this in more detail. eed to be updated to take into account the other pseudo-classes from that module.

Web Controls 1.0
Working Draft 8 November 2004

This version:

http://www.whatwg.org/specs/web-controls/current-work/

Latest version:

http://www.whatwg.org/specs/web-controls/current-work/

Editor:

Ian Hickson, Opera Software, ian@hixie.ch

Abstract

...

Status of this document

This is a work in progress! This document is changing on a daily if not hourly basis in response to comments and as a general part of its development process. Comments are very welcome, please send them to whatwg@whatwg.org. Thank you.

It is very wrong to cite this as anything other than a work in progress. Do not implement this in a production product. It is not ready yet! At all!

This document is the result of a loose collaboration between interested parties in the context of the Web hypertext application technology working group. To become involved in the development of this document, please send comments to the address given above. **Your input will be taken into consideration.**

This is a working draft and may therefore be updated, replaced or rendered obsolete by other documents at any time. It is inappropriate to use Working Drafts as reference material or to cite them as other than "work in progress".

This draft may contain namespaces that use the `data:` URI scheme. These are temporary and will be changed before this specification is ready to be implemented.

To find the latest version of this working draft, please follow the "Latest version" link above.

Table of contents

- 1. Introduction
  - 1.1. Relationship to Web Forms 2.0
  - 1.2. Relationship to Web Apps 1.0
  - 1.3. Relationship to XBL 2.0
  - 1.4. Relationship to CSS3 UI
  - 1.5. Conformance requirements
  - 1.6. Terminology
- 2. ...
  - 2.1. The 'appearance' property
  - 2.2. Pseudo-classes
  - 2.3. Focus
  - 2.4. Key handling
- 3. Properties for selection, focus, and editing
  - 3.1. Text Selection
  - 3.2. Focus
  - 3.3. Editing
  - 3.4. Changes to text-transform
  - 3.5. Changes to position
- 4. DOM Interfaces
  - 4.1. The DocumentUI Interface
  - 4.2. The ElementUI Interface
  - 4.3. Popup Display Events
  - 4.4. UI Exceptions
  - 4.5. The Window Interface
- References
- Acknowledgements

---

1. Introduction

...

1.1. Relationship to Web Forms 2.0

1.2. Relationship to Web Apps 1.0

1.3. Relationship to XBL 2.0

1.4. Relationship to CSS3 UI

The CSS3 UI specification [CSS3UI] introduces a number of properties suitable for Web-based application development. This specification expands on those properties and specifies their interaction with scripting-based environments and the DOM.

1.5. Conformance requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Diagrams, examples, and notes are non-normative. All other content in this specification is intended to be normative.

Documents that use the new features described in this specification using HTML over HTTP must be served as `text/html`. Documents that use the new features described in this specification using XHTML or other XML languages over HTTP must be served using an XML MIME type such as `application/xml` or `application/xhtml+xml`. [RFC3023]

(In other words, for the purposes of [RFC2854], documents conforming to this specification using its HTML formulation should be considered HTML documents, but documents using the XML formulation should not be considered HTML-compatible.)

1.6. Terminology

...

2. ...

...

2.1. The 'appearance' property

CSS3 UI [CSS3UI]

The exact appearance used will vary based on other properties and out of band data. In particular, the metadata and dynamic states described in the section on the ElementUI DOM interface directly affect how widgets are rendered. For example, a checkbox will look different based on whether the STATE_GROUP_CHECKED state is set to STATE_CHECKED_UNCHECKED, STATE_CHECKED_CHECKED, or STATE_CHECKED_INDETERMINATE. Progress bars, scroll bars and sliders (track bars) use the STATE_GROUP_VALUE metadata state to determine their position. The value 0 represents the lowest progress/scroll position/slider position, and the value 2^{16}-1 (65535) represents the highest progress/scroll position/slider position.

2.2. Pseudo-classes

The following dynamic pseudo-classes are defined in terms of state information that can be manipulated through the DOM.

:active

This pseudo-class matches the element that has its DYNAMIC_ACTIVE state set, if any.

:hover

This pseudo-class matches the element that has its DYNAMIC_HOVER state set, if any, as well as any ancestors of that element.

:focus

This pseudo-class matches the element which has focus, if any, as well as all its ancestors. See the `focus()` and `focusByMethod()` methods.

:open

This element matches all elements which have their DYNAMIC_OPEN state set.

:closed

This element matches all elements which have their DYNAMIC_OPEN state unset.

:enabled

This pseudo-class matches all elements that their STATE_GROUP_ENABLED state set to STATE_ENABLED_ENABLED, and whose ancestors (including those that are not focusable) all have their STATE_GROUP_ENABLED state set to STATE_ENABLED_ENABLED.

:disabled

This pseudo-class matches all elements that have either their STATE_GROUP_ENABLED state set to STATE_ENABLED_DISABLED or which have an ancestor whose STATE_GROUP_ENABLED state set to STATE_ENABLED_DISABLED (including ancestors that cannot be focused).

:checked

This element matches all elements which have their STATE_GROUP_CHECKED state set to STATE_CHECKED_CHECKED.

:unchecked

This element matches all elements which have their STATE_GROUP_CHECKED state set to STATE_CHECKED_UNCHECKED.

:indeterminate

This element matches all elements which have their STATE_GROUP_CHECKED state set to STATE_CHECKED_INDETERMINATE.

:selected

This element matches all elements which have their STATE_GROUP_SELECTED state set to STATE_CHECKED_SELECTED. This pseudo-class is unrelated to the similarly named ::selection pseudo-element, which applies to the text selection.

:unselected

This element matches all elements which have their STATE_GROUP_SELECTED state set to STATE_CHECKED_UNSELECTED.

:default

This pseudo-class matches elements that have their STATE_GROUP_DEFAULT state set to STATE_DEFAULT_DEFAULT.

:valid

This element matches all elements which have their STATE_GROUP_VALID state set to STATE_VALID_VALID.

:invalid

This element matches all elements which have their STATE_GROUP_VALID state set to

STATE_VALID_INVALID.

:required

This element matches all elements which have their STATE_GROUP_REQUIRED state set to STATE_REQUIRED_REQUIRED.

:optional

This element matches all elements which have their STATE_GROUP_REQUIRED state set to STATE_REQUIRED_OPTONAL.

:read-only

This element matches all elements which have their STATE_GROUP_DATA state set to exactly STATE_DATA_READABLE. Note that this is independent of the actual editable state of the control as presented to the user. The actual state of the control is governed by the computed value of the 'user-modify' property on a per-view basis, while the :read-only pseudo-class relates to the intended state of the control as set by the data model or other script.

:read-write

This element matches all elements which have their STATE_GROUP_DATA state set to exactly STATE_DATA_EDITABLE. Note that this is independent of the actual editable state of the control as presented to the user. The actual state of the control is governed by the computed value of the 'user-modify' property on a per-view basis, while the :read-write pseudo-class relates to the intended state of the control as set by the data model or other script.

:write-only

This element matches all elements which have their STATE_GROUP_DATA state set to exactly STATE_DATA_WRITABLE. Note that this is independent of the actual editable state of the control as presented to the user. The actual state of the control is governed by the computed value of the 'user-modify' property on a per-view basis, while the :write-only pseudo-class relates to the intended state of the control as set by the data model or other script.

See the ElementUI section for details explaining how to toggle these pseudo-classes.

2.3. Focus

In UAs that implement this module, the 'nav-index' property doesn't do anything directly. It is used by methods on the DocumentUI interface to determine the order used for sequential focus changing (what is commonly referred to as "tab order").

The value none means that the element is not part of the normal tab order, regardless of the value of 'user-can-focus'.

The value 'auto' means that the UA may decide where in the tab order the element goes.

The directional navigational properties ('nav-up' and so forth) similarly work in terms of the directional focus methods.

2.4. Key handling

The 'key-equivalent' property doesn't do anything directly. It is used by in XBL as a filter for key events. The idea is that if the window receives a key event that matches an element's key-equivalent, that element will be sent a separate event.

3. Properties for selection, focus, and editing

Unlike the UI module properties, which apply to non-scriptable environments as well as dynamic, scriptable UAs, these properties control features that are directly related to the DOM.

3.1. Text Selection

	'user-can-select'
Value:	never | always | only-with-modifier
Initial:	always
Applies To:	all elements and generated content
Inherited:	yes
Percentages:	n/a
Media:	visual
Computed Value:	specified value

A property to decide if the UA should allow inline selection. Note that this is a distinct concept from element selection -- an element can be flagged as selected independently of its contents being selected.

never

The contents of the element cannot be selected.

always

The contents of the element can be selected using the default UA-defined mechanism.

only-with-modifier

The contents of the element may only be selected using an alternate UA-defined selection mechanism.

For example, if by default a UA uses mouse dragging as a selection mechanism, then with 'user-can-select' is set to 'only-with-modifier' the same UA could change to mouse dragging with the shift key held down. The name 'only-with-modifier' is not meant to restrict the alternate mechanism, only to indicate that the UA should not enable the default selection mechanism.

When an elements' contents are selected (in part, in whole, or as part of a greater selection), that element contains a ::selection pseudo-element which applies to that selection.

User agents may treat the never value as only-with-modifier in untrusted documents, which typically would cover any document on the web, although the exact definition of 'trusted' is UA-defined and may be dependent on user preferences.

3.2. Focus

	'user-can-focus'
Value:	takes-focus | leaves-focus | resets-focus

Initial:	leaves-focus
Applies To:	all elements (but not pseudo-elements)
Inherited:	no
Percentages:	n/a
Media:	visual
Computed Value:	specified value

These properties decide if the UA should allow an element to gain focus.

takes-focus
: If an attempt is made to focus the element, the active focus is moved to the element.

leaves-focus
: If an attempt is made to focus the element, and none of the element's ancestors have their 'user-can-focus' property set to 'takes-focus', then nothing changes, otherwise focus is transferred to the nearest ancestor element with 'takes-focus' set.

resets-focus
: If an attempt is made to focus the element, and none of the element's ancestors have their 'user-can-focus' property set to 'takes-focus', then focus is reset to the window, otherwise focus is transferred to the nearest ancestor element with 'takes-focus' set.

In this definition, the **window** is a user-agent-defined construct, typically related to the viewport. When the window has focus, the user is typically able to scroll the document using arrow keys on a keyboard.

Note that this property has no *direct* effect on the :focus pseudo-class. This property merely determines whether an element can *gain* focus in a particular view, not whether it can *retain* focus.

'user-focus-select-pointer'

Value:	all | reset | ignore
Initial:	ignore
Applies To:	all elements with user-can-focus: takes-focus and user-can-select: always
Inherited:	no
Percentages:	n/a
Media:	visual
Computed Value:	specified value

'user-focus-select-key'

Value:	all | reset | ignore
Initial:	ignore
Applies To:	all elements with user-can-focus: takes-focus and user-can-select: always
Inherited:	no
Percentages:	n/a
Media:	visual
Computed Value:	specified value

'user-focus-select-unknown'

Value:	all | reset | ignore

Initial:	ignore
Applies To:	all elements with user-can-focus: takes-focus and user-can-select: always
Inherited:	no
Percentages:	n/a
Media:	visual
Computed Value:	specified value

	'user-focus-select'
Value:	all | reset | ignore
Initial:	not defined for shorthand properties
Applies To:	see prose
Inherited:	no
Percentages:	n/a
Media:	visual
Computed Value:	specified value

These properties explain what part of the contents of the element should be selected when focussing the element.

all
: If the element is focused, all its contents should be selected.

reset
: If the element is focused, then the selection is reset.

ignore
: Nothing happens to the selection is the element is focused.

The property 'user-focus-select-key' applies when the focus change was requested via a keyboard event, the property 'user-focus-select-pointer' applies when the focus change was requested via a pointer event, and the property 'user-focus-select-unknown' applies in other cases. The property 'user-focus-select' can be used as a shorthand for setting the other two properties.

3.3. Editing

	'user-modify'
Value:	read-only | all | [add-text || edit-text || add-elements || move-elements || attributes || deletable]
Initial:	read-only
Applies To:	all elements that can obtain focus
Inherited:	yes
Percentages:	n/a
Media:	interactive
Computed Value:	specified value

This property establishes whether the UA should allow editing of the node and its children.

read-only
: The element should not be editable by the user (unless overriden by the UA, e.g. if the user invokes an "edit" mode on the document).

all
: Any child node may be edited and the element itself may be deleted. This is equivalent to specifying all the other flags together.

children
: Equivalent to add-text edit-text add-elements move-elements

add-text
: New text nodes may be added to the element and edited.

edit-text
: Any existing child text and CDATA nodes of the element may be edited.

add-elements
: Child elements may be inserted.

move-elements
: Child elements may be moved.

attributes
: The attributes of the element may be added, changed, and removed.

deletable
: The element itself may be deleted.

This property is inherited, but only applies to editing the element itself and its immediate children. The following makes an element `foo` and its children all completely editable:

```
foo, foo * { user-modify: all; }
```

That would also allow the element itself to be deleted, however. In order to disallow that, the following rules would be used:

```
foo { user-modify: children; }
foo * { user-modify: all; }
```

The exact mechanisms used to enable editing is up to the UA.

3.4. Changes to text-transform

The 'text-transform' property is modified to allow a string value to be used. When a string value is specified, all characters in the string are replaced with this string. For example:

```
input[type=password] { text-transform: '*'; }
```

This would replace each character in an input field with an asterisk, as seen in many existing web browsers.

3.5. Changes to position

The value popup is added to the 'position' property. When this value is applied to an element, the following changes apply:

1. The element is taken out of flow and not shown (same as display:none).
2. If the element's 'overflow' property has a specified value of visible then the computed value becomes hidden. Other values stay unaffected.

The ElementUI interface can then be used to actually trigger the popup.

4. DOM Interfaces

4.1. The DocumentUI Interface

The `DocumentUI` interface contains methods for moving focus around the document.

IDL Definition

```
interface DocumentUI {
  void moveFocusForward()
  void moveFocusBackward()
  void moveFocusUp()
  void moveFocusRight()
  void moveFocusDown()
  void moveFocusLeft()
};
```

Properties

Future drafts of this specification will provide properties for the current default element and the current focused element.

Methods

`moveFocusForward`
: This uses 'nav-index' and 'user-can-focus' properties to find the next focussable element and focusses it.

`moveFocusBackward`
: This uses 'nav-index' and 'user-can-focus' properties to find the previous focussable element and focusses it.

`moveFocusUp`
: This uses 'nav-up' and 'user-can-focus' properties to find the next focussable element and focusses it.

`moveFocusLeft`
: This uses 'nav-left' and 'user-can-focus' properties to find the next focussable element and focusses it.

`moveFocusDown`
: This uses 'nav-down' and 'user-can-focus' properties to find the next focussable element and focusses it.

`moveFocusRight`
: This uses 'nav-right' and 'user-can-focus' properties to find the next focussable element and focusses it.

4.2. The ElementUI Interface

The `ElementUI` interface contains methods for displaying popups and toggling the state of dynamic pseudo-classes.

IDL Definition

```
interface ElementUI {

  /* POPUPS */
  const unsigned short      BEFORE_START                   = 0;
  const unsigned short      BEFORE_END                     = 1;
  const unsigned short      AFTER_START                    = 2;
  const unsigned short      AFTER_END                      = 3;
  const unsigned short      START_BEFORE                   = 4;
  const unsigned short      START_AFTER                    = 5;
  const unsigned short      END_BEFORE                     = 6;
  const unsigned short      END_AFTER                      = 7;
  const unsigned short      OVERLAP                        = 8;
  const unsigned short      AFTER_POINTER                  = 9;

  void showPopup(in unsigned short alignment,
                 in Element target, in Element anchor);

  void hidePopup();



  /* METADATA STATE */
  /* Metadata states are the same in all views */
  const unsigned short      STATE_GROUP_ENABLED            = 0; //
  const unsigned short      STATE_GROUP_DEFAULT            = 1; //
  const unsigned short      STATE_GROUP_CHECKED            = 2; //
  const unsigned short      STATE_GROUP_SELECTED           = 3; //
  const unsigned short      STATE_GROUP_VALID              = 4; //
  const unsigned short      STATE_GROUP_REQUIRED           = 5; //
  const unsigned short      STATE_GROUP_DATA               = 6; //
  const unsigned short      STATE_GROUP_VALUE              = 10; //

  const unsigned short      STATE_ENABLED_DISABLED         = 0; //
  const unsigned short      STATE_ENABLED_ENABLED          = 1; //

  const unsigned short      STATE_DEFAULT_NORMAL           = 0; //
  const unsigned short      STATE_DEFAULT_DEFAULT          = 1; //

  const unsigned short      STATE_CHECKED_UNCHECKED        = 0; //
  const unsigned short      STATE_CHECKED_CHECKED          = 1; //
  const unsigned short      STATE_CHECKED_INDETERMINATE    = 2;

  const unsigned short      STATE_SELECTED_UNSELECTED      = 0; //
  const unsigned short      STATE_SELECTED_SELECTED        = 1; //

  const unsigned short      STATE_VALID_INVALID            = 0; //
  const unsigned short      STATE_VALID_VALID              = 1; //

  const unsigned short      STATE_REQUIRED_OPTIONAL        = 0; //
  const unsigned short      STATE_REQUIRED_REQUIRED        = 1; //

  const unsigned short      STATE_DATA_INACCESSIBLE        = 0; //
  const unsigned short      STATE_DATA_READABLE            = 1; //
  const unsigned short      STATE_DATA_WRITABLE            = 2; //
  const unsigned short      STATE_DATA_EDITABLE            = 3; //

  void setMetadataState(in unsigned short state, in long value);
  long getMetadataState(in unsigned short state);


  /* DYNAMIC STATE */
  /* Dynamic states are defined on a per-view basis */
  const unsigned short      DYNAMIC_ACTIVE                 = 0;
  const unsigned short      DYNAMIC_HOVER                  = 1;
  const unsigned short      DYNAMIC_OPEN                   = 2;

  void setDynamicState(in unsighed short state, in bool value);
  bool getDynamicState(in unsigned short state);


  /* FOCUS */
  const unsigned short      FOCUSED_BY_UNKNOWN            = 0;
  const unsigned short      FOCUSED_BY_KEYBOARD           = 1;
  const unsigned short      FOCUSED_BY_POINTER            = 2;

  void focus();

  void focusByMethod(in unsigned short method);
};
```

Methods

`showPopup`

Shows the element as a popup. If the element doesn't have the 'position' property set to popup, this will raise a NOT_A_POPUP_ERR. Calling this method will cause the following events: First, a popupShowing event is triggered (it is cancellable, cancelling that event will prevent the popup from showing), second, the popup is aligned (potentially triggering a popupPositioning event), and finally a popupShown event is triggered.

Parameters

`alignment` of type `unsigned short`

One of the alignment constants. See the descriptions below.

`target` of type `Element`

The element that the popup will be positioned relative to (it may itself be part of a popup, for example this is the case for cascading menus). If the `target` is `null`, then the point representing the (last known) position of the pointing device is used instead. If the `target` element is not visible, or if there is it is `null` but there is no pointing device, then the viewport is used instead. If the viewport is not visible, then the whole screen is used.

This ensures that the popup will appear, regardless of how it was triggered.

`anchor` of type `Element`

The element to position relative to the target. This must be either the popup itself, or one of the popup's descendants (excluding anonymous nodes that are not in the default view). If `null` is used, then that implies the element. If the `anchor` has no defined position (which can happen if is hidden using CSS for example) then its closest ancestor which will have a position when the popup is shown is used instead.

No Return Value

Exceptions

`UIException` NOT_A_POPUP_ERR

Raised if the element doesn't have the 'position' property set to popup.

`UIException` HIERARCHY_ERR

Raised if the `anchor` is not a descendant of the element or if it is an annoymous descendant that is not in the default view.

`hidePopup`

Hides the popup associated with the element, if any.

No Parameters

No Return Value

No Exceptions

`setMetadataState`

Sets various flags relating to the logical state of the element. The different flags have different semantics. See the list of constants below.

Parameters

`state` of type `unsigned short`

This parameter is a flag indicating which metadata state should be changed.

`value` of type `long`

What value to give the state in question.

No Return Value

No Exceptions

`getMetadataState`

Retrieves various flags relating to the logical state of the element. The different flags have different semantics. See the list of constants below.

Parameters

`state` of type `unsigned short`

This parameter is a flag indicating which metadata state should be retrieved.

Return Value

`long`

The current value for the specified state.

No Exceptions

`setDynamicState`

Sets various flags relating to the dynamic state of the element (in the default view). The different flags have different semantics. See the list of constants below.

Parameters

`state` of type `unsigned short`

This parameter is a flag indicating which dynamic state should be changed.

`value` **of type** `boolean`

What value to give the state in question.

No Return Value

No Exceptions

`getDynamicState`

Retrieves various flags relating to the dynamic state of the element in the current view. The different flags have different semantics. See the list of constants below.

Parameters

`state` **of type** `unsigned short`

This parameter is a flag indicating which dynamic state should be retrieved.

Return Value

`long`

The current value for the specified state.

No Exceptions

`focus`

Same as calling focusByMethod(FOCUS_BY_UNKNOWN).

No Parameters

No Return Value

No Exceptions

`focusByMethod`

Focusses the element, using the properties appropriate for the method specified to select the content of the element.

Parameters

`method` **of type** `unsigned short`

This parameter is a flag indicating which method was used to focus the element.

No Return Value

No Exceptions

Defined Constants For Popups

The first eight of these constants use a consistent naming scheme based on the writing-mode independent "BASE" system (Before After Start End, which map, in a left-to-right top-to-bottom writing mode such as English, to the Top Bottom Left and Right sides respectively). The first word represents the side of the target which should be adjacent to the opposite side of the anchor, and the second word represents the sides of the target and anchor which should be aligned so as to be parallel to each other.

`BEFORE_START`

The "before" side of the target is made adjacent to the "after" side of the anchor, and the "start" sides of the target and anchor are aligned.

`BEFORE_END`

The "before" side of the target is made adjacent to the "after" side of the anchor, and the "end" sides of the target and anchor are aligned.

`AFTER_START`

The "after" side of the target is made adjacent to the "before" side of the anchor, and the "start" sides of the target and anchor are aligned.

`AFTER_END`
: The "after" side of the target is made adjacent to the "before" side of the anchor, and the "end" sides of the target and anchor are aligned.

`START_BEFORE`
: The "start" side of the target is made adjacent to the "end" side of the anchor, and the "before" sides of the target and anchor are aligned.

`START_AFTER`
: The "start" side of the target is made adjacent to the "end" side of the anchor, and the "after" sides of the target and anchor are aligned.

`END_BEFORE`
: The "end" side of the target is made adjacent to the "start" side of the anchor, and the "before" sides of the target and anchor are aligned.

`END_AFTER`
: The "end" side of the target is made adjacent to the "start" side of the anchor, and the "after" sides of the target and anchor are aligned.

`OVERLAP`
: The before/start corners of the target and anchor elements are overlapped so that (if they have the same size) they appear over each other.

`AFTER_POINTER`
: The "after" side of the target is made adjacent to the "before" side of the anchor, and the "start" side of the anchor is aligned with the last known position of the pointing device, if any, or with the "start" side of the target, if there is no pointing device.

Special Cases

If the `target` is `null` and there is a pointing device, then the last known position of the pointer is used as the target in the cases above. So for example, if `alignment` is set to `BEFORE_START` or `END_AFTER`, then the before/start corner of the popup is positioned at the pointer.

When the target is a point and not a region, the following constants become equivalent:

- BEFORE_START and END_AFTER
- BEFORE_END and START_AFTER
- AFTER_START, END_BEFORE, OVERLAP and AFTER_POINTER
- AFTER_END and START_BEFORE

If the popup is smaller than the screen, then the popup *must* appear completely on the screen. If, after aligning the popup as specified by the arguments to the showPopup method, the popup would overflow the screen, then the following algorithm should be used. The algorithm should be followed step by step until the popup fits on the screen.

1. If the popup is wider (taller) than the screen, then its outer margin width (height) should be set to the width (height) of the screen (the `'overflow'` property applies).

2. If the alignment is `OVERLAP` and the anchor is not the popup element itself, then position the anchor as required, then shift the popup such that it touches the sides of the screen where the anchor overflows the screen (if that is in only one direction, the other axis therefore remains unaffected), and apply the `'overflow'` property at that scroll position.

3. Otherwise, some as-yet-undecided algorithm is used to try each alignment value in turn, and if no possibility is found, then the popup menu is positioned on the side that had the most room, and then the dimensions of the popup are changed so that (from drop downs) it touches the relevant side of the screen or (for side opening popups) it is moved so that it touches one of the sides.

Defined Constants For Metadata State

`STATE_GROUP_ENABLED`

`STATE_ENABLED_DISABLED`

`STATE_ENABLED_ENABLED`

These constants are used to enable and disable user interface elements. Every element is either STATE_ENABLED_ENABLED or STATE_ENABLED_DISABLED. See the :enabled and :disabled pseudo-classes. Elements all initially start with their STATE_GROUP_ENABLED state set to STATE_ENABLED_ENABLED.

`STATE_GROUP_DEFAULT`

`STATE_DEFAULT_NORMAL`

`STATE_DEFAULT_DEFAULT`

These constants are used to toggle the default element of a document. Only one element per document may be the default at any one time. Setting an element to STATE_DEFAULT_DEFAULT automatically resets any other element to the STATE_DEFAULT_NORMAL state. See the :default pseudo-class. Elements all initially start with their STATE_GROUP_DEFAULT state set to STATE_DEFAULT_NORMAL.

`STATE_GROUP_CHECKED`

`STATE_CHECKED_UNCHECKED`

`STATE_CHECKED_CHECKED`

`STATE_CHECKED_INDETERMINATE`

These constants are used to toggle the value of check box and radio button interface elements. Every element has one of these values. See the :checked, :unchecked and :indeterminate pseudo-classes. Elements all initially start with their STATE_GROUP_CHECKED state set to STATE_CHECKED_UNCHECKED.

`STATE_GROUP_SELECTED`

`STATE_SELECTED_UNSELECTED`

`STATE_SELECTED_SELECTED`

These constants are used to determine the selection state of entire elements. (Note: This is distinct from text selection and focus.) Every element is either STATE_SELECTED_SELECTED or STATE_SELECTED_UNSELECTED. See the :selected and :unselected pseudo-classes. Elements all initially start with their STATE_GROUP_SELECTED state set to STATE_SELECTED_UNSELECTED.

`STATE_GROUP_VALID`

`STATE_VALID_INVALID`

`STATE_VALID_VALID`

These constants are used to determine the validity state of entire elements. For example, the XForms data model can place constraints on the possible values of controls in a form. Every element is either STATE_VALID_VALID or STATE_VALID_INVALID. See the :valid and :invalid pseudo-classes. Elements all initially start with their STATE_GROUP_VALID state set to STATE_VALID_VALID.

`STATE_GROUP_REQUIRED`

`STATE_REQUIRED_OPTIONAL`

`STATE_REQUIRED_REQUIRED`

These constants are used to determine whether entire elements are considered optional or required. For example, the XForms data model can require that fields be filled before allowing a form to be submitted. Every element is either STATE_REQUIRED_REQUIRED or STATE_REQUIRED_OPTIONAL. See the :required and :optional pseudo-classes. Elements all initially start with their STATE_GROUP_REQUIRED state set to STATE_REQUIRED_OPTIONAL.

`STATE_GROUP_DATA`

`STATE_DATA_INACCESSIBLE`

`STATE_DATA_READABLE`

`STATE_DATA_WRITABLE`

`STATE_DATA_EDITABLE`

These constants are used to determine the intended state of elements with respect to their content being edited. Note, however, that this state causes absolutely nothing to change with respect to elements' actual editing state! Typically, this state will be set by the data model being used, and selectors will then be used to change the value of the 'user-modify' property of a pseudo-element. Every element is in a logical bitwise addition of zero, one, or both of STATE_DATA_READABLE and STATE_DATA_WRITABLE. Two additional constants are provided for convenience: STATE_DATA_EDITABLE (both values set) and STATE_DATA_INACCESSIBLE (neither value set). See the :read-only, :write-only and :read-write pseudo-classes. Elements all initially start with their STATE_GROUP_DATA state set to STATE_DATA_READABLE.

`STATE_GROUP_VALUE`

This constant is used to access the generic integer value of user interface elements. Every element has such a value, although it is only typically used by scroll bars, track bars and progress bars. Elements all initially start with their STATE_GROUP_VALUE state set to zero.

Defined Constants For Dynamic State

`DYNAMIC_ACTIVE`

This constant is used to toggle the active state of elements. Only one element per view may be active at any one time. Setting an element to active automatically resets any other active element to its non-active state. See the :active pseudo-class.

`DYNAMIC_HOVER`

This constant is used to toggle the hover state of elements. Only one element per view may be in the hover state at any one time. Setting an element's hover state to true automatically resets any other element's hover state. See the :hover pseudo-class. User agents may, in addition to changes triggered using the setDynamicState() method, automatically set this state on elements in response to user events (such as moving the mouse).

`DYNAMIC_OPEN`

This constant is used to toggle the open/closed state of elements. Every element has an open/closed state for each view. Toggling the open/closed state does not affect other elements or other views. (Note: At the moment there is no way to change the open/closed state of elements in any view other than the default view.) See the :open and :closed pseudo-classes.

Defined Constants For Focus

`FOCUSED_BY_UNKNOWN`

An unspecified method was used to focus the element.

`FOCUSED_BY_KEYBOARD`

The element was focused by some keyboard action.

`FOCUSED_BY_MOUSE`
The element was focused by the pointing device.

4.3. Popup Display Events

Four new events are introduced, all related to popups. These use the base DOM Event interface to pass contextual information. The different types of such events that can occur are:

popupShowing
The popupShowing event occurs when the `popupShow` method is called. This event is only valid for elements whose 'position' property has the value popup. If the event is cancelled, then the popup will not be shown.

- Bubbles: No
- Cancelable: Yes
- Context Info: None

popupShown
The popupShown event occurs after a popup has been displayed. This event is only valid for elements whose 'position' property has the value popup.

- Bubbles: No
- Cancelable: No
- Context Info: None

popupHiding
The popupHiding event occurs when the `popupHide` method is called. This event is only valid for elements whose 'position' property has the value popup and which are currently displayed. If the event is cancelled, then the popup will remain visible.

- Bubbles: No
- Cancelable: Yes
- Context Info: None

popupHidden
The popupHidden event occurs after a popup has been hidden (also referred to as cancelling the popup). This event is only valid for elements whose 'position' property has the value popup.

- Bubbles: No
- Cancelable: No
- Context Info: None

4.4. UI Exceptions

This section defines some error codes used by the ElementUI interface.

IDL Definition

```
exception UIException {
  // UIExceptionCode
  const unsigned short      NOT_A_POPUP_ERR                = 1;
  const unsigned short      HIERARCHY_ERR                  = 2;

  unsigned short   code;
};
```

Defined Constants

`NOT_A_POPUP_ERR`
If the element does not have the 'position' property set to popup.

`HIERARCHY_ERR`
If the element specified is not a child of the popup element.

4.5. The Window Interface

This section will define some aspects of the DOM Level 0 "window" interface. In particular, window.open (with all its flags), resizing, possibly window.location and other features.

A great emphasis should be placed here on backwards compatibility. Only the parts that are needed for UI should be defined here.

The Window interface only applies to the default view. There is currently no defined way of obtaining the window interface for another view.

References

[CGI]

The CGI Specification. NCSA HTTPd Development Team, November 1995. The CGI Specification is available at http://hoohoo.ncsa.uiuc.edu/cgi/interface.html

[CSJSR]

Client-Side JavaScript Reference (1.3). Netscape Communications Corporation, May 1999. The Client-Side JavaScript Reference (1.3) is available at http://devedge.netscape.com /library/manuals/2000/javascript/1.3/reference/index.html

[CHARMOD]

Character Model for the World Wide Web 1.0, M. Dürst, F. Yergeau, R. Ishida, M. Wolf, T. Texin. W3C, August 2003. The latest version of the Character Model specification is available at http://www.w3.org/TR/charmod/

[CSS21]

CSS 2.1 Specification, B. Bos, T. Çelik, I. Hickson, H. Lie. W3C, September 2003. The latest version of the CSS 2.1 specification is available at http://www.w3.org/TR/CSS21

[CSS3UI]

CSS3 Basic User Interface Module, T. Çelik. W3C, July 2003. The latest version of the CSS3

UI module is available at http://www.w3.org/TR/css3-ui

[CSS3CONTENT]

CSS3 Generated and Replaced Content Module, I. Hickson. W3C, May 2003. The latest version of the CSS3 Generated and Replaced Content module is available at http://www.w3.org/TR/css3-content

[DOM3CORE]

Document Object Model (DOM) Level 3 Core Specification, A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne. W3C, November 2003. The latest version of the DOM Level 3 Core specification is available at http://www.w3.org/TR/DOM-Level-3-Core/

[DOM3EVENTS]

Document Object Model (DOM) Level 3 Events Specification, P. Le Hégaret, T. Pixley. W3C, November 2003. The latest version of the DOM Level 3 Events specification is available at http://www.w3.org/TR/DOM-Level-3-Events/

[DOM2HTML]

Document Object Model (DOM) Level 2 HTML Specification, J. Stenback, P. Le Hégaret, A. Le Hors. W3C, January 2003. The latest version of the DOM Level 2 HTML specification is available at http://www.w3.org/TR/DOM-Level-2-HTML/

[DOM3LS]

Document Object Model (DOM) Level 3 Load and Save Specification, J. Stenback, A. Heninger. W3C, November 2003. The latest version of the DOM Level 3 Load and Save specification is available at http://www.w3.org/TR/DOM-Level-3-LS/

[ECMA262]

ECMAScript Language Specification, Third Edition. ECMA, December 1999. This version of the ECMAScript Language is available at http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM

[HTC]

HTML Components, Chris Wilson. Microsoft, September 1998. The HTML Components submission is available at http://www.w3.org/TR/1998/NOTE-HTMLComponents-19981023

[HTML4]

HTML 4.01 Specification, D. Raggett, A. Le Hors, I. Jacobs. W3C, December 1999. The latest version of the HTML4 specification is available at http://www.w3.org/TR/html4

[ISO8601]

ISO8601:2000 Data elements and interchange formats -- Information interchange -- Representation of dates and times. ISO, December 2000. ISO8601 is available for purchase at http://www.iso.ch/

[RFC959]

File Transfer Protocol (FTP), J. Postel, J. Reynolds. IETF, October 1985. RFC959 is available at http://www.ietf.org/rfc/rfc959

[RFC2119]

Key words for use in RFCs to Indicate Requirement Levels, S. Bradner. IETF, March 1997. RFC2119 is available at http://www.ietf.org/rfc/rfc2119

[RFC1738]

Uniform Resource Locators (URL), T. Berners-Lee, L. Masinter, M. McCahill. IETF, Decembed

1998. RFC1738 is available at http://www.ietf.org/rfc/rfc1738

[RFC2368]

The mailto URL scheme, P. Hoffman, L. Masinter, J. Zawinski. IETF, July 1998. RFC2368 is available at http://www.ietf.org/rfc/rfc2368

[RFC2387]

The "data" URL scheme, L. Masinter. IETF, August 1998. RFC2387 is available at http://www.ietf.org/rfc/rfc2387

[RFC2396]

Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter. IETF, August 1998. RFC2396 is available at http://www.ietf.org/rfc/rfc2396

[RFC2616]

Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. IETF, June 1999. RFC2616 is available at http://www.ietf.org/rfc/rfc2616

[RFC2806]

URLs for Telephone Calls, A. Vaha-Sipila. IETF, April 2000. RFC2806 is available at http://www.ietf.org/rfc/rfc2806

[RFC2822]

Internet Message Format, P. Resnick. IETF, April 2001. RFC2822 is available at http://www.ietf.org/rfc/rfc2822

[RFC2854]

The 'text/html' Media Type, D. Connolly, L. Masinter. IETF, June 2000. RFC2854 is available at http://www.ietf.org/rfc/rfc2854

[RFC3023]

XML Media Types, M. Murata, S. St.Laurent, D. Kohn. IETF, January 2001. RFC 3023 is available at http://www.ietf.org/rfc/rfc3023

[RFC3106]

ECML v1.1: Field Specifications for E-Commerce, D. Eastlake, T Goldstein. IETF, April 2001. RFC 3106 is available at http://www.ietf.org/rfc/rfc3106

[WF2]

Web Forms 2.0, I. Hickson. Opera Software, February 2004. The latest version of the Web Forms 2.0 proposed specification is available at http://www.hixie.ch/specs/html/forms/web-forms

[XBL]

XML Binding Language, David Hyatt. Mozilla, February 2001. The XBL submission is available at http://www.w3.org/TR/2001/NOTE-xbl-20010223/

[XML]

Extensible Markup Language (XML) 1.0 (Second Edition), T Bray, J Paoli, C. M. Sperberg-McQueen, E. Maler. W3C, October 2000. The latest version of the XML specification is available at http://www.w3.org/TR/REC-xml/

[XHTML1]

XHTML™ 1.1 - Module-based XHTML, M. Altheim, S. McCarron. W3C, May 2001. The latest version of the XHTML 1.1 specification is available at http://www.w3.org/TR/xhtml11

[XHTML2]
XXX

[XForms]
XForms 1.0, M. Dubinko, L. Klotz, R. Merrick, T. Raman. W3C, October 2003. The latest version of the XForms specification is available at http://www.w3.org/TR/xforms

Acknowledgements

Thanks to Håkon Wium Lie, David Hyatt and Brendan Eich for their important support. Thanks to Matthew Mastracci and Steven Garrity for their useful and substantial comments.