Document Object Model (DOM) Level 3 Views and Formatting Specification

Version 1.0

W3C Working Draft *15 November 2000*

This version:
http://www.w3.org/TR/2000/WD-DOM-Level-3-Views-20001115
(PostScript file , PDF file , plain text , ZIP file)
Latest version:
http://www.w3.org/TR/DOM-Level-3-Views

Editors:
Ray Whitmer, *Netscape Communications Corporation*

Abstract

This specification defines the Document Object Model Views and Formatting Level 3, a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model Views and Formatting Level 3 builds on the Document Object Model Views Level 2.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This specification is a very early version of the Views and Formatting API. This document is not guarantee to be part of the DOM Level 3 specification since the Working Group is waiting for more experience and experimentation before going further.

It is a W3C Working Draft for review by W3C members and other interested parties and may act as a starting point for the future DOM Working Group if such a Group is approved by the W3C Director. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in

progress".

Comments and experiences on this document are invited and are to be sent to the public mailing list www-dom@w3.org. The DOM Working Group will respond in the mailing list. An archive is available at http://lists.w3.org/Archives/Public/www-dom/.

This document has been produced as part of the W3C DOM Activity. The authors of this document are the DOM WG members. Different modules of the Document Object Model have different editors.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

Table of contents

Expanded Table of Contents

Copyright Notice

This document is published under the W3C Document Copyright Notice and License [p.5] . The bindings within this document are published under the W3C Software Copyright Notice and License [p.6] . The software license requires "Notice of any changes or modifications to the W3C files, including the date changes were made." Consequently, modified versions of the DOM bindings must document that they do not conform to the W3C standard; in the case of the IDL definitions, the pragma prefix can no longer be 'w3c.org'; in the case of the Java language binding, the package names can no longer be in the 'org.w3c' package.

W3C Document Copyright Notice and License

Note: This section is a copy of the W3C Document Notice and License and could be found at http://www.w3.org/Consortium/Legal/copyright-documents-19990405.

http://www.w3.org/Consortium/Legal/

Public documents on the W3C site are provided by the copyright holders under the following license. The software or Document Type Definitions (DTDs) associated with W3C specifications are governed by the Software Notice. By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on *ALL* copies of the document, or portions thereof, that you use:

1. A link or URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it doesn't exist, a notice of the form: "Copyright © [$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/" (Hypertext is preferred, but a textual representation is permitted.)
3. *If it exists*, the STATUS of the W3C document.

When space permits, inclusion of the full text of this **NOTICE** should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the Copyright FAQ) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

W3C Software Copyright Notice and License

Note: This section is a copy of the W3C Software Copyright Notice and License and could be found at http://www.w3.org/Consortium/Legal/copyright-software-19980720

Copyright © 1994-2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

http://www.w3.org/Consortium/Legal/

This W3C work (including software, documents, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and modify this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications, that you make:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers. If none exist, then a notice of the following form: "Copyright © [$date-of-software] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/."

3. Notice of any changes or modifications to the W3C files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

1. Document Object Model Views and Formatting

Editors
Ray Whitmer, Netscape Communications Corp

1.1. Overview

This chapter describes the optional DOM Level 3 *Views and Formatting* feature. A DOM application can use the `hasFeature` method of the `DOMImplementation` interface to determine whether this feature is supported or not. The feature string for generic interfaces is "ViewsAndFormatting". The feature string for visual properties and interfaces is "VisualViewsAndFormatting". The additional feature strings will be used to identify support specific to other media,

DOM implementations frequently create views of the document content available through DOM APIs. Such views present content in different ways using various processing, styling, and presentation systems. While a strong separation is typically maintained between content and the view, DOM applications may need to correlate characteristics such as position within a visual view with specific content presented within the view in order to augment and interact with the presentation.

This API allows a DOM application access to a view’s computed layout and presentation. This feature functions independently from any specific styling system that may have been applied. An implementation of this API must be able to maintain a correspondence between specific content and its presentation within the view, however the presentation was computed. Presentation state such as selection or scrolling may be manipulatable though this interface, but state which is computed or supplied from the content must be manipulated through the content.

Two versions of the API have been supplied, which are redundant in their functionality. The DOM WG has not decided which of the two is better, or if both are needed. The generic API, described first, is more robust because the specifics are contained in identifying strings passed to general mechanisms. The medium-specific APIs, described last, directly expose the attributes of the medium on the interface, which provides a flatter, simpler model for the user, but one which is less able to adapt to new or extended media types or different uses.

1.1.1. Issues

Issue VF-Issue-1:
: There are not enough examples in this document.

Issue VF-Issue-2:
: We need to comprehensively look at typical presentations and decide the initial segments, properties, orders, and lookup criteria we want to support, at least in the visual case. We also need to see if we think that Visual needs to be further subclassed. We are clearly lacking things, but how many segment types and properties do we need to reasonably meet requirements for level 3?

Issue VF-Issue-3:
: How should we represent types such as colors, fonts, and so on. How much time we can spend defining reporting value types or supporting arbitrary display value types. CSS style properties have done significant work in these areas, but it is not clear that their work is applicable for this view

model, due to differences between stylistic intent and computed results.

Issue VF-Issue-4:

What types of events should we support: keyboard, mouse, selection, repaint, layout, properties, etc. Is it reasonable to support these at the view level, without exposing the lower layers of the presentation? Can we wait for a future version of the spec? We need compelling use cases for the first release.

Issue VF-Issue-5:

Is it reasonable to expose computed content? Should this be done by creating appropriate DocumentFragments? How does this figure into the ordering of the segments, since segments which present computed content instead of other content have less natural order. We need compelling use cases for the first release.

Issue VF-Issue-6:

What about case insensitivity when comparing string values? Should it be an option, the rule, or automatic?

1.1.2. Segments

A `Segment` [p.15] is a distinct part of a view. Each Segment is privately owned and maintained by the containing view, which may destroy or reconstruct it at any time. Each Segment has a type related to the presentation medium and the function of that part of the presentation. Additional properties specific to the segment type contain information about that part of the view and identify the corresponding content, if any. Segments may also contain embedded segments where appropriate to the structure of the presentation.

A `Segment` [p.15] is not expected to have any particular structure beyond its properties, any contained segments, and dependency on the content it presents. Containment of one segment within another does not change the fact that properties such as offsets are relative to the entire view so that they may be matched, applied, and compared from anywhere within the view.

The actual segments or parts of the view are not directly available to the DOM application, but this API provides generic `Segment` [p.15] objects which can more-generally find and return items of the actual parts of the view.

1.1.3. View

A `View` [p.12] is the root of a presentation, owned and maintained by a `Document`. A view formats the contents of a document into a particular type of presentation. A view may contain general properties of the view, resource segments, and segments representing the content of a document, prepared for presentation.

A `Segment` [p.15] object specifies the actual criteria of segments to match, and captures items of each matched segment.

1.1.4. Generic and Medium-Specific APIs

The generic API provides access to variety of view and segment types by way of a medium-specific table of strings used to identify properties of medium-specific segment types:

```
// Find all selected runs of characters in the view at least half an inch from the edges.

View v = (View)((DocumentView)document).getDefaultView();
Segment q = v.createSegment();
q.setOrder("Content");
MatchSet m = q.createMatchSet(m.SET_ALL);
int hu = v.getIntegerProperty("HorizontalDPI");
int vu = v.getIntegerProperty("VerticalDPI");
n.addMatch(q.createMatchString(m.IS_EQUAL, "Type", "VisualCharacterRun");
m.addMatch(q.createMatchInteger(m.INT_FOLLOWS_OR_EQUALS, "LeftOffset", hu/2));
m.addMatch(q.createMatchInteger(m.INT_FOLLOWS_OR_EQUALS, "RightOffset", hu/2));
m.addMatch(q.createMatchInteger(m.INT_FOLLOWS_OR_EQUALS, "TopOffset", vu/2));
m.addMatch(q.createMatchInteger(m.INT_FOLLOWS_OR_EQUALS, "RightOffset", vu/2));
m.addMatch(q.createMatchBoolean(m.IS_EQUAL, "Selected", true);
q.setCriteria(m);
ContentItem start = q.createContentItem("StartContent");
ContentItem end = q.createContentItem("EndContent");
q.addItem(start);
q.addItem(end);
v.matchFirstSegment(q, 0);
while (q.getExists())
{
  // ... do Something with range from start to end...
  q.getNext();
}
```

Medium-specific APIs are flatter and easier to use, but usually sacrifice capabilities of the more-general API.

```
// Find all selected runs of characters in the view at least half an inch from the edges.


VisualView v = (VisualView)((DocumentView)document).getDefaultView();
int hu = v.getHorizontalDPI();
int vu = v.getVerticalDPI();
CharacterRun cr = v.createCharacterRun();
cr.setMatchInside(true);
cr.setMatchX(hu / 2);
cr.setMatchY(vu / 2);
cr.setMatchXR(v.getWidth() - hu);
cr.setMatchYR(v.getHeight() - vu);
cr.setMatchSelected(true);
v.matchSegment(cr);
while (cr.getExists())
{
  // ... do Something with range from start to end...
  cr.getNext();
}
```

1.2. Formal Interface Definition for a Generic View

This is the verbose, general-purpose mechanism that can handle all properties of all media types. This relies on a separate table of segment types and the associated properties and property types, because it is a single API.

Interface *View* (introduced in **DOM Level 3**)

`View` is used as the root `Segment` [p.15] , as well as providing additional global functionality such as selection.

IDL Definition

```
// Introduced in DOM Level 3:
interface View {
  void               select(in Node boundary,
                            in unsigned long offset,
                            in boolean extend,
                            in boolean add);
  Segment            createSegment();
  boolean            matchFirstSegment(inout Segment todo)
                                       raises(DOMException);
  long               getIntegerProperty(in DOMString name)
                                       raises(DOMException);
  DOMString          getStringProperty(in DOMString name)
                                       raises(DOMException);
  boolean            getBooleanProperty(in boolean name)
                                       raises(DOMException);
  Node               getContentPropertyNode(in DOMString name)
                                       raises(DOMException);
  unsigned long      getContentPropertyOffset(in DOMString name)
                                       raises(DOMException);
};
```

Methods

`createSegment`

Creates a segment that can be used to obtain segment items from the view.

Return Value

`Segment` [p.15]	A new segment object, that can be set up to obtain information about the view.

No Parameters

No Exceptions

`getBooleanProperty`

Returns the value of a boolean property of the segment, used by `Match` [p.20] es and `Item` [p.23] s.

Parameters

`name` of type `boolean`
The name of the boolean property of the segment to be retrieved.

Return Value

`boolean` The value of the named property of the `Segment` [p.15] .

Exceptions

`DOMException` NOT_SUPPORTED_ERR: Raised if the named property does not exist on the view or is not a boolean.

`getContentPropertyNode`

Returns the Node value of a content property of the segment, used by `Match` [p.20] es and `Item` [p.23] s.

Parameters

`name` of type `DOMString`
The name of the content property of the segment to be retrieved.

Return Value

`Node` The Node value of the named property of the `Segment` [p.15] .

Exceptions

`DOMException` NOT_SUPPORTED_ERR: Raised if the named property does not exist on the view or is not content.

`getContentPropertyOffset`

Returns the offset value of a content property of the segment, used by `Match` [p.20] es and `Item` [p.23] s.

Parameters

`name` of type `DOMString`
The name of the content property of the segment to be retrieved.

Return Value

`unsigned long` The offset value of the named property of the `Segment` [p.15] .

Exceptions

`DOMException` NOT_SUPPORTED_ERR: Raised if the named property does not exist on the view or is not content.

`getIntegerProperty`

Returns the value of an integer property of the segment, used by `Match` [p.20] es and `Item` [p.23] s.

Parameters

`name` of type `DOMString`

The name of the integer property of the segment to be retrieved.

Return Value

`long` The value of the named property of the `Segment` [p.15] .

Exceptions

`DOMException` NOT_SUPPORTED_ERR: Raised if the named property does not exist on the view or is not an integer.

`getStringProperty`

Returns the value of a string property of the segment, used by `Match` [p.20] es and `Item` [p.23] s.

Parameters

`name` of type `DOMString`

The name of the string property of the segment to be retrieved.

Return Value

`DOMString` The value of the named property of the `Segment` [p.15] .

Exceptions

`DOMException` NOT_SUPPORTED_ERR: Raised if the named property does not exist on the view or is not a string.

`matchFirstSegment`

Executes a `Segment` [p.15] against all nested `Segment`s, fetching`Item` [p.23] s associated the requested match number, if it exists.

Parameters

`todo` of type `Segment` [p.15]

The `Segment` to match within the view.

Return Value

`boolean` `true` if the desired match number was found, otherwise `false`.

Exceptions

`DOMException` NOT_SUPPORTED_ERR: If the segment request could not be interpreted.

`select`

Selects a new region of the document or adds to the existing selection.

Parameters

`boundary` of type `Node`

The Node at which to create or extend the selection.

`offset` of type `unsigned long`

The offset within the node at which to create or extend the selection.

`extend` of type `boolean`

If false, sets a selection anchor. If true, extends the selection with respect to the most-recently-set anchor.

`add` of type `boolean`

If false, clears any existing selection. If true adds a new region to existing selection regions.

No Return Value

No Exceptions

Interface *Segment* (introduced in **DOM Level 3**)

`Segment` is used to retrieve specific items from specific segments. Segments may be nested as a match and may be repeatedly applied for traversing multiple matching segments.

Note: Types and names of properties of segments of Visual media types

```
Integer TopOffset
Integer BottomOffset
Integer LeftOffset
Integer RightOffset
Integer Width
Integer Height
Boolean Visible
Boolean Selected
Integer ForegroundColor
Integer BackgroundColor
String FontName
String FontHeight
String FontBaseline
String FontSpace Width
String FontMaximum Width
```

Segment types

```
// Display info and root (the default segment)
Display
// An area that objects or text lines flow in
//  or are anchored to
Frame
// A single character
Character
// Sequentially-appearing characters
```

```
//  with identical properties
CharacterRun
FormField {Text | Label | Button | Menu ...}
Embedded Object
Image
```

Possible properties of specific types:

```
(Image) String URL
(Image) Boolean isLoaded
(Image) Integer ScalingFactor
(Button) Boolean isPressed
(Frame) Boolean isScrollable
```

IDL Definition

```
// Introduced in DOM Level 3:
interface Segment : Match {
           attribute Match            criteria;
           attribute DOMString        order;
  void               addItem(in Item add);
  MatchString        createMatchString(in unsigned short test,
                                       in DOMString name,
                                       in DOMString value);
  MatchInteger       createMatchInteger(in unsigned short test,
                                        in DOMString name,
                                        in long value);
  MatchBoolean       createMatchBoolean(in unsigned short test,
                                        in DOMString name,
                                        in boolean value);
  MatchContent       createMatchContent(in unsigned short test,
                                        in DOMString name,
                                        in unsigned long offset,
                                        in Node node);
  MatchSet           createMatchSet(in unsigned short test);
  StringItem         createStringItem(in DOMString name);
  IntegerItem        createIntegerItem(in DOMString name);
  BooleanItem        createBooleanItem(in DOMString name);
  ContentItem        createContentItem(in DOMString name);
  void               getItem(in unsigned long index);
  boolean            getNext();
};
```

Attributes

`criteria` of type `Match` [p.20]

The `criteria` `Match` [p.20] of a `Segment`, specified during creation, controls which `Segment`s will match.

After setting this attribute, the results of any related call to `getNext` are unpredictable until the segment has been requested again by calling `matchFirstSegment`.

`order` of type `DOMString`

The `order` string of a `Segment`, specified during creation, controls the order in which matching segments will be returned. If this attribute is not specified, the order defaults to an implementation-specific order.

After setting this attribute, the results of any related call to `getNext` are unpredictable

until the segment has been requested again by calling `matchFirstSegment`.

Methods

`addItem`

Adds a specific `Item` [p.23] to the `Segment`.

Parameters

`add` of type `Item` [p.23]

The `Item` to be added.

After adding a result, the results of any related call to `getNext` are unpredictable until the segment has been requested again by calling `matchFirstSegment`.

No Return Value

No Exceptions

`createBooleanItem`

Creates an item for a segment that can receive a boolean value.

Parameters

`name` of type `DOMString`

The name of a boolean property to be received.

Return Value

`BooleanItem` [p.24]	The requested `BooleanItem`.

No Exceptions

`createContentItem`

Creates an item for a segment that can receive a content value.

Parameters

`name` of type `DOMString`

The name of a content property to be received.

Return Value

`ContentItem` [p.24]	The requested `ContentItem`.

No Exceptions

`createIntegerItem`

Creates an item for a segment that can receive an integral value.

Parameters

`name` of type `DOMString`

The name of an integral property to be received.

Return Value

`IntegerItem` [p.24]	The requested `IntegerItem`.

No Exceptions

`createMatchBoolean`

Creates a match for a boolean value, which can be used to specify a criterium to find desired segments.

Parameters

`test` of type `unsigned short`
The match test desired.
`name` of type `DOMString`
The name of a boolean property to be compared against.
`value` of type `boolean`
The boolean value to be compared against.
Return Value

`MatchBoolean` [p.21] The requested `MatchBoolean`.

No Exceptions

`createMatchContent`
Creates a match for a content value, which can be used to specify a criterium to find desired segments.
Parameters
`test` of type `unsigned short`
The match test desired.
`name` of type `DOMString`
The name of an integer property to be compared against.
`offset` of type `unsigned long`
The offset of the content value to be compared against.
`node` of type `Node`
The Node of the content value to be compared against.
Return Value

`MatchContent` [p.22] The requested `MatchContent`.

No Exceptions

`createMatchInteger`
Creates a match for an integral value, which can be used to specify a criterium to find desired segments.
Parameters
`test` of type `unsigned short`
The match test desired.
`name` of type `DOMString`
The name of an integer property to be compared against.
`value` of type `long`
The integer value to be compared against.
Return Value

`MatchInteger` [p.21] The requested `MatchInteger`.

No Exceptions

`createMatchSet`

Creates a match for an set of matches, which can be used to specify a criterium to find desired segments.

Parameters

`test` of type `unsigned short`

The match test desired.

Return Value

`MatchSet` [p.22] The requested `MatchSet`.

No Exceptions

`createMatchString`

Creates a match for a string value, which can be used to specify a criterium to find desired segments.

Parameters

`test` of type `unsigned short`

The match test desired.

`name` of type `DOMString`

The name of a string property to be compared against.

`value` of type `DOMString`

The string value to be compared against.

Return Value

`MatchString` [p.21] The requested `MatchString`.

No Exceptions

`createStringItem`

Creates an item for a segment that can receive a string value.

Parameters

`name` of type `DOMString`

The name of a string property to be received.

Return Value

`StringItem` [p.23] The requested `StringItem`.

No Exceptions

`getItem`

Returns a specific `Item` [p.23] , of the list specified during the creation of the `Segment`, which is to be fetched during `Segment` execution, or returns null if the specified index does not correspond to a `Item`.

Parameters

`index` of type `unsigned long`

The index of the `Item` [p.23] to be retrieved.

No Return Value

No Exceptions

`getNext`

Fetches the results of the next matching `Segment`, if any.

Return Value

`boolean` `true` if another match, otherwise `false` (same value as `exists`).

No Parameters

No Exceptions

Interface *Match* (introduced in **DOM Level 3**)

The `Match` identifies `Segment` [p.15] s of which a `Segment` should fetch the `Item` [p.23] s.

IDL Definition

```
// Introduced in DOM Level 3:
interface Match {

  // MatchTestGroup
  const unsigned short      IS_EQUAL                    = 0;
  const unsigned short      IS_NOT_EQUAL                = 1;
  const unsigned short      INT_PRECEDES                = 2;
  const unsigned short      INT_PRECEDES_OR_EQUALS      = 3;
  const unsigned short      INT_FOLLOWS                 = 4;
  const unsigned short      INT_FOLLOWS_OR_EQUALS       = 5;
  const unsigned short      STR_STARTS_WITH             = 6;
  const unsigned short      STR_ENDS_WITH               = 7;
  const unsigned short      STR_CONTAINS                = 8;
  const unsigned short      SET_ANY                     = 9;
  const unsigned short      SET_ALL                     = 10;
  const unsigned short      SET_NOT_ANY                 = 11;
  const unsigned short      SET_NOT_ALL                 = 12;

  readonly attribute unsigned short   test;
};
```

Definition group *MatchTestGroup*

Defined Constants

`INT_FOLLOWS`
`INT_FOLLOWS_OR_EQUALS`
`INT_PRECEDES`
`INT_PRECEDES_OR_EQUALS`
`IS_EQUAL`
`IS_NOT_EQUAL`
`SET_ALL`
`SET_ANY`
`SET_NOT_ALL`
`SET_NOT_ANY`
`STR_CONTAINS`

STR_ENDS_WITH
STR_STARTS_WITH

Attributes

`test` of type `unsigned short`, readonly

The `test` value of a `Match`, specified during creation, controls the test to be applied.

Interface *MatchString* (introduced in **DOM level 3**)

The `MatchString` identifies `Segment` [p.15] s where a string property matches a specific value.

IDL Definition

```
// Introduced in DOM level 3:
interface MatchString : Match {
  readonly attribute DOMString        name;
  readonly attribute DOMString        value;
};
```

Attributes

`name` of type `DOMString`, readonly

The name of a string property of each `Segment` [p.15] to be compared against, which is specified during construction.

`value` of type `DOMString`, readonly

The string value to be compared against, which is specified during construction.

Interface *MatchInteger* (introduced in **DOM level 3**)

The `MatchInteger` identifies `Segment` [p.15] s where an integer property matches a specific value.

IDL Definition

```
// Introduced in DOM level 3:
interface MatchInteger : Match {
  readonly attribute DOMString        name;
  readonly attribute long             value;
};
```

Attributes

`name` of type `DOMString`, readonly

The name of an integer property of each `Segment` [p.15] to be compared against, which is specified during construction.

`value` of type `long`, readonly

The integer value to be compared against, which is specified during construction.

Interface *MatchBoolean* (introduced in **DOM level 3**)

The `MatchBoolean` identifies `Segment` [p.15] s where a boolean property matches a specific value.

IDL Definition

```
// Introduced in DOM level 3:
interface MatchBoolean : Match {
  readonly attribute DOMString        name;
  readonly attribute boolean          value;
};
```

Attributes

`name` of type `DOMString`, readonly

The name of an boolean property of each `Segment` [p.15] to be compared against, which is specified during construction.

`value` of type `boolean`, readonly

The boolean value to be compared against, which is specified during construction.

Interface *MatchContent* (introduced in **DOM level 3**)

The `MatchContent` identifies `Segment` [p.15] s where a content property matches a specific value.

IDL Definition

```
// Introduced in DOM level 3:
interface MatchContent : Match {
  readonly attribute DOMString        name;
  readonly attribute Node             node;
  readonly attribute unsigned long    offset;
};
```

Attributes

`name` of type `DOMString`, readonly

The name of an content property of each `Segment` [p.15] to be compared against, which is specified during construction.

`node` of type `Node`, readonly

The Node value to be compared against, which is specified during construction.

`offset` of type `unsigned long`, readonly

The offset value to be compared against, which is specified during construction.

Interface *MatchSet* (introduced in **DOM level 3**)

The `MatchSet` identifies `Segment` [p.15] s where a set of matches evaluate in a specified way.

IDL Definition

```
// Introduced in DOM level 3:
interface MatchSet : Match {
  readonly attribute Node             node;
  void                addMatch(in Match add);
  Match               getMatch(in unsigned long index);
};
```

Attributes

`node` of type `Node`, readonly

The Node value to be compared against, which is specified during construction.

Methods

`addMatch`

Adds a specific `Match` [p.20] to the set.

Parameters

`add` of type `Match` [p.20]

The `Match` to be added.

After adding a match, the results of any related call to `getNext` are unpredictable until the segment has been requested again by calling `matchFirstSegment`.

No Return Value

No Exceptions

`getMatch`

Returns a specific `Match` [p.20] , of the set, which is to be matched during `MatchSet` evaluation, or returns null if the specified index does not correspond to a `Match`.

Parameters

`index` of type `unsigned long`

The index of the `Match` [p.20] to be retrieved.

Return Value

`Match` [p.20]	The requested match, if any, or null.

No Exceptions

Interface *Item* (introduced in **DOM Level 3**)

The `Item` represents information to be fetched by a `Segment` [p.15] .

IDL Definition

```
// Introduced in DOM Level 3:
interface Item {
  readonly attribute boolean          exists;
  readonly attribute DOMString        name;
};
```

Attributes

`exists` of type `boolean`, readonly

The `exists` boolean of a `Segment` [p.15] , initially set to false during creation, is set after an attempt to fetch the values of a `Item` to indicate whether or not the required data was present. A true value indicates that it was.

`name` of type `DOMString`, readonly

The name of a property of the matched `Segment` [p.15] to be fetched, which is specified during construction.

Interface *StringItem* (introduced in **DOM Level 3**)

The `StringItem` represents a string property to be fetched by a `Segment` [p.15] .

IDL Definition

```
// Introduced in DOM Level 3:
interface StringItem : Item {
  readonly attribute DOMString        value;
};
```

Attributes

`value` of type `DOMString`, readonly

The string value returned by the `Segment` [p.15] , which is undefined if `exists` is false.

Interface *IntegerItem* (introduced in **DOM Level 3**)

The `IntegerItem` represents an integer property to be fetched by a `Segment` [p.15] .

IDL Definition

```
// Introduced in DOM Level 3:
interface IntegerItem : Item {
  readonly attribute long             value;
};
```

Attributes

`value` of type `long`, readonly

The integer value returned by the `Segment` [p.15] , which is undefined if `exists` is false.

Interface *BooleanItem* (introduced in **DOM Level 3**)

The `BooleanItem` represents a boolean property to be fetched by a `Segment` [p.15] .

IDL Definition

```
// Introduced in DOM Level 3:
interface BooleanItem : Item {
           attribute boolean          value;
};
```

Attributes

`value` of type `boolean`

The boolean value returned by the `Segment` [p.15] , which is undefined if `exists` is false.

Interface *ContentItem* (introduced in **DOM Level 3**)

The `ContentItem` represents a content property to be fetched by a `Segment` [p.15] .

IDL Definition

```
// Introduced in DOM Level 3:
interface ContentItem : Item {
           attribute Node             node;
           attribute unsigned long    offset;
};
```

Attributes

`node` of type `Node`

The `Node` value returned by the `Segment` [p.15] , which is undefined if `exists` is false.

`offset` of type `unsigned long`

The offset value returned by the `Segment` [p.15] , which is undefined if `exists` is false.

1.3. Formal Interface Definition for a Visual View

This is the flatter mechanism that handles only one specific medium, in this case, visual. This does not rely on a table of property names, because all supported criteria and properties are attributes of the interfaces.

Interface *VisualView*

Presents a flatter model of a visual view.

IDL Definition

```
interface VisualView {
  readonly attribute DOMString        fontScheme;
  readonly attribute unsigned long    width;
  readonly attribute unsigned long    height;
  readonly attribute unsigned long    horizontalDPI;
  readonly attribute unsigned long    verticalDPI;
  VisualCharacter    createVisualCharacter();
  VisualCharacterRun createVisualCharacterRun();
  VisualFrame        createVisualFrame();
  VisualImage        createVisualImage();
  VisualFormButton   createVisualFormButton();
  VisualFormField    createVisualFormField();
  void               select(in Node boundary,
                            in unsigned long offset,
                            in boolean extend,
                            in boolean add);
  void               matchSegment(in VisualResource segment);
};
```

Attributes

`fontScheme` of type `DOMString`, readonly

A string identifying the type of fonts on the system so that font name strings may be properly interpreted.

`height` of type `unsigned long`, readonly

The height, in vertical units, of the view.

`horizontalDPI` of type `unsigned long`, readonly

The number of horizontal dots per inch in the view, used to interpret horizontal values.

`verticalDPI` of type `unsigned long`, readonly

The number of vertical dots per inch in the view, used to interpret vertical values.

`width` of type `unsigned long`, readonly

The width, in horizontal units, of the view.

Methods

`createVisualCharacter`

Creates a visual character to match and return information on a single visual character of the view.

Return Value

`VisualCharacter` [p.31] The requested `VisualCharacter`.

No Parameters

No Exceptions

`createVisualCharacterRun`

Creates a visual character run to match and return information on a run of similar ajdacent visual characters of the view.

This will match the largest character run that meets the specified criteria, is not contiguously displayed on the view and has homogeneous display properties.

Return Value

`VisualCharacterRun` [p.31] The requested `VisualCharacterRun`.

No Parameters

No Exceptions

`createVisualFormButton`

Creates a visual form button to match and return information on a form button of the view.

Return Value

`VisualFormButton` [p.32] The requested `VisualFormButton`.

No Parameters

No Exceptions

`createVisualFormField`

Creates a visual form field to match and return information on a form field of the view.

Return Value

`VisualFormField` [p.32] The requested `VisualFormField`.

No Parameters

No Exceptions

`createVisualFrame`

Creates a visual frame to match and return information on a frame of the view.

Return Value

`VisualFrame` [p.32] The requested `VisualFrame`.

No Parameters
No Exceptions

`createVisualImage`
Creates a visual image to match and return information on an image of the view.
Return Value

`VisualImage` [p.32]	The requested `VisualImage`.

No Parameters
No Exceptions

`matchSegment`
Parameters
`segment` of type `VisualResource` [p.27]
No Return Value
No Exceptions

`select`
Parameters
`boundary` of type `Node`
`offset` of type `unsigned long`
`extend` of type `boolean`
`add` of type `boolean`
No Return Value
No Exceptions

Interface *VisualResource*

Visual segments allow things within a visual view to be accessed.

IDL Definition

```
interface VisualResource {
};
```

Interface *VisualFont*

Visual font resources contain match criteria and result attributes for getting information about fonts available to a view.

IDL Definition

```
interface VisualFont : VisualResource {
           attribute DOMString        matchFontName;
  readonly attribute boolean          exists;
  readonly attribute DOMString        fontName;
  boolean            getNext();
};
```

Attributes

`exists` of type `boolean`, readonly
: Returns true result if the desired font was located, or false if it was not. If this value is set to false, no other results are set. If this value is set to true, all other results are set.

`fontName` of type `DOMString`, readonly
: When a font is matched, the name of the font is returned here.

`matchFontName` of type `DOMString`
: May be set to cause fonts with the corresponding name to be matched.

Methods

`getNext`
: Fetches the results of the next matching `VisualFont`, if any.

Return Value

```
boolean
```

No Parameters

No Exceptions

Interface *VisualSegment*

Visual segments contain match criteria attributes and result attributes common to visual views of a document. When this structure is created, all booleans are set to false, all integral values are set to 0, and all strings and object references are set to null. Match criteria are then set. After setting match criteria, `matchSegment` is called passing this segment or another segment that references this segment, which finds a matching segment and sets result attributes.

IDL Definition

```
interface VisualSegment : VisualResource {
           attribute boolean          matchPosition;
           attribute boolean          matchInside;
           attribute boolean          matchContaining;
           attribute long             matchX;
           attribute long             matchY;
           attribute long             matchXR;
           attribute long             matchYR;
           attribute boolean          matchContent;
           attribute boolean          matchRange;
           attribute Node             matchNode;
           attribute unsigned long    matchOffset;
           attribute Node             matchNodeR;
           attribute unsigned long    matchOffsetR;
           attribute boolean          matchContainsSelected;
           attribute boolean          matchContainsVisible;
  readonly attribute boolean          exists;
  readonly attribute Node             startNode;
  readonly attribute unsigned long    startOffset;
  readonly attribute Node             endNode;
  readonly attribute unsigned long    endOffset;
  readonly attribute long             topOffset;
  readonly attribute long             bottomOffset;
  readonly attribute long             leftOffset;
  readonly attribute long             rightOffset;
```

```
  readonly attribute unsigned long    width;
  readonly attribute unsigned long    height;
  readonly attribute boolean          selected;
  readonly attribute boolean          visible;
  readonly attribute unsigned long    foregroundColor;
  readonly attribute unsigned long    backgroundColor;
  readonly attribute DOMString        fontName;
  readonly attribute DOMString        fontHeight;
  boolean            getNext();
};
```

Attributes

`backgroundColor` of type `unsigned long`, readonly

Whenever a segment is matched, this is set to the integral value of the background color of that segment, or transparent if there is no background color. The 32 bits of this value are divided into the following 8-bit sub-fields, from most significant to least significant: alpha, red, green, blue. The color fields range from 0 for no intensity to 255 to indicate the contribution of each color. The alpha field ranges from 0 for transparent to 255 for completely opaque. For a transparent alpha value of 0, the color fields are be normalized to 0 as well.

`bottomOffset` of type `long`, readonly

Whenever a segment is matched, this is set to the bottom offset of the segment within the view, specified in vertical view units.

`endNode` of type `Node`, readonly

Whenever a segment is matched, this is set to the last node presented by the matched segment or null if the segment does not present any specific document content.

`endOffset` of type `unsigned long`, readonly

Whenever a segment is matched, this is set to first offset not presented within the last node presented by the matched segment or 0 if the segment does not present any specific document content.

`exists` of type `boolean`, readonly

Returns true result if the desired segment was located, or false if it was not. If this value is set to false, no other results are set. If this value is set to true, all other results are set.

`fontHeight` of type `DOMString`, readonly

`fontName` of type `DOMString`, readonly

The font name is a view-specific designation of the font name.

`foregroundColor` of type `unsigned long`, readonly

Whenever a segment is matched, this is set to the integral value of the foreground color of that segment, or transparent if there is no foreground color. The 32 bits of this value are divided into the following 8-bit sub-fields, from most significant to least significant: alpha, red, green, blue. The color fields range from 0 for no intensity to 255 to indicate the contribution of each color. The alpha field ranges from 0 for transparent to 255 for completely opaque. For complete transparency, the color fields will be normalized to 0 as well.

`height` of type `unsigned long`, readonly

Whenever a segment is matched, this is set to the width of the segment within the view, specified in vertical view units.

`leftOffset` of type `long`, readonly
: Whenever a segment is matched, this is set to the left offset of the segment within the view, specified in horizontal view units.

`matchContaining` of type `boolean`
: May be set to cause the corresponding segment to be matched only if it contains the specified rectangular region bounded by `matchX`, `matchY`, `matchXR`, and `matchYR`.

`matchContainsSelected` of type `boolean`
: May be set to cause the corresponding segment to only be matched if the content being presented contains a cursor or part of a selected region.

`matchContainsVisible` of type `boolean`
: May be set to cause the corresponding segment to only be matched if the segment being presented contains some part that is visible.

`matchContent` of type `boolean`
: May be set to cause the corresponding segment to only be matched if it presents the `matchNode` content, offset by `matchOffset`.

`matchInside` of type `boolean`
: May be set to cause the corresponding segment to be matched only if it is inside the specified rectangular region bounded by `matchX`, `matchY`, `matchXR`, and `matchYR`.

`matchNode` of type `Node`
: The node, or first node in a range to use to match segments which present specified content.

  If matching content is enabled, but this is set to null, then only segments that are not associated with content will be matched.

`matchNodeR` of type `Node`
: The second node in a range to use to match segments which present specified content.

  If matching a content range is enabled, but this is set to null, then only segments that are not associated with content will be matched.

`matchOffset` of type `unsigned long`
: The offset, or first offset in a range to use to match segments which present specified content.

`matchOffsetR` of type `unsigned long`
: The offset, or first offset in a range to use to match segments which present specified content.

`matchPosition` of type `boolean`
: May be set to cause the corresponding segment to be matched only if it contains the specified `matchX` and `matchY` positions.

`matchRange` of type `boolean`
: May be set to cause the corresponding segment to only be matched if the content it presents is within the range of content between Node `matchNode` offset `matchOffset` and Node `matchNodeR` offset `matchOffsetR`.

`matchX` of type `long`
: An integral X coordinate, specified in horizontal view units, that may be used to match a point or region.

`matchXR` of type `long`
: An integral X coordinate, specified in horizontal view units, that may be used to match a region.

`matchY` of type `long`
: An integral Y coordinate, specified in vertical view units, that may be used to match a point or region.

`matchYR` of type `long`
: An integral Y coordinate, specified in vertical view units, that may be used to match a region.

`rightOffset` of type `long`, readonly
: Whenever a segment is matched, this is set to the right offset of the segment within the view, specified in horizontal view units.

`selected` of type `boolean`, readonly
: Whenever a segment is matched, this is set to true if the segment presents the content with the cursor or selected content, otherwise, this is set to false.

`startNode` of type `Node`, readonly
: Whenever a segment is matched, this is set to the first node presented by the matched segment or null if the segment does not present any specific document content.

`startOffset` of type `unsigned long`, readonly
: Whenever a segment is matched, this is set to the first offset presented within the first node presented by the matched segment or 0 if the segment does not present any specific document content.

`topOffset` of type `long`, readonly
: Whenever a segment is matched, this is set to the top offset of the segment within the view, specified in vertical view units.

`visible` of type `boolean`, readonly
: Whenever a segment is matched, this is set to true if the segment contains some part that is visible, otherwise, this is set to false.

`width` of type `unsigned long`, readonly
: Whenever a segment is matched, this is set to the width of the segment within the view, specified in horizontal view units.

Methods

`getNext`
: Fetches the results of the next matching `VisualResource` [p.27] , if any.

Return Value

```
boolean
```

No Parameters

No Exceptions

Interface *VisualCharacter*

IDL Definition

```
interface VisualCharacter : VisualSegment {
};
```

Interface *VisualCharacterRun*

IDL Definition

```
interface VisualCharacterRun : VisualSegment {
};
```

Interface *VisualFrame*

IDL Definition

```
interface VisualFrame : VisualSegment {
  readonly attribute VisualSegment    embedded;
};
```

Attributes

`embedded` of type `VisualSegment` [p.28] , readonly

May be set to contain embedded visual segments inside the frame. If this value is set, the embedded segment serves as a conditional for the frame while receiving the results of the embedded segment that was matched.

Interface *VisualImage*

IDL Definition

```
interface VisualImage : VisualSegment {
  readonly attribute DOMString        imageURL;
  readonly attribute boolean          isLoaded;
};
```

Attributes

`imageURL` of type `DOMString`, readonly

`isLoaded` of type `boolean`, readonly

Interface *VisualFormButton*

IDL Definition

```
interface VisualFormButton : VisualSegment {
  readonly attribute boolean          isPressed;
};
```

Attributes

`isPressed` of type `boolean`, readonly

Interface *VisualFormField*

IDL Definition

```
interface VisualFormField : VisualSegment {
  readonly attribute DOMString        formValue;
};
```

Attributes

`formValue` of type `DOMString`, readonly

Appendix A: IDL Definitions

This appendix contains the complete OMG IDL [OMGIDL] for the Level 3 Document Object Views and Formatting definitions.

The IDL files are also available as:
http://www.w3.org/TR/2000/WD-DOM-Level-3-Views-20001115/idl.zip

views.idl:

```
// File: views.idl

#ifndef _VIEWS_IDL_
#define _VIEWS_IDL_

#include "dom.idl"

#pragma prefix "dom.w3c.org"
module views
{

  typedef dom::Node Node;
  typedef dom::DOMString DOMString;

  interface Segment;
  interface VisualResource;
  interface VisualCharacter;
  interface VisualCharacterRun;
  interface VisualFrame;
  interface VisualImage;
  interface VisualFormButton;
  interface VisualFormField;

  // Introduced in DOM Level 3:
  interface View {
    void               select(in Node boundary,
                              in unsigned long offset,
                              in boolean extend,
                              in boolean add);
    Segment            createSegment();
    boolean            matchFirstSegment(inout Segment todo)
                                    raises(dom::DOMException);
    long               getIntegerProperty(in DOMString name)
                                    raises(dom::DOMException);
    DOMString          getStringProperty(in DOMString name)
                                    raises(dom::DOMException);
    boolean            getBooleanProperty(in boolean name)
                                    raises(dom::DOMException);
    Node               getContentPropertyNode(in DOMString name)
                                    raises(dom::DOMException);
    unsigned long      getContentPropertyOffset(in DOMString name)
                                    raises(dom::DOMException);
  };
```

```
// Introduced in DOM Level 3:
interface Match {

  // MatchTestGroup
  const unsigned short      IS_EQUAL                      = 0;
  const unsigned short      IS_NOT_EQUAL                  = 1;
  const unsigned short      INT_PRECEDES                  = 2;
  const unsigned short      INT_PRECEDES_OR_EQUALS        = 3;
  const unsigned short      INT_FOLLOWS                   = 4;
  const unsigned short      INT_FOLLOWS_OR_EQUALS         = 5;
  const unsigned short      STR_STARTS_WITH               = 6;
  const unsigned short      STR_ENDS_WITH                 = 7;
  const unsigned short      STR_CONTAINS                  = 8;
  const unsigned short      SET_ANY                       = 9;
  const unsigned short      SET_ALL                       = 10;
  const unsigned short      SET_NOT_ANY                   = 11;
  const unsigned short      SET_NOT_ALL                   = 12;

  readonly attribute unsigned short   test;
};

// Introduced in DOM level 3:
interface MatchString : Match {
  readonly attribute DOMString        name;
  readonly attribute DOMString        value;
};

// Introduced in DOM level 3:
interface MatchInteger : Match {
  readonly attribute DOMString        name;
  readonly attribute long             value;
};

// Introduced in DOM level 3:
interface MatchBoolean : Match {
  readonly attribute DOMString        name;
  readonly attribute boolean          value;
};

// Introduced in DOM level 3:
interface MatchContent : Match {
  readonly attribute DOMString        name;
  readonly attribute Node             node;
  readonly attribute unsigned long    offset;
};

// Introduced in DOM level 3:
interface MatchSet : Match {
  readonly attribute Node             node;
  void                addMatch(in Match add);
  Match               getMatch(in unsigned long index);
};

// Introduced in DOM Level 3:
interface Item {
  readonly attribute boolean          exists;
  readonly attribute DOMString        name;
```

```
};

// Introduced in DOM Level 3:
interface StringItem : Item {
  readonly attribute DOMString        value;
};

// Introduced in DOM Level 3:
interface IntegerItem : Item {
  readonly attribute long             value;
};

// Introduced in DOM Level 3:
interface BooleanItem : Item {
           attribute boolean          value;
};

// Introduced in DOM Level 3:
interface ContentItem : Item {
           attribute Node             node;
           attribute unsigned long    offset;
};

interface VisualView {
  readonly attribute DOMString        fontScheme;
  readonly attribute unsigned long    width;
  readonly attribute unsigned long    height;
  readonly attribute unsigned long    horizontalDPI;
  readonly attribute unsigned long    verticalDPI;
  VisualCharacter    createVisualCharacter();
  VisualCharacterRun createVisualCharacterRun();
  VisualFrame        createVisualFrame();
  VisualImage        createVisualImage();
  VisualFormButton   createVisualFormButton();
  VisualFormField    createVisualFormField();
  void               select(in Node boundary,
                            in unsigned long offset,
                            in boolean extend,
                            in boolean add);
  void               matchSegment(in VisualResource segment);
};

interface VisualResource {
};

interface VisualFont : VisualResource {
           attribute DOMString        matchFontName;
  readonly attribute boolean          exists;
  readonly attribute DOMString        fontName;
  boolean            getNext();
};

interface VisualSegment : VisualResource {
           attribute boolean          matchPosition;
           attribute boolean          matchInside;
           attribute boolean          matchContaining;
           attribute long             matchX;
```

```
           attribute long             matchY;
           attribute long             matchXR;
           attribute long             matchYR;
           attribute boolean          matchContent;
           attribute boolean          matchRange;
           attribute Node             matchNode;
           attribute unsigned long    matchOffset;
           attribute Node             matchNodeR;
           attribute unsigned long    matchOffsetR;
           attribute boolean          matchContainsSelected;
           attribute boolean          matchContainsVisible;
  readonly attribute boolean          exists;
  readonly attribute Node             startNode;
  readonly attribute unsigned long    startOffset;
  readonly attribute Node             endNode;
  readonly attribute unsigned long    endOffset;
  readonly attribute long             topOffset;
  readonly attribute long             bottomOffset;
  readonly attribute long             leftOffset;
  readonly attribute long             rightOffset;
  readonly attribute unsigned long    width;
  readonly attribute unsigned long    height;
  readonly attribute boolean          selected;
  readonly attribute boolean          visible;
  readonly attribute unsigned long    foregroundColor;
  readonly attribute unsigned long    backgroundColor;
  readonly attribute DOMString        fontName;
  readonly attribute DOMString        fontHeight;
  boolean            getNext();
};

interface VisualCharacter : VisualSegment {
};

interface VisualCharacterRun : VisualSegment {
};

interface VisualFrame : VisualSegment {
  readonly attribute VisualSegment    embedded;
};

interface VisualImage : VisualSegment {
  readonly attribute DOMString        imageURL;
  readonly attribute boolean          isLoaded;
};

interface VisualFormButton : VisualSegment {
  readonly attribute boolean          isPressed;
};

interface VisualFormField : VisualSegment {
  readonly attribute DOMString        formValue;
};

// Introduced in DOM Level 3:
interface Segment : Match {
           attribute Match            criteria;
```

```
             attribute DOMString        order;
    void               addItem(in Item add);
    MatchString        createMatchString(in unsigned short test,
                                         in DOMString name,
                                         in DOMString value);
    MatchInteger       createMatchInteger(in unsigned short test,
                                          in DOMString name,
                                          in long value);
    MatchBoolean       createMatchBoolean(in unsigned short test,
                                          in DOMString name,
                                          in boolean value);
    MatchContent       createMatchContent(in unsigned short test,
                                          in DOMString name,
                                          in unsigned long offset,
                                          in Node node);
    MatchSet           createMatchSet(in unsigned short test);
    StringItem         createStringItem(in DOMString name);
    IntegerItem        createIntegerItem(in DOMString name);
    BooleanItem        createBooleanItem(in DOMString name);
    ContentItem        createContentItem(in DOMString name);
    void               getItem(in unsigned long index);
    boolean            getNext();
  };
};

#endif // _VIEWS_IDL_
```

views.idl:

Appendix B: Java Language Binding

This appendix contains the complete Java [Java] bindings for the Level 3 Document Object Views and Formatting.

The Java files are also available as
http://www.w3.org/TR/2000/WD-DOM-Level-3-Views-20001115/java-binding.zip

org/w3c/dom/views/View.java:

```
package org.w3c.dom.views;

import org.w3c.dom.Node;
import org.w3c.dom.DOMException;

public interface View {
    public void select(Node boundary,
                       int offset,
                       boolean extend,
                       boolean add);

    public Segment createSegment();

    public boolean matchFirstSegment(Segment todo)
                                     throws DOMException;

    public int getIntegerProperty(String name)
                                  throws DOMException;

    public String getStringProperty(String name)
                                    throws DOMException;

    public boolean getBooleanProperty(boolean name)
                                      throws DOMException;

    public Node getContentPropertyNode(String name)
                                       throws DOMException;

    public int getContentPropertyOffset(String name)
                                        throws DOMException;

}
```

org/w3c/dom/views/Segment.java:

```
package org.w3c.dom.views;

import org.w3c.dom.Node;

public interface Segment extends Match {
    public Match getCriteria();
    public void setCriteria(Match criteria);

    public String getOrder();
```

```
    public void setOrder(String order);

    public void addItem(Item add);

    public MatchString createMatchString(short test,
                                         String name,
                                         String value);

    public MatchInteger createMatchInteger(short test,
                                           String name,
                                           int value);

    public MatchBoolean createMatchBoolean(short test,
                                           String name,
                                           boolean value);

    public MatchContent createMatchContent(short test,
                                           String name,
                                           int offset,
                                           Node node);

    public MatchSet createMatchSet(short test);

    public StringItem createStringItem(String name);

    public IntegerItem createIntegerItem(String name);

    public BooleanItem createBooleanItem(String name);

    public ContentItem createContentItem(String name);

    public void getItem(int index);

    public boolean getNext();

}
```

org/w3c/dom/views/Match.java:

```
package org.w3c.dom.views;

public interface Match {
    // MatchTestGroup
    public static final short IS_EQUAL                  = 0;
    public static final short IS_NOT_EQUAL              = 1;
    public static final short INT_PRECEDES              = 2;
    public static final short INT_PRECEDES_OR_EQUALS    = 3;
    public static final short INT_FOLLOWS               = 4;
    public static final short INT_FOLLOWS_OR_EQUALS     = 5;
    public static final short STR_STARTS_WITH           = 6;
    public static final short STR_ENDS_WITH             = 7;
    public static final short STR_CONTAINS              = 8;
    public static final short SET_ANY                   = 9;
    public static final short SET_ALL                   = 10;
    public static final short SET_NOT_ANY               = 11;
    public static final short SET_NOT_ALL               = 12;
```

```
    public short getTest();

}
```

org/w3c/dom/views/MatchString.java:

```
package org.w3c.dom.views;

public interface MatchString extends Match {
    public String getName();

    public String getValue();

}
```

org/w3c/dom/views/MatchInteger.java:

```
package org.w3c.dom.views;

public interface MatchInteger extends Match {
    public String getName();

    public int getValue();

}
```

org/w3c/dom/views/MatchBoolean.java:

```
package org.w3c.dom.views;

public interface MatchBoolean extends Match {
    public String getName();

    public boolean getValue();

}
```

org/w3c/dom/views/MatchContent.java:

```
package org.w3c.dom.views;

import org.w3c.dom.Node;

public interface MatchContent extends Match {
    public String getName();

    public Node getNode();

    public int getOffset();

}
```

org/w3c/dom/views/MatchSet.java:

```
package org.w3c.dom.views;

import org.w3c.dom.Node;

public interface MatchSet extends Match {
    public Node getNode();

    public void addMatch(Match add);

    public Match getMatch(int index);

}
```

org/w3c/dom/views/Item.java:

```
package org.w3c.dom.views;

public interface Item {
    public boolean getExists();

    public String getName();

}
```

org/w3c/dom/views/StringItem.java:

```
package org.w3c.dom.views;

public interface StringItem extends Item {
    public String getValue();

}
```

org/w3c/dom/views/IntegerItem.java:

```
package org.w3c.dom.views;

public interface IntegerItem extends Item {
    public int getValue();

}
```

org/w3c/dom/views/BooleanItem.java:

```
package org.w3c.dom.views;

public interface BooleanItem extends Item {
    public boolean getValue();
    public void setValue(boolean value);

}
```

org/w3c/dom/views/ContentItem.java:

```
package org.w3c.dom.views;

import org.w3c.dom.Node;

public interface ContentItem extends Item {
    public Node getNode();
    public void setNode(Node node);

    public int getOffset();
    public void setOffset(int offset);

}
```

org/w3c/dom/views/VisualView.java:

```
package org.w3c.dom.views;

import org.w3c.dom.Node;

public interface VisualView {
    public String getFontScheme();

    public int getWidth();

    public int getHeight();

    public int getHorizontalDPI();

    public int getVerticalDPI();

    public VisualCharacter createVisualCharacter();

    public VisualCharacterRun createVisualCharacterRun();

    public VisualFrame createVisualFrame();

    public VisualImage createVisualImage();

    public VisualFormButton createVisualFormButton();

    public VisualFormField createVisualFormField();

    public void select(Node boundary,
                       int offset,
                       boolean extend,
                       boolean add);

    public void matchSegment(VisualResource segment);

}
```

org/w3c/dom/views/VisualResource.java:

```
package org.w3c.dom.views;

public interface VisualResource {
}
```

org/w3c/dom/views/VisualFont.java:

```
package org.w3c.dom.views;

public interface VisualFont extends VisualResource {
    public String getMatchFontName();
    public void setMatchFontName(String matchFontName);

    public boolean getExists();

    public String getFontName();

    public boolean getNext();

}
```

org/w3c/dom/views/VisualSegment.java:

```
package org.w3c.dom.views;

import org.w3c.dom.Node;

public interface VisualSegment extends VisualResource {
    public boolean getMatchPosition();
    public void setMatchPosition(boolean matchPosition);

    public boolean getMatchInside();
    public void setMatchInside(boolean matchInside);

    public boolean getMatchContaining();
    public void setMatchContaining(boolean matchContaining);

    public int getMatchX();
    public void setMatchX(int matchX);

    public int getMatchY();
    public void setMatchY(int matchY);

    public int getMatchXR();
    public void setMatchXR(int matchXR);

    public int getMatchYR();
    public void setMatchYR(int matchYR);

    public boolean getMatchContent();
    public void setMatchContent(boolean matchContent);

    public boolean getMatchRange();
```

```
    public void setMatchRange(boolean matchRange);

    public Node getMatchNode();
    public void setMatchNode(Node matchNode);

    public int getMatchOffset();
    public void setMatchOffset(int matchOffset);

    public Node getMatchNodeR();
    public void setMatchNodeR(Node matchNodeR);

    public int getMatchOffsetR();
    public void setMatchOffsetR(int matchOffsetR);

    public boolean getMatchContainsSelected();
    public void setMatchContainsSelected(boolean matchContainsSelected);

    public boolean getMatchContainsVisible();
    public void setMatchContainsVisible(boolean matchContainsVisible);

    public boolean getExists();

    public Node getStartNode();

    public int getStartOffset();

    public Node getEndNode();

    public int getEndOffset();

    public int getTopOffset();

    public int getBottomOffset();

    public int getLeftOffset();

    public int getRightOffset();

    public int getWidth();

    public int getHeight();

    public boolean getSelected();

    public boolean getVisible();

    public int getForegroundColor();

    public int getBackgroundColor();

    public String getFontName();

    public String getFontHeight();

    public boolean getNext();

}
```

org/w3c/dom/views/VisualCharacter.java:

```
package org.w3c.dom.views;

public interface VisualCharacter extends VisualSegment {
}
```

org/w3c/dom/views/VisualCharacterRun.java:

```
package org.w3c.dom.views;

public interface VisualCharacterRun extends VisualSegment {
}
```

org/w3c/dom/views/VisualFrame.java:

```
package org.w3c.dom.views;

public interface VisualFrame extends VisualSegment {
    public VisualSegment getEmbedded();

}
```

org/w3c/dom/views/VisualImage.java:

```
package org.w3c.dom.views;

public interface VisualImage extends VisualSegment {
    public String getImageURL();

    public boolean getIsLoaded();

}
```

org/w3c/dom/views/VisualFormButton.java:

```
package org.w3c.dom.views;

public interface VisualFormButton extends VisualSegment {
    public boolean getIsPressed();

}
```

org/w3c/dom/views/VisualFormField.java:

```
package org.w3c.dom.views;

public interface VisualFormField extends VisualSegment {
    public String getFormValue();

}
```

Appendix C: ECMA Script Language Binding

This appendix contains the complete ECMA Script [ECMAScript] binding for the Level 3 Document Object Model Views and Formatting definitions.

Object **View**

The **View** object has the following methods:

select(boundary, offset, extend, add)

This method has no return value.
The **boundary** parameter is a **Node** object.
The **offset** parameter is of type **Number**.
The **extend** parameter is of type **Boolean**.
The **add** parameter is of type **Boolean**.

createSegment()

This method returns a **Segment** object.

matchFirstSegment(todo)

This method returns a **Boolean**.
The **todo** parameter is a **Segment** object.
This method can raise a **DOMException** object.

getIntegerProperty(name)

This method returns a **long** object.
The **name** parameter is of type **String**.
This method can raise a **DOMException** object.

getStringProperty(name)

This method returns a **String**.
The **name** parameter is of type **String**.
This method can raise a **DOMException** object.

getBooleanProperty(name)

This method returns a **Boolean**.
The **name** parameter is of type **Boolean**.
This method can raise a **DOMException** object.

getContentPropertyNode(name)

This method returns a **Node** object.
The **name** parameter is of type **String**.
This method can raise a **DOMException** object.

getContentPropertyOffset(name)

This method returns a **Number**.
The **name** parameter is of type **String**.
This method can raise a **DOMException** object.

Object **Segment**

Segment has the all the properties and methods of the **Match** object as well as the properties and methods defined below.

The **Segment** object has the following properties:

criteria

This property is a **Match** object.

order

This property is of type **String**.

The **Segment** object has the following methods:

addItem(add)

This method has no return value.

The **add** parameter is a **Item** object.

createMatchString(test, name, value)

This method returns a **MatchString** object.

The **test** parameter is of type **Number**.

The **name** parameter is of type **String**.

The **value** parameter is of type **String**.

createMatchInteger(test, name, value)

This method returns a **MatchInteger** object.

The **test** parameter is of type **Number**.

The **name** parameter is of type **String**.

The **value** parameter is a **long** object.

createMatchBoolean(test, name, value)

This method returns a **MatchBoolean** object.

The **test** parameter is of type **Number**.

The **name** parameter is of type **String**.

The **value** parameter is of type **Boolean**.

createMatchContent(test, name, offset, node)

This method returns a **MatchContent** object.

The **test** parameter is of type **Number**.

The **name** parameter is of type **String**.

The **offset** parameter is of type **Number**.

The **node** parameter is a **Node** object.

createMatchSet(test)

This method returns a **MatchSet** object.

The **test** parameter is of type **Number**.

createStringItem(name)

This method returns a **StringItem** object.

The **name** parameter is of type **String**.

createIntegerItem(name)

This method returns a **IntegerItem** object.

The **name** parameter is of type **String**.

createBooleanItem(name)

This method returns a **BooleanItem** object.

The **name** parameter is of type **String**.

createContentItem(name)

This method returns a **ContentItem** object.

The **name** parameter is of type **String**.

getItem(index)

This method has no return value.

The **index** parameter is of type **Number**.

getNext()

This method returns a **Boolean**.

Prototype Object **Match**

The **Match** class has the following constants:

Match.IS_EQUAL

This constant is of type **Number** and its value is **0**.

Match.IS_NOT_EQUAL

This constant is of type **Number** and its value is **1**.

Match.INT_PRECEDES

This constant is of type **Number** and its value is **2**.

Match.INT_PRECEDES_OR_EQUALS

This constant is of type **Number** and its value is **3**.

Match.INT_FOLLOWS

This constant is of type **Number** and its value is **4**.

Match.INT_FOLLOWS_OR_EQUALS

This constant is of type **Number** and its value is **5**.

Match.STR_STARTS_WITH

This constant is of type **Number** and its value is **6**.

Match.STR_ENDS_WITH

This constant is of type **Number** and its value is **7**.

Match.STR_CONTAINS

This constant is of type **Number** and its value is **8**.

Match.SET_ANY

This constant is of type **Number** and its value is **9**.

Match.SET_ALL

This constant is of type **Number** and its value is **10**.

Match.SET_NOT_ANY

This constant is of type **Number** and its value is **11**.

Match.SET_NOT_ALL

This constant is of type **Number** and its value is **12**.

Object **Match**

The **Match** object has the following properties:

test

This read-only property is of type **Number**.

Object **MatchString**

MatchString has the all the properties and methods of the **Match** object as well as the properties and methods defined below.

The **MatchString** object has the following properties:

name

This read-only property is of type **String**.

value

This read-only property is of type **String**.

Object **MatchInteger**

MatchInteger has the all the properties and methods of the **Match** object as well as the properties and methods defined below.

The **MatchInteger** object has the following properties:

name

This read-only property is of type **String**.

value

This read-only property is a **long** object.

Object **MatchBoolean**

MatchBoolean has the all the properties and methods of the **Match** object as well as the properties and methods defined below.

The **MatchBoolean** object has the following properties:

name

This read-only property is of type **String**.

value

This read-only property is of type **Boolean**.

Object **MatchContent**

MatchContent has the all the properties and methods of the **Match** object as well as the properties and methods defined below.

The **MatchContent** object has the following properties:

name

This read-only property is of type **String**.

node

This read-only property is a **Node** object.

offset

This read-only property is of type **Number**.

Object **MatchSet**

MatchSet has the all the properties and methods of the **Match** object as well as the properties and methods defined below.

The **MatchSet** object has the following properties:

node

This read-only property is a **Node** object.

The **MatchSet** object has the following methods:

addMatch(add)

This method has no return value.

The **add** parameter is a **Match** object.

getMatch(index)

This method returns a **Match** object.

The **index** parameter is of type **Number**.

Object **Item**

The **Item** object has the following properties:

exists

This read-only property is of type **Boolean**.

name

This read-only property is of type **String**.

Object **StringItem**

StringItem has the all the properties and methods of the **Item** object as well as the properties and methods defined below.

The **StringItem** object has the following properties:

value

This read-only property is of type **String**.

Object **IntegerItem**

IntegerItem has the all the properties and methods of the **Item** object as well as the properties and methods defined below.

The **IntegerItem** object has the following properties:

value

This read-only property is a **long** object.

Object **BooleanItem**

BooleanItem has the all the properties and methods of the **Item** object as well as the properties and methods defined below.

The **BooleanItem** object has the following properties:

value

This property is of type **Boolean**.

Object **ContentItem**

ContentItem has the all the properties and methods of the **Item** object as well as the properties and methods defined below.

The **ContentItem** object has the following properties:

node

This property is a **Node** object.

offset

This property is of type **Number**.

Object **VisualView**

The **VisualView** object has the following properties:

fontScheme

This read-only property is of type **String**.

width

This read-only property is of type **Number**.

height

This read-only property is of type **Number**.

horizontalDPI

This read-only property is of type **Number**.

verticalDPI

This read-only property is of type **Number**.

The **VisualView** object has the following methods:

createVisualCharacter()

This method returns a **VisualCharacter** object.

createVisualCharacterRun()

This method returns a **VisualCharacterRun** object.

createVisualFrame()

This method returns a **VisualFrame** object.

createVisualImage()

This method returns a **VisualImage** object.

createVisualFormButton()

This method returns a **VisualFormButton** object.

createVisualFormField()
This method returns a **VisualFormField** object.
select(boundary, offset, extend, add)
This method has no return value.
The **boundary** parameter is a **Node** object.
The **offset** parameter is of type **Number**.
The **extend** parameter is of type **Boolean**.
The **add** parameter is of type **Boolean**.
matchSegment(segment)
This method has no return value.
The **segment** parameter is a **VisualResource** object.

Object **VisualResource**

Object **VisualFont**

VisualFont has the all the properties and methods of the **VisualResource** object as well as the properties and methods defined below.
The **VisualFont** object has the following properties:
matchFontName
This property is of type **String**.
exists
This read-only property is of type **Boolean**.
fontName
This read-only property is of type **String**.
The **VisualFont** object has the following methods:
getNext()
This method returns a **Boolean**.

Object **VisualSegment**

VisualSegment has the all the properties and methods of the **VisualResource** object as well as the properties and methods defined below.
The **VisualSegment** object has the following properties:
matchPosition
This property is of type **Boolean**.
matchInside
This property is of type **Boolean**.
matchContaining
This property is of type **Boolean**.
matchX
This property is a **long** object.
matchY
This property is a **long** object.
matchXR
This property is a **long** object.
matchYR
This property is a **long** object.
matchContent
This property is of type **Boolean**.

matchRange
This property is of type **Boolean**.
matchNode
This property is a **Node** object.
matchOffset
This property is of type **Number**.
matchNodeR
This property is a **Node** object.
matchOffsetR
This property is of type **Number**.
matchContainsSelected
This property is of type **Boolean**.
matchContainsVisible
This property is of type **Boolean**.
exists
This read-only property is of type **Boolean**.
startNode
This read-only property is a **Node** object.
startOffset
This read-only property is of type **Number**.
endNode
This read-only property is a **Node** object.
endOffset
This read-only property is of type **Number**.
topOffset
This read-only property is a **long** object.
bottomOffset
This read-only property is a **long** object.
leftOffset
This read-only property is a **long** object.
rightOffset
This read-only property is a **long** object.
width
This read-only property is of type **Number**.
height
This read-only property is of type **Number**.
selected
This read-only property is of type **Boolean**.
visible
This read-only property is of type **Boolean**.
foregroundColor
This read-only property is of type **Number**.
backgroundColor
This read-only property is of type **Number**.
fontName
This read-only property is of type **String**.

fontHeight

This read-only property is of type **String**.

The **VisualSegment** object has the following methods:

getNext()

This method returns a **Boolean**.

Object **VisualCharacter**

VisualCharacter has the all the properties and methods of the **VisualSegment** object as well as the properties and methods defined below.

Object **VisualCharacterRun**

VisualCharacterRun has the all the properties and methods of the **VisualSegment** object as well as the properties and methods defined below.

Object **VisualFrame**

VisualFrame has the all the properties and methods of the **VisualSegment** object as well as the properties and methods defined below.

The **VisualFrame** object has the following properties:

embedded

This read-only property is a **VisualSegment** object.

Object **VisualImage**

VisualImage has the all the properties and methods of the **VisualSegment** object as well as the properties and methods defined below.

The **VisualImage** object has the following properties:

imageURL

This read-only property is of type **String**.

isLoaded

This read-only property is of type **Boolean**.

Object **VisualFormButton**

VisualFormButton has the all the properties and methods of the **VisualSegment** object as well as the properties and methods defined below.

The **VisualFormButton** object has the following properties:

isPressed

This read-only property is of type **Boolean**.

Object **VisualFormField**

VisualFormField has the all the properties and methods of the **VisualSegment** object as well as the properties and methods defined below.

The **VisualFormField** object has the following properties:

formValue

This read-only property is of type **String**.

References

For the latest version of any W3C specification please consult the list of W3C Technical Reports available at http://www.w3.org/TR.

D.1: Normative references

ECMAScript

ECMA (European Computer Manufacturers Association) ECMAScript Language Specification. Available at http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM

Java

Sun Microsystems Inc. The Java Language Specification, James Gosling, Bill Joy, and Guy Steele, September 1996. Available at http://java.sun.com/docs/books/jls

OMGIDL

OMG (Object Management Group) IDL (Interface Definition Language) defined in The Common Object Request Broker: Architecture and Specification, version 2.3.1, October 1999. Available at http://sisyphus.omg.org/technology/documents/formal/corba_2.htm

Index