.

Content attributes:

N/A.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLPropertiesElement : MAPMLElement {
  [HTMLConstructor] constructor();
};
```

A `feature` element can have zero or one `properties` element, which contains zero or more unknown elements, whose content is text.

5.2.19 The `<geometry>` element §

Categories:

Feature data.

Contexts in which this element can be used:

Child of the `feature` element.

Content model:

A single geometry value, described in the table below.

Content attributes:

Global attributes.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLGeometryElement : MAPMLElement {
  [HTMLConstructor] constructor();

  readonly attribute DOMString type;
};
```

A `geometry` element has one child element, which can be a `point`, `linestring`, `polygon`, `multipoint`, `multilinestring`, `multipolygon`, or `geometrycollection`.

`geometry` value	Content model	Non-schema constraints
`point`	A `coordinates` element containing a single position.	Axis order - x followed by y, separated by whitespace.
`linestring`	A `coordinates` element containing two or more positions.	Axis order - x followed by y, separated by whitespace.
`polygon`	One or more `coordinates` elements, each containing three or more positions.	Axis order - x followed by y, separated by whitespace. The first and last positions in every child `coordinates` element are equal / at the same position. The first `coordinates` element represents the outside of the polygon, and subsequent `coordinates` elements represent holes. The "winding order" of positions in child `coordinates` should depend on the axis orientation of the coordinate reference system in use, and whether the

`geometry` value	Content model	Non-schema constraints
		`coordinates` element represents the exterior of a polygon, or a hole. For WGS84, the exterior should be counterclockwise and holes should be clockwise.
`multipoint`	One `coordinates` element, containing one or more positions.	Axis order - x followed by y, separated by whitespace.
`multilinestring`	One or more `coordinates` elements, each containing two or more positions.	Axis order - x followed by y, separated by whitespace.
`multipolygon`	One or more `polygon` elements.	Axis order - x followed by y, separated by whitespace. For each member polygon, the same non-schema constraints apply to multipolygon descendant `coordinates` elements, as for polygon `coordinates` descendant elements.

`geometry` value	Content model	Non-schema constraints
`geometrycollection`	One or more `point`, `linestring`, `polygon`, `multipoint`, `multilinestring`, `multipolygon` elements.	For each member geometry, the same non-schema constraints apply as to the unique geometry type above.

5.2.20 The `<coordinates>` element §

Categories:

Feature data.

Contexts in which this element can be used:

A descendant of the `geometry` element, as the coordinate data which defines a feature geometry.

Content model:

One or more positions in x followed by y, separated by whitespace, with each position separated from the adjacent position by whitespace.

Content attributes:

N/A.

DOM interface:

WebIDL

```
[Exposed=Window]
interface MAPMLCoordinatesElement : MAPMLElement {
  [HTMLConstructor] constructor();
};
```

In MapML, features have positions in 2D space. A position is sometimes marked up with coordinate data explicitly identified by axis, for example in the `tile` element, the attributes `row` and `col` are used.

Equally common is coordinate data which omits explicit markup of positions' axes, for example in the `coordinates` element. In the `coordinates` element, positions must be encoded in `x`,`y` order, separated by whitespace. The `coordinates` element is used to model the content of various geometries in MapML. Its content and the meaning of its content it depends on what geometry value it is used in. The context and content of the `coordinates` element is described in the `geometry` element table. A RelaxNG schema for MapML is provided below, however schema validation is not able to fully validate the context and content of the `coordinates` element. Such requirements are listed under the column "Non-schema constraints"

The positions in the `coordinates` element define the location of the feature. For `geometry` elements whose value is `linestring`, `multipoint` or `multilinestring`, this specification neither assigns nor requires special ordering or other constraints, apart from the axis order described above. For `geometry` elements whose value is a `polygon` or `multipolygon` element, the descendant `coordinates` elements must follow additional constraints.

5.3 Styling §

MapML documents can be styled with Cascading Style Sheets.

> ISSUE 71: How does styling MapML with CSS work?

5.4 Examples §

This section is non-normative.

EXAMPLE 3

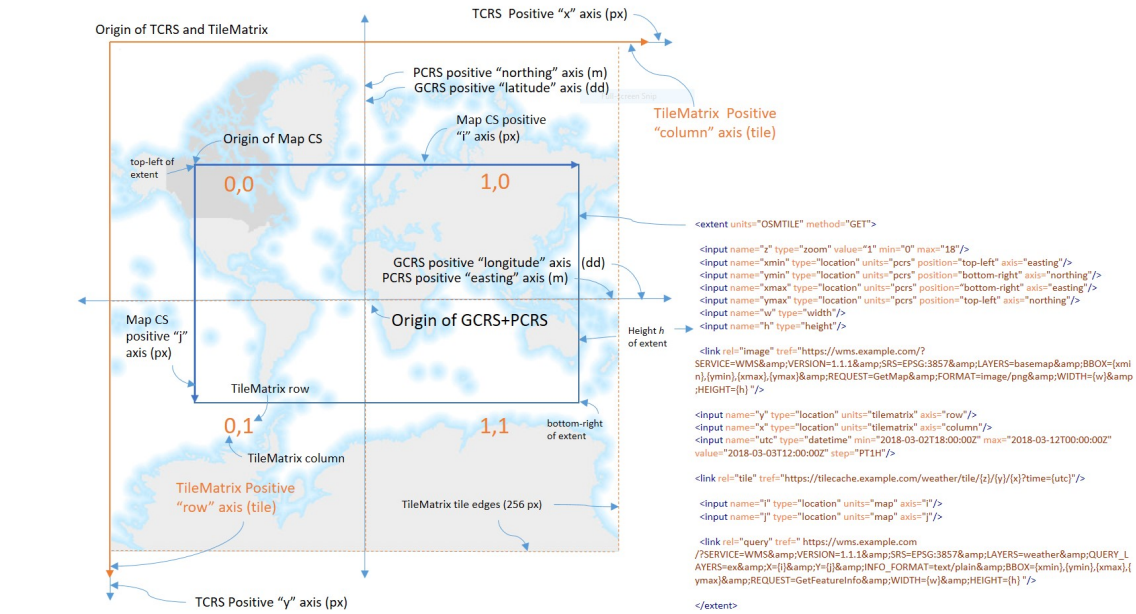


Figure 1 Prototype MapML services are available, representing tiled and vector data.

6. Security Considerations §

EDITOR'S NOTE

This technology relies on web resources loaded over the internet, which carries security implications. The architecture is not unlike allowing embedded images encoded in SVG format, and it is thought that similar security considerations apply. The author would be grateful if commenters could highlight such implications, so that they could be explicitly made part or the considerations of this draft.

7. Glossary of Terms and Datatypes §

> EDITOR'S NOTE
>
> TODO: Provide a glossary of terms and datatypes used in this specification.

8. Schema §

A schema is useful for machine processing of documents, be it document creation, transformation, or validation.

RelaxNG Schema §

Map Markup Language provides a schema written in RelaxNG compact syntax [RELAXNG-SCHEMA], a namespace-aware schema language that uses the datatypes from XML Schema Part 2 [xmlschema-2].

Further map document validation can be done with Schematron. Please see the MapML schema directory for further information. [MAPML-SCHEMA]

Unlike a DTD, the schema used for validation is not hardcoded into the document instance. There is no equivalent to the `DOCTYPE` declaration. Simply point your editor or other validation tools to the URL of the schema (or your local cached copy, as you prefer).

MapML schema (under development).

A. IDL Index §

WebIDL

```
[Exposed=Window, LegacyFactoryFunction=Map(unsigned long width, unsigned long
height, double lat, double lon, unsigned short zoom, optional DOMString
projection, boolean controls)]
interface HTMLMapElement : HTMLElement {
  [HTMLConstructor] constructor();

  [SameObject] readonly attribute HTMLCollection layers;
  [SameObject] readonly attribute HTMLCollection areas;
  [CEReactions] attribute unsigned short zoom;
  [CEReactions] attribute double lat;
```

```
  [CEReactions] attribute double lon;
  readonly attribute DOMString projection;
  [CEReactions] attribute boolean controls;
  [SameObject, PutForwards=value] readonly attribute
      DOMTokenList controlsList;
  [CEReactions] attribute unsigned long width;
  [CEReactions] attribute unsigned long height;
  undefined zoomTo(double latitude, double longitude, optional unsigned short
zoom);
};


[Exposed=Window]
interface HTMLAreaElement : HTMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString href;
  attribute DOMString alt;
  attribute DOMString coords;
  attribute DOMString shape;
};


[Exposed=Window]
interface MAPMLMapMLElement : MAPMLElement {
  [HTMLConstructor] constructor();
};


[Exposed=Window]
interface MAPMLHeadElement : MAPMLElement {
  [HTMLConstructor] constructor();
};


[Exposed=Window]
interface MAPMLTitleElement : MAPMLElement {
  [HTMLConstructor] constructor();
};


[Exposed=Window]
interface MAPMLBaseElement : MAPMLElement {
```

```
  [HTMLConstructor] constructor();
};


[Exposed=Window]
interface MAPMLMetaElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString name;
  attribute DOMString httpEquiv;
  attribute DOMString content;
};


[Exposed=Window]
interface MAPMLBodyElement : MAPMLElement {
  [HTMLConstructor] constructor();
};


[Exposed=Window]
interface MAPMLExtentElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString units;
  readonly attribute MAPMLExtentControlsCollection elements;
  readonly attribute long length;
  getter Element (unsigned long index);
  getter (RadioNodeList or Element) (DOMString name);
};


[Exposed=Window]
interface MAPMLInputElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString type;
  attribute DOMString name;
  attribute boolean shard;
  readonly attribute MAPMLElement? list;
  attribute DOMString min;
  attribute DOMString max;
  attribute DOMString step;
```

```
  attribute DOMString units;
  attribute DOMString axis;
  attribute DOMString rel;
  attribute DOMString position;
  attribute DOMString value;
};


[Exposed=Window]
interface MAPMLDatalistElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString id;
};


[Exposed=Window]
interface MAPMLLabelElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString for;
};


[Exposed=Window]
interface MAPMLSelectElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString id;
  attribute DOMString name;
};


[Exposed=Window]
interface MAPMLOptionElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString value;
  attribute DOMString label;
};


[Exposed=Window]
```

```
interface MAPMLTileElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute long row;
  attribute long col;
  attribute DOMString src;
  attribute DOMString type;
};


[Exposed=Window]
interface MAPMLImageElement : MAPMLElement {
  [HTMLConstructor] constructor();
};


[Exposed=Window]
interface MAPMLFeatureElement : MAPMLElement {
  [HTMLConstructor] constructor();

  attribute DOMString zoom;
};


[Exposed=Window]
interface MAPMLPropertiesElement : MAPMLElement {
  [HTMLConstructor] constructor();
};


[Exposed=Window]
interface MAPMLGeometryElement : MAPMLElement {
  [HTMLConstructor] constructor();

  readonly attribute DOMString type;
};


[Exposed=Window]
interface MAPMLCoordinatesElement : MAPMLElement {
  [HTMLConstructor] constructor();
};
```

B. Index §

B.1 Terms defined by this specification §

B.2 Terms defined by reference §

- [DOM] defines the following:
  - `DOMTokenList` interface
  - `Element` interface
  - `HTMLCollection` interface
- [EPSG-REGISTRY] defines the following:
  - EPSG Geodetic Parameter Dataset
- [HTML] defines the following:
  - `[CEReactions]` extended attribute
  - HTML Standard
  - `[HTMLConstructor]` extended attribute
  - `HTMLElement` interface
  - `RadioNodeList` interface
  - `Window` interface
- [MICROXML] defines the following:
  - MicroXML
- [RFC7946] defines the following:
  - The GeoJSON Format
- [WEBIDL] defines the following:
  - `boolean` type
  - `DOMString` interface
  - `double` type
  - `[Exposed]` extended attribute
  - `[LegacyFactoryFunction]` extended attribute
  - `long` type
  - `[PutForwards]` extended attribute
  - `[SameObject]` extended attribute
  - `undefined` type
  - `unsigned long` type
  - `unsigned short` type

C. References §

C.1 Normative references §

[BCP47]
Tags for Identifying Languages. A. Phillips; M. Davis. IETF. September 2009. IETF Best Current Practice. URL: https://tools.ietf.org/html/bcp47

[css-color-3]
CSS Color Module Level 3. Tantek Çelik; Chris Lilley; David Baron. W3C. 19 June 2018. W3C Recommendation. URL: https://www.w3.org/TR/css-color-3/

[css-images-3]
CSS Images Module Level 3. Tab Atkins Jr.; Elika Etemad; Lea Verou. W3C. 10 October 2019. W3C Candidate Recommendation. URL: https://www.w3.org/TR/css-images-3/

[dom]
DOM Standard. Anne van Kesteren. WHATWG. Living Standard. URL: https://dom.spec.whatwg.org/

[HTML]
HTML Standard. Anne van Kesteren; Domenic Denicola; Ian Hickson; Philip Jägenstedt; Simon Pieters. WHATWG. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[MAPML-SCHEMA]

MapML Schema. Peter Rushforth. W3C Maps for HTML Community Group. URL: https://github.com/Maps4HTML/MapML/tree/gh-pages/schema

[MicroXML]

MicroXML. J. CLark, J. Cowan, eds. World Wide Web Consortium MicroXML Community Group, work in progress, 2012. URL: https://dvcs.w3.org/hg/microxml/raw-file/tip/spec/microxml.html

[RELAXNG-SCHEMA]

Information technology -- Document Schema Definition Language (DSDL) -- Part 2: Regular-grammar-based validation -- RELAX NG. ISO/IEC. 2008. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c052348_ISO_IEC_19757-2_2008(E).zip

[RFC2119]

Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[RFC8174]

Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://tools.ietf.org/html/rfc8174

[SVG2]

Scalable Vector Graphics (SVG) 2. Amelia Bellamy-Royds; Bogdan Brinza; Chris Lilley; Dirk Schulze; David Storey; Eric Willigers. W3C. 4 October 2018. W3C Candidate Recommendation. URL: https://www.w3.org/TR/SVG2/

[webidl]

Web IDL. Boris Zbarsky. W3C. 15 December 2016. W3C Editor's Draft. URL: https://heycam.github.io/webidl/

[XML]

Extensible Markup Language (XML) 1.0 (Fifth Edition). Tim Bray; Jean Paoli; Michael Sperberg-McQueen; Eve Maler; François Yergeau et al. W3C. 26 November 2008. W3C Recommendation. URL: https://www.w3.org/TR/xml/

[xmlschema-2]

XML Schema Part 2: Datatypes Second Edition. Paul V. Biron; Ashok Malhotra. W3C. 28 October 2004. W3C Recommendation. URL: https://www.w3.org/TR/xmlschema-2/

C.2 Informative references §

[EPSG-REGISTRY]

EPSG Geodetic Parameter Dataset. International Association of Oil ans Gas Producers. URL: http://www.epsg-registry.org/

[rfc7946]

The GeoJSON Format. H. Butler; M. Daly; A. Doyle; S. Gillies; S. Hagen; T. Schaub. IETF. August 2016. Proposed Standard. URL: https://tools.ietf.org/html/rfc7946

[Web-Map-Custom-Element]

Web Map Custom Element. W3C Maps for HTML Community Group. URL: https://github.com/Maps4HTML/Web-Map-Custom-Element

↑