Doc. A/100-2
09 March 2003

DTV APPLICATION SOFTWARE ENVIRONMENT LEVEL 1 (DASE-1)
PART 2: DECLARATIVE APPLICATIONS AND ENVIRONMENT

ATSC Standard

Note that this documents is past the customary 5-year review point. No update of the document is in process.

Advanced Television Systems Committee
1750 K Street, N.W.
Suite 1200
Washington, D.C. 20006
www.atsc.org

Blank Page

Table of Contents

DASE-1 DECLARATIVE APPLICATIONS AND ENVIRONMENT 1
1. SCOPE 1
1.1 Status 1
1.2 Purpose 1
1.3 Application 1
1.4 Organization 1
2. REFERENCES 3
2.1 Normative References 3
2.2 Informative References 4
2.3 Reference Acquisition 5
3. DEFINITIONS 6
3.1 Conformance Keywords 6
3.2 Acronyms and Abbreviations 6
3.3 Terms 6
4. BEHAVIOR 7
4.1 Application Processing 7
4.2 Application Decoding 7
4.3 Application Presentation 7
4.3.1 Display Configuration 8
4.4 Application Interaction 8
4.5 Trigger Processing 9
4.5.1 Event Processing 9
4.5.1.1 *bubbles* attribute 9
4.5.1.2 *cancelable* attribute 9
4.5.1.3 *target* attribute 9
4.5.2 Script Event Processing 9
4.6 Relative Identifier Resolution 10
4.7 Ecmascript Identifier Scheme 10
5. FACILITIES 12
5.1 Markup Content 12
5.1.1 application/xhtml+xml 12
5.1.1.1 Well Formedness 12
5.1.1.2 Validity 13
5.1.1.3 XML Declaration 13
5.1.1.4 Document Type Declaration 14
5.1.1.5 Attribute Semantics 14
5.1.1.5.1 Universal Resource Identifiers (URIs) 14
5.1.1.5.2 *name* attribute 15
5.1.1.5.3 *style* attribute 15
5.1.1.5.4 *xml:base* attribute 16
5.1.1.5.5 *xml:lang* attribute 16
5.1.1.6 Element Semantics 16
5.1.1.6.1 *a* (anchor) element 16
5.1.1.6.2 *base* element 19
5.1.1.6.3 *form* element 19
5.1.1.6.4 *img* element 19
5.1.1.6.5 *input* element 19
5.1.1.6.6 *link* element 20
5.1.1.6.7 *meta* element 20
5.1.1.6.8 *object* element 21
5.1.1.6.9 *script* element 23
5.1.1.6.10 *style* element 24
5.1.1.7 External Stylesheets 24
5.2 Stylesheet Content 24

5.2.1 text/css ... 24
5.2.1.1 Validity ... 25
5.2.1.2 Selectors ... 25
5.2.1.3 Character Set Rules ... 26
5.2.1.4 Import Rules ... 26
5.2.1.5 Font Face Rules ... 26
5.2.1.5.1 'src' descriptor ... 27
5.2.1.6 Page Rules ... 27
5.2.1.7 Media Types ... 27
5.2.1.7.1 'atsc-tv' media type ... 27
5.2.1.8 Style Properties ... 28
5.2.1.8.1 Alternative Semantic Interpretations ... 28
5.2.1.8.2 Standard Property Values ... 30
5.2.1.8.3 Non-Standard Properties ... 30
5.2.1.8.4 Non-Standard Property Values ... 32
5.2.1.9 Inline Stylesheets ... 32
5.3 Script Content ... 33
5.3.1 text/ecmascript ... 33
5.3.1.1 Native Objects ... 33
5.3.1.1.1 Non-Instantiable Built-In Objects ... 33
5.3.1.1.2 Instantiable Built-In Objects ... 34
5.3.1.2 Host Objects ... 37
5.3.1.2.1 Core Module Objects ... 38
5.3.1.2.2 Core Extension (XML) Module Objects ... 42
5.3.1.2.3 HTML Module Objects ... 43
5.3.1.2.4 Views Module Objects ... 55
5.3.1.2.5 StyleSheets Module Objects ... 58
5.3.1.2.6 CSS Module Objects ... 59
5.3.1.2.7 Events Module Objects ... 59
5.3.1.2.8 Event Set Module Objects ... 59
5.3.1.2.9 Environment Module Objects ... 65
ANNEX A. DOCUMENT TYPE DEFINITIONS ... 78
A.1 XDML Document Type ... 78
A.1.1 DTD Driver ... 78
A.1.2 Document (Content) Model ... 81
ANNEX B. DEFAULT STYLESHEET ... 85
B.1 Stylesheet ... 85
B.2 Default Fonts ... 86
ANNEX C. DOCUMENT OBJECT MODEL INTERFACES ... 87
C.1 Native Extensions Module ... 87
C.2 Core Extensions Module ... 87
C.3 Environment Extensions Module ... 88
C.4 Events Extensions Module ... 89
C.5 HTML Extensions Module ... 92
C.6 Views Extensions Module ... 93
ANNEX D. DOCUMENT OBJECT MODEL BINDINGS ... 95
D.1 Native Extensions Module Bindings ... 95
D.2 Core Extensions Module Bindings ... 95
D.3 Environment Extensions Module Bindings ... 96
D.4 Events Extensions Module Bindings ... 98
D.5 HTML Extensions Module Bindings ... 104
D.6 Views Extensions Module Bindings ... 106
ANNEX E. UNSUPPORTED HTML FEATURES ... 107
E.1 Syntax ... 107
E.1.1 Case Insensitivity ... 107

E.1.2 Markup Minimization 107
E.1.3 Reference End 109
E.2 Semantics 109
E.2.1 Constrained Attribute Interpretation 109
E.2.2 Unsupported Attributes 110
E.2.3 Constrained Element Interpretation 110
E.2.4 Unsupported Elements 111
ANNEX F. TRANSCODING HTML TO XDML 112
F.1 Syntax 112
F.1.1 Character Encodings 112
F.1.2 Comments 113
F.1.3 Processing Instructions 113
F.1.4 Document Type Declaration 113
F.1.5 Elements 113
F.1.6 Character References 113
F.1.7 Marked Sections 113
F.2 Semantics 114
F.2.1 Common Attributes 114
F.2.2 Elements 114
ANNEX G. ACCESSIBILITY CONSIDERATIONS 134
ANNEX H. EXAMPLES OF OBJECT ELEMENT USAGE 135
H.1 Active Object Content Example 135
H.2 Trigger Content Object Element Example 135
CHANGES 136
Changes from Candidate Standard to Standard 136

Table of Figures

Figure 1 Display Reference Model 56

Table of Tables

Table 1 CSS Selector Support 25
Table 2 CSS Font Face Rule Support 26
Table 3 CSS Property Support 28
Table 4 CSS Relative Font Size Mappings 30
Table 5 ECMAScript Non-Instantiable Native Object Support 33
Table 6 ECMAScript Instantiable Native Object Support 34
Table 7 Legacy Font Size Attribute Mappings 35
Table 8 DOM Core Interface Support 38
Table 9 DOM Core Extensions Interface Support 43
Table 10 DOM HTML Interface Support 43
Table 11 DOM Views Interface Support 55
Table 12 DOM Stylesheet Interface Support 59
Table 13 DOM Cascading Stylesheet Interface Support 59
Table 14 DOM Event Interface Support 59
Table 15 DOM Event Set Interface Support 60
Table 16 DOM Environment Interface Support 66
Table 17 List Element Type Attribute Mapping 126
Table 18 Ordered List Element Type Attribute Mapping 128
Table 19 Changes from Candidate Standard 136

DASE-1 Declarative Applications and Environment

ATSC Standard

1. SCOPE

1.1 Status

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 by the ATSC.

This specification is an ATSC Standard, having passed ATSC Member Ballot on September 16, 2002. This document is an editorial revision of the Approved Proposed Standard (PS/100-2) dated November 5, 2002.

The ATSC believes that this specification is stable, that it has been substantially demonstrated in independent implementations, and that it defines criteria that are necessary for effective implementation and interoperability of Advanced Television Systems. A list of cumulative changes made to this specification may be found at the end of this document.

A list of current ATSC Standards and other technical documents can be found at http://www.atsc.org/standards.html.

1.2 Purpose

This specification defines an architecture and a collection of facilities by means of which declarative applications may be delivered in an ATSC data broadcast service to a user agent embodied by a compliant receiver.[1]

A declarative application is an organization of information which primarily uses declarative as opposed to procedural mechanisms to express its information content. An example of a declarative application is a multimedia document composed of markup, style rules, scripts, and embedded graphics, video, and audio.

A user agent is a software environment which decodes and presents a declarative application to an end-user. An example of a user agent is a multimedia document browser. An alternate name for a user agent is *presentation engine*.

1.3 Application

The architecture and facilities of this specification are intended to apply to terrestrial (over-the-air) broadcast systems and receivers. In addition, the same architecture and facilities may be applied to other transport systems (such as cable or satellite).

1.4 Organization

This specification is organized as follows:

- Section 1 Describes purpose, application and organization of this specification
- Section 2 Enumerates normative and informative references
- Section 3 Defines acronyms, terminology, and conventions

[1] The user's attention is called to the possibility that compliance with this standard may require use of an invention covered by patent rights. By publication of this standard, no position is taken with respect to the validity of this claim or of any patent rights in connection therewith. One or more patent holders have, however, filed a statement regarding the terms on which such patent holder(s) may be willing to grant a license under these rights to individuals or entities desiring to obtain such a license. Details may be obtained from the ATSC Secretary and the patent holder. (This note was editorially updated on 11 June 2008.)

- Section 4 Specifies declarative application and environment behavior
- Section 5 Specifies declarative application and environment facilities
- Annex A Specifies document type definitions for markup facilities
- Annex B Specifies default stylesheet for style facilities
- Annex C Specifies document object model for scripting facilities
- Annex D Specifies binding of document object model to ECMAScript
- Annex E Describes unsupported HTML features
- Annex F Describes transcoding from HTML to markup facilities
- Annex G Describes accessibility considerations
- Annex H Depicts examples of use of *object* element
- Changes Cumulative changes to specification

Unless explicitly indicated otherwise, all annexes shall be interpreted as normative parts of this specification.

This specification makes use of certain notational devices to provide valuable informative and explanatory information in the context of normative and, occasionally, informative sections. These devices take the form of paragraphs labeled as *Example* or *Note*. In each of these cases, the material is to be considered informative in nature.

2. REFERENCES

This section defines the normative and informative references employed by this specification. With the exception of Section 2.1, this section and its subsections are informative; in contrast, Section 2.1 is normative.

2.1 Normative References

The following documents contain provisions which, through reference in this specification, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All referenced documents are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent edition of the referenced document.

When a conflict exists between this specification and a referenced document, this specification takes precedence.

> *Note:* This specification uses a reference notation based on acronyms or convenient labels for identifying a reference (as opposed to using numbers).

[CSS2]

Cascading Style Sheets, Level 2, Recommendation, W3C

[DASE]

DASE-1 Part 1: Introduction, Architecture, and Common Facilities, A/100-1, ATSC

[DOM2]

Document Object Model (DOM) Level 2 Core, Recommendation, W3C

[DOM2-EVENTS]

Document Object Model (DOM) Level 2 Events, Recommendation, W3C

[DOM2-HTML]

Document Object Model (DOM) Level 2 HTML, Recommendation, W3C

[DOM2-STYLE]

Document Object Model (DOM) Level 2 Style, Recommendation, W3C

[DOM2-VIEWS]

Document Object Model (DOM) Level 2 Views, Recommendation, W3C

[ECMASCRIPT]

ECMAScript Language Specification, 3rd Ed., ECMA-262, ECMA

[HTML]

HTML 4.01 Specification, Recommendation, W3C

[HTTP]

Hypertext Transfer Protocol – HTTP/1.1, RFC2616, IETF

[HTTP-STATE]

HTTP State Management Mechanism, RFC2965, IETF

[LANG-TAGS]

Tags for the Identification of Languages, RFC3066, IETF

[UNICODE]

Unicode Character Encoding Standard, Version 3.2, Unicode Consortium

[URI]

Uniform Resource Identifiers: Generic Syntax, RFC2396, IETF

[XHTMLBASIC]

XHTML Basic, Recommendation, 19 Dec 2000, W3C

[XHTMLMIME]

The 'application/xhtml+xml' Media Type, RFC3236, IETF

[XHTMLMOD]

Modularization of XHTML, Recommendation, 10 Apr 2001, W3C

[XML]

Extensible Markup Language (XML) 1.0, Recommendation, W3C

[XMLBASE]

XML Base, Recommendation, 27 Jun 2001, W3C

[XMLNAMES]

Namespaces in XML, Recommendation, W3C

[XMLSTYLE]

Associating Style Sheets with XML Documents, Recommendation, W3C

2.2 *Informative References*

[CSS1]

Cascading Style Sheets, Level 1, Recommendation, W3C

[CSS2-ERR]

Errata in CSS2 REC-CSS2-19980512, W3C

[DDE-1]

Declarative Data Essence, Content Level 1, SMPTE 363M, SMPTE

[DOM0]

Declarative Data Essence – Document Object Model Level 0 (DOM-0) and Related Object Environment, SMPTE 366M, SMPTE

[DOM2-ERR]

Errata of the Document Object Model Level 2 Specifications, W3C

[IDL]

The Common Object Request Broker: Architecture and Specification, OMG

[SGML]

Standard Generalized Markup Language, ISO 8879, ISO

[TIRESIAS]

The Tiresias Typeface Family, http://www.tiresias.org/fonts/home.htm, Royal National Institute for the Blind (RNIB)

[WAI-ATAG]

Authoring Tool Accessibility Guidelines 1.0, Recommendation, W3C

[WAI-UAAG]

User Agent Accessibility Guidelines 1.0, Recommendation, W3C

[WAI-WCAG]

Web Content Accessibility Guidelines 1.0, Recommendation, W3C

2.3 Reference Acquisition

ATSC Standards

Advanced Television Systems Committee (ATSC), 1750 K Street N.W., Suite 1200 Washington, DC 20006 USA; Phone: +1 202 828 3130; Fax: +1 202 828 3131; http://www.atsc.org/.

ECMA Standards

ECMA, 114, rue du Rhône, CH-1204 Geneva, Switzerland; Phone: +41 22 849 60 00; Fax: +41 22 849 60 01; http://www.ecma.ch/.

IETF Standards

Internet Engineering Task Force (IETF), c/o Corporation for National Research Initiatives, 1895 Preston White Drive, Suite 100, Reston, VA 20191-5434, USA; Phone: +1 703 620 8990; Fax: +1 703 758 5913; http://www.ietf.org/.

ISO Standards

International Organization for Standardization (ISO), 1, rue de Varembé, Case postale 56, CH-1211 Geneva 20, Switzerland; Phone: +41 22 749 01 11; Fax: +41 22 733 34 30; http://www.iso.ch/.

OMG Standards

Object Management Group (OMG), 250 First Avenue, Suite 201, Needham, MA 02494, USA; Phone: +1 781 444 0404; Fax: +1 781 444 0320; http://www.omg.org/.

RNIB Standards

Royal National Institute for the Blind, 224 Great Portland Street, London W1N 6AA, United Kingdom; Phone: +44 20 7388 1266; Fax: +44 20 7388 2034; http://www.rnib.org.uk/.

SMPTE Standards

Society of Motion Picture and Television Engineers, 595 W. Hartsdale Avenue, White Plains, NY 10607-1824, USA; Phone: +1 914 761 1100; Fax: +1 914 761 3115; http://www.smpte.org/.

Unicode Standards

The Unicode Consortium, P.O. Box 391476, Mountain View, CA 94039-1476, USA; Phone: +1 650 693 3921; Fax: +1 650 693 3010; http://www.unicode.org/.

W3C Standards

World Wide Web Consortium (W3C), Massachusetts Institute of Technology, Laboratory for Computer Science, 200 Technology Square, Cambridge, MA 02139, USA; Phone: +1 617 253 2613; Fax: +1 617 258 5999; http://www.w3.org/.

3. DEFINITIONS

This section defines conformance keywords, acronyms and abbreviations, and terms as employed by this specification.

All acronyms, abbreviations, and terms defined by [DASE] apply to this specification. Only those acronyms, abbreviations, and terms specific to this document and not common to DASE in its entirety are defined herein.

3.1 Conformance Keywords

As used in this document, the conformance keyword *shall* denotes a mandatory provision of the standard. The keyword *should* denotes a provision that is recommended but not mandatory. The keyword *may* denotes a feature whose presence does not preclude compliance, that may or may not be present at the option of the content author or the user agent implementer.

As used in this document, the auxiliary verb *will* denotes simple futurity and is not considered a conformance keyword. It may, however, by used in conjunction with statements containing conformance keywords in order to elaborate the meaning of those statements.

As used in this document, the term *deprecate* is to be given the force of *should not*, i.e., it serves as an explicit dis-recommendation.

3.2 Acronyms and Abbreviations

CSS Cascading Style Sheet
DTD Document Type Definition
HTML Hypertext Markup Language
IDL Interface Definition Language
W3C World Wide Web Consortium

3.3 Terms

monomedia content type: a type of content which does not support the general embedding of other content types within its serialized or presented forms; for example, a PNG image is an instance of a monomedia content type.

multimedia content type: a type of content which supports limited or general embedding of other content types within its serialized or presented forms; for example, an XDML document is an instance of a multimedia content type.

user agent: an embodiment of a declarative application environment.

4. BEHAVIOR

This section describes certain normative behavior for DASE Applications and Systems which employ the declarative facilities defined by this specification.

4.1 Application Processing

A declarative application environment performs three types of processing in order to manifest the behavioral objectives of the application's author:

- decoding
- presentation
- interaction

During an application's lifecycle, that is, its period of processing in the application environment, the application progresses through various states. These states are described formally in [DASE], Section 5.1.3, *Application Lifecycle*, and further elaborated as pertains to declarative application processing in the following subsections.

4.2 Application Decoding

A declarative application environment begins decoding a declarative application's initial entity resource during application activation processing.

During the decoding of the application initial entity resource, other resources may be referenced, and, depending upon the context of the reference, the application environment may fetch those resources from its resource cache and begin decoding them as well.

In general, each resource content type is associated with a distinct decoder which understands the syntax and semantics of that content type. Some content types support the content or presentation embedding of other resources of different content types. These are referred to as *multimedia content types*. In contrast, content types which do not support such embedding are referred to as *monomedia content types*.

A content type and a decoder may be distinctly characterized as to whether they permit progressive decoding or not. A content type that permits a decoder to operate on deltas of successively longer prefixes of a resource's content is referred to as being *progressively decodable*.

Most content types permitted by this specification are progressively decodable. Irrespective of whether a content type is progressively decodable or whether a resource makes use of progressive coding, a decoder may or may not choose to support progressive decoding; that is, a decoder may wait until all content is available prior to decoding.

> *Note:* This specification does not define the mechanism by means of which a user agent may obtain a partial sequence of an application resource's content.

The internal representation of the decoded form of a resource content type is not defined by this specification.

4.3 Application Presentation

After partial or complete decoding of an application's initial entity resource, the application environment initiates the presentation of decoded content. Unlike procedural applications, declarative applications have an implied contract to present decoded content. This presentation is effected by the intrinsic semantics of the declarative application environment.

The semantics of content presentation depend upon the content type. In general, the presentation semantics of declarative applications may be mapped to the following two-dimensional graphics primitives:

- filling a rectangular or circular area
- stroking the outline of a rectangular or circular area
- compositing a raster image
- compositing a rasterized glyph sequence (a special case of the previous primitive)

In addition to the above, the following higher-level presentation semantics are typically involved in content presentation:

- viewport clipping and scrolling
- user interface controls or widgets (e.g., buttons, checkboxes, text edit fields, etc.)
- color and palette management

Certain content types entail the use of a relatively complex content formatting model which operates prior to or during actual presentation. In particular, markup content in conjunction with stylesheet content requires the use of a box or area formatting model in order to translate decoded content into the graphics primitives that present the decoded content.

Just as each resource content type is associated with a distinct decoder, each resource content type is typically associated with a distinct presentation component. This need not always be the case, however, since the decoded content of multiple content types may by represented by a unified data type capable of being presented by a single presentation component. For example, the various graphical image content types may be decoded into a single image format which can be presented by a single image presentation component.

> *Note:* The use of one or more presentation components for a collection of similar content types is an optimization which may be performed at the discretion of the application environment implementer.
>
> *Note:* Some resource content types are not directly presented as content but serve other roles in contributing to application behavior, e.g., stylesheet, script, and font content types.

A content type and its presentation component may be distinctly characterized as to whether they permit progressive presentation or not. A content type that permits a presentation component to operate on deltas of successively longer sequences of decoded content is referred to as being *progressively presentable*.

Most content types permitted by this specification are progressively presentable. Irrespective of whether a content type is progressively presentable or whether a resource makes use of progressive coding, a presentation component may or may not choose to support progressive presentation.

4.3.1 Display Configuration

A declarative application environment shall employ the display model prescribed by [DASE], Section 5.2.5, *Display Model*.

For presenting a declarative application, an application environment should provide a full-screen graphics plane which is aligned to the video plane; however, a more general configuration may be used as depicted by Figure 1 Display Reference Model in Section 5.3.1.2.4.1 below.

4.4 Application Interaction

Once an application is activated after the successful decoding of the application initial entity, the application environment may enable user interaction with the presented content. At this point, the end-user is able to navigate among content, e.g., by scrolling, by invoking hyperlinks, etc.

During periods where interaction is permitted with presented content, a focus hierarchy is active. This hierarchy is established by either the nesting of logical content or the presented form of this content. For example, a relatively complex focus hierarchy might be:

- on screen display containing active application region
- frame

- active object
- text field

Note: This specification does not require support for multiple applications, so inter-application focus movement is not defined; however, inter-frame, object, and field focus movement is possible within a single application.

Note: Some resource content types do not have interaction semantics as content but serve other roles in contributing to application behavior, e.g., stylesheet, script, and font content types.

4.5 Trigger Processing

A declarative application environment shall process trigger content as defined by [DASE], Section 6.9. Trigger content may declare one or more events of a specific type which require processing by a declarative application environment. A declarative application environment shall process all events with a *type* attribute of the following value:

- `script`

4.5.1 Event Processing

An event whose type is defined as requiring processing by a declarative application environment is processed by instantiating an `Event` object based on the information provided by the *event* element then dispatching the `Event` object to a target document object instance hierarchy according to the event propagation semantics specified by [DOM2-EVENTS], Section 1.2.

4.5.1.1 *bubbles* attribute

This attribute shall be processed by a declarative application environment.

Note: See [DASE], Section 6.9.1.6.1.1, for more information on this attribute.

4.5.1.2 *cancelable* attribute

This attribute shall be processed by a declarative application environment.

Note: See [DASE], Section 6.9.1.6.1.2, for more information on this attribute.

4.5.1.3 *target* attribute

This attribute shall reference a resource of content type `application/xhtml+xml`. If the URI specifies a fragment identifier, then the fragment identifier shall identify a unique target element within the resource referenced by the URI; otherwise, the target element shall be the document element of the document object instance hierarchy produced as a result of decoding the resource referenced by the URI.

Note: See [DASE], Section 6.9.1.6.1.3, for more information on this attribute.

4.5.2 Script Event Processing

An event whose type attribute has a value of `"script"` shall be processed as a *script* event.

The *script* event type provides support for certain types of legacy content. The parameter set of an event of this type includes a required *code* parameter. The value of the *code* parameter shall adhere to the syntax of the *StatementList* non-terminal of [ECMASCRIPT], Section 12.1; in particular, it shall be syntactically valid for use as the content of the *Block* non-terminal.

When a script event is processed, the following steps shall occur:

(1) an `Event` object is instantiated and initialized with `Event::initEvent`, where *eventTypeArg* is `"org.atsc.script"`, *canBubbleArg* is the value of the *bubbles* attribute, and *cancelableArg* is the value of the *cancelable* attribute;
(2) a `Function` object of one argument and a body equal to the value of the *code* parameter specified by a *param* child element of this *event* element is instantiated and registered as a non-capturing event listener using the `EventTarget::addEventListener` method on the target element object (which shall implement both the `EventTarget` interface and the `HTMLTriggerObjectElementExt` interfaces);
(3) the `Event` object created in (1) is dispatched using the `EventTarget::dispatchEvent` on the document element object (which shall implement the `EventTarget` interface);
(4) the event listener registered in (2) is removed using `EventTarget::-removeEventListener`;

See [DOM2-EVENTS], Section 1, for information on `Event` and `EventTarget` interfaces. See Section 5.3.1.2.3.7 for information on the `HTMLTriggerObjectElementExt` interface.

4.6 Relative Identifier Resolution

A declarative application employs resource identifiers in order to reference various types of application resources in a variety of contexts. These identifiers take a form as described by [DASE], Section 5.1.2.3.1, *Resource Identifiers*. A reference to a resource may take an absolute or a relative form as described by [DASE], Section 5.1.2.3.2, *Resource References*. When a resource reference takes a relative form, the following interpretive rules shall be applied.

Given a relative identifier, the base identifier to be used to absolutize a relative identifier shall be determined by the following rule. If this rule does not permit resolution of a relative identifier, then an attempt shall be made to use rule two and following as specified by [DASE], Section 5.1.2.3.2.1, *Relative Resource Identifiers*.

(1) for a relative identifier used within an entity of the markup content type, use the rules prescribed by [XMLBASE], Section 4, *Resolving Relative URIs*.

4.7 Ecmascript Identifier Scheme

A URI which employs the *ecmascript* scheme, referred to below as an *ecmascript* URI, may be employed by a legacy declarative application and shall be supported by a declarative application environment.

Use of an *ecmascript* URI shall be restricted to the following contexts:

- *a* (anchor) element, *href* attribute
- *area* element, *href* attribute
- *frame* element, *src* attribute

Use of a *ecmascript* URI in any context other than the above either shall be ignored, producing no side effect, or shall cause an implementation defined notification to the end-user.

A *ecmascript* URI shall adhere to the following syntax:

```
ecmascript_URI : "ecmascript:" statementList
```

The construct *statementList* shall be non-empty and shall adhere to the syntax prescribed by [ECMASCRIPT], Section 12.1. Furthermore, the scheme-specific-part of a *ecmascript* URI (i.e., the *statementList*) shall satisfy the syntax of the `opaque_part` non-terminal of the generic URI syntax prescribed by [URI].

The resolution of an *ecmascript* URI shall adhere to the following procedure or a logical equivalent thereof:

(1) Evaluate *statementList* in an execution context identical to that used to evaluate *Global Code*, as further described by [ECMASCRIPT], Section 10.2.1;

(2) If the resulting value produced by (1) is not the *undefined* value, then convert the resulting value to the *String Type*; if the resulting string value is non-empty and a valid document instance or fragment of `text/html` content according to either the traditional or frameset document types specified by [HTML], then transcode this content to `application/xhtml+xml` and use the resulting value as the content of the resource accessed by this URI;

(3) If the resulting value produced by (1) is the *undefined* value, take no further action.

Note: If the resulting string is non-empty and invalid content of type `text/html`, the result of accessing the resource associated with the URI is unspecified.

Note: The form of an *ecmascript* URI which does not specify a statement list after the scheme component is not supported by this specification; in certain legacy user agents, when accessed through an *a* (anchor) element, this would cause an ecmascript interpreter user interface to be presented to the end-user to permit the direct entry of ecmascript statements. This feature is not supported by this specification.

Note: If any invocation of `document.write()` within *statementList* would modify the document which evaluates the *ecmascript* URI, then the result of accessing the resource associated with the URI is unspecified.

5. FACILITIES

This specification defines a number of facilities, each of which defines a category of content types by enumerating a set of one or more specific content types. The following categories are defined:

- markup content
- stylesheet content
- script content

A declarative application environment shall implement all the facilities specified in this section, and declarative application entities may use these facilities.

5.1 Markup Content

This facility consists of a markup language content type. Every declarative application shall contain one or more application entities which take the form of this markup language. In particular, the initial entity of a declarative application shall take the form of this markup language.

If the initial entity of a declarative application is a content type other than that of the markup language defined by this facility and this application is invoked, then the user agent shall abort the application.

5.1.1 application/xhtml+xml

Markup content shall adhere to the *Extensible Markup Language, Version 1.0* [XML] and shall be identified as content type `application/xhtml+xml` in accordance with [XHTMLMIME].

Furthermore, markup content shall adhere to one of the following document types, labeled here according to its formal public identifier, and defined in ANNEX A Document Type Definitions and [XHTMLBASIC] according to procedures set forth in *Modularization of XHTML* [XHTMLMOD]:

- `"-//ATSC//DTD XHTML XDML 1.0//EN"`
- `"-//W3C//DTD XHTML Basic 1.0//EN"`

Note: This first of the above document types, defined below in Section A.1, is referred to as an XDML (Extensible DTV Markup Language) document type. The elements declared by this document type are collectively referred to as XDML elements. Application entities which employ this document type are referred to as XDML documents or XDML content.

Note: The XDML document type includes all [XHTMLMOD] modules except for the following: applet, basic forms, basic tables, edit, iframe, legacy, and server-side image map. See ANNEX E, Unsupported HTML Features, and ANNEX F, Transcoding HTML to XDML, for more information on which features of HTML content are not supported by XDML and which features are supported through transcoding.

A user agent which implements this facility shall satisfy the XHTML Family User Agent Conformance criteria specified in [XHTMLMOD], Section 3.5.

Note: The XHTML Family User Agent Conformance criteria items (4)-(6) refer to scenarios where an unknown element type, attribute, or attribute value may not be recognized by a user agent; here, the term *recognize* is to be interpreted as semantic recognition as expressed by presentation and processing semantics, and not as syntactic recognition. This optional semantic recognition is independent from the requirement that every document specify a document type and that it be both well-formed and valid.

5.1.1.1 Well Formedness

An application entity which employs this content type shall be well formed as prescribed by [XML], Section 2.1.

If an entity of a declarative application uses content type `application/xhtml+xml`, is not well formed, and the entity is processed, then the user agent shall abort the application.

5.1.1.2 Validity

An application entity which employs this content type shall be valid as prescribed by [XML], Section 2.8.

If an entity of this content type is not valid, then the user agent should report violations of validity constraints set forth in [XML]. If an entity of a declarative application uses content type `application/xhtml+xml`, is not valid, and the entity is processed, then the user agent should abort the application.

A user agent which processes this content type implements an XML processor. An XML processor may be characterized as validating or non-validating. In both cases, adherence to [XML] requires the XML processor to check and report all violations of well-formedness constraints. However, an XML processor may or may not check and report violations of validity constraints. If it does check and report violations of validity constraints, it is referred to as a validating XML processor. This specification recommends, but does not require a user agent to implement a validating XML processor.

> *Note:* See [XML], Section 5.1, *Validating and Non-Validating Processors*, for more information on validation processing.

The manner in which a user agent reports violations to XML validity constraints is not defined by this specification. A user agent may simply abort the application or it may provide detailed information about the violations. It is recommended that user agents provide some form of feedback to the end-user in the case that an application is aborted due to such violations, particularly in cases where the application was explicitly invoked or selected by the user.

5.1.1.3 XML Declaration

An application entity which employs this content type shall specify a valid XML declaration [XML:23].

> *Note*: The notation [XML:23] refers to [XML] grammar production 23.

If an entity of a declarative application uses content type `application/xhtml+xml`, does not specify a valid XML declaration, and the entity is processed, then the user agent should abort the application.

An application entity which employs this content type shall specify an XML declaration with an encoding declaration [XML:80] according to one of the following character encoding systems, and, furthermore, the entity's representation shall employ the specified encoding system as its actual character encoding system.

- `"UTF-8"`
- `"ISO-8859-1"`

If an entity of a declarative application uses content type `application/xhtml+xml`, does not specify one of the preceding encoding declarations in the XML declaration according to the actual employed character encoding system, and the entity is processed, then the user agent may abort the application.

An application entity which employs this content type shall not specify an XML declaration with a standalone document declaration [XML:32] with the value `"yes"`.

If an entity of a declarative application uses content type `application/xhtml+xml`, does specify a standalone document declaration with the value `"yes"`, and the entity is processed, then the user agent may abort the application.

Note: A standalone document is one which makes no use of external declarations, but rather, incorporates all declarations into the document's internal declaration subset. However, this specification does not permit the use of internal declarations subsets (see Section 5.1.1.4, Document Type Declaration), and, therefore, does not support standalone documents.

5.1.1.4 Document Type Declaration

An application entity which employs this content type shall specify a valid document type declaration [XML:28].

If an entity of a declarative application uses content type `application/xhtml+xml`, does not specify a valid document type declaration, and the entity is processed, then the user agent should abort the application.

An application entity which employs this content type shall specify a document type declaration with an external identifier [XML:75] containing one of the following public identifiers [XML:12]:

- `"-//ATSC//DTD XHTML XDML 1.0//EN"`
- `"-//W3C//DTD XHTML Basic 1.0//EN"`

If an entity of a declarative application uses content type `application/xhtml+xml`, does not specify one of the preceding public identifiers in an external identifier in the document type declaration, and the entity is processed, then the user agent may abort the application.

In order to provide for forward compatibility with future revisions of the DASE Standard, a declarative application environment should not abort and should process an application entity whose document type declaration specifies a public text description beginning with `"XHTML"`.

Note: See [XHTMLMOD], Section 3.5, for further information on processing XHTML Host Language Conformant Document Types.

An application entity which employs this content type shall not specify an internal declaration subset. An internal declaration subset is that part of [XML:28] consisting of the delimiters `'['`, `']'`, and all intervening whitespace, delimiters, and non-terminals.

If an entity of a declarative application uses content type `application/xhtml+xml`, does specify an internal declaration subset (whether empty or not), and the entity is processed, then the user agent may abort the application.

5.1.1.5 Attribute Semantics

With the exception of differences noted in the following subsections, the semantics of XHTML attributes derive from the semantics specified by [XHTMLMOD], which, in turn, derive from the semantics specified by [HTML].

5.1.1.5.1 Universal Resource Identifiers (URIs)

Attributes whose values are specified with the `%URI.datatype;` or `%URIs.datatype;` parameter entities as defined in [XHTMLMOD], Section F.2.3, *XHTML Datatypes*, shall adhere to the syntax specified by *Universal Resource Identifiers* [URI] for representing URIs; furthermore, a URI shall adhere to the syntax of one of the schemes admitted by [DASE], Section 5.1.2.3.1, *Resource Identifiers*, as further constrained below.

If a URI is processed for resource access and the URI's scheme type is not supported, then the user agent should take some well-defined, default action; for example, present an error indication to the end-user.

If an entity of a declarative application uses content type `application/xhtml+xml`, does specify a URI for which the scheme part of its absolutized form is not one of the admitted scheme

types, and the URI is processed for resource access, then the user agent shall not abort the application.

5.1.1.5.1.1 Fragment Identifiers

If a URI specifies a fragment identifier and the referenced resource takes the form of the content type `application/xhtml+xml`, then the following procedure shall be followed to determine the URI destination:

(1) if the value of the *id* attribute of an element matches the fragment identifier, then that element is the destination;
(2) otherwise, if the value of the *name* attribute of an element matches the fragment identifier, then the first element whose *name* attribute matches is the destination;
(3) otherwise, the document element is the destination.

5.1.1.5.2 *name* attribute

Except for those element types which require the presence of a *name* attribute, an XHTML document instance shall not use the *name* attribute unless the document instance is produced as a result of transcoding a legacy `text/html` document, in which case a *meta* element with a `Transcoded-From-Type` specifying `text/html` shall be present. See Section 5.1.1.6.7.3 for additional information on transcoding information.

If an entity of a declarative application uses content type `application/xhtml+xml`, does specify an element with a *name* attribute, and is not a legacy application, then the user agent may abort the application.

Note: See [DASE], Section 3.3, for the definition of *legacy application*.

If a *name* attribute is permitted according to the above constraints, the decoding semantics of any element type is restricted in regard to the form and uniqueness of the element's *name* attribute. The value of the *name* attribute, if specified, shall be restricted to start with a Latin letter `[A-Za-z]`, followed by zero or more Latin letters, digits `[0-9]`, hyphens `'-'`, underscores `'_'`, and, if no script content is used by the application, colons`':'` and full stops `'.'`.

A user agent should ignore the *name* attribute of an element which does not satisfy the above constraints. If an entity of a declarative application uses content type `application/xhtml+xml`, does specify an element with a *name* attribute whose value does not satisfy the above constraints, then the user agent shall not abort the application.

Where the default value of a *name* attribute is defined by the applicable document type declaration as #IMPLIED, the implied default value shall be either (1) the value of the *id* attribute, if specified, or (2) the empty string.

Except for those element types which require the presence of a *name* attribute, use of this attribute is deprecated; the *id* attribute should be used instead.

Note: Optional use of this attribute is provided to obtain interoperability with transcoded, legacy content.

5.1.1.5.3 *style* attribute

The decoding semantics of the inline *style* attribute are restricted in regard to the form and meaning of the attribute value. The form and meaning of the attribute value shall adhere to the syntax and semantics of the document's default style content type, possibly constrained further by the style content type itself. See Section 5.1.1.6.7.2, Default Style Content Type.

If the default style content type is `text/css`, then the form and meaning of the inline style attribute's value shall adhere to the constraints specified in Section 5.2.1.9.

5.1.1.5.4 *xml:base* attribute

The resolution of relative URIs shall employ the *xml:base* attribute as described above by Section 4.6, Relative Identifier Resolution, and as specified by [XMLBASE]. This attribute is permitted on any XDML element. Its declared value shall conform to `CDATA`. No default value is specified.

> *Note*: This attribute may or may not be supported by a given XHTML Host Language Conformant Document Type. If it is not supported, then the information present in a *base* element is used in relative identifier resolution.

5.1.1.5.5 *xml:lang* attribute

The value of the *xml:lang* attribute shall adhere to the syntax prescribed by [LANG-TAGS]. No default value is specified.

5.1.1.6 Element Semantics

With the exception of differences noted in the following subsections, the semantics of XHTML elements derive from the semantics specified by [XHTMLMOD], which, in turn, derive from the semantics specified by [HTML]. The semantics of XHTML elements may be characterized according to the following:

- decoding semantics
- presentation semantics
- interaction semantics

Decoding semantics govern the way an element and its content are interpreted by the application environment. Presentation semantics govern the way an element and its content are formatted and presented to the end-user. Interaction semantics govern the way the end-user interacts with the element and its content. Certain elements have null presentation semantics, meaning that the element does not specify any direct presentation properties; e.g., the *meta* element. Similarly, certain elements have null interaction semantics, meaning that the element does not specify any direct interaction properties; e.g., the *div* element.

> *Note*: An element that has non-null presentation semantics may be altered by stylesheet content in order to cause it to be interpreted as if it had null presentation semantics. For example, this may be accomplished by using the `display` style property with the value `none`.
>
> *Note*: An element that has null interaction semantics may be altered by script content in order to cause it to be interpreted as if it had non-null interaction semantics. For example, this may be accomplished by using an event handler that responds to anchor activation events.
>
> *Note*: Certain basic presentation and interaction semantics cannot be altered by an application; e.g., *table* presentation and *anchor* interaction semantics cannot be changed and cannot be associated with another element type.

5.1.1.6.1 *a* (anchor) element

The section specifies constraints on the semantics of the *a* (anchor) element.

If an *a* (anchor) element does not specify a *target* attribute, then the user agent shall behave as if a target attribute with the value `"_self"` were specified.

If an *a* (anchor) element does specify a *target* attribute with the value `"_blank"`, then a user agent which does not support multiple windows shall behave as if a *target* attribute with the value `"_self"` were specified.

> *Note*: A user agent is not required to support multiple top-level windows.
>
> *Note*: See [HTML], Section 16.3.2, *Target Semantics*, for more information on the use of the *target* attribute. Regarding the interpretation of these semantics, since this content

type does not support the *base* element, but instead uses the *xml:base* attribute, then precedence rules (2) and (3) should be interpreted as if no *base* element were present. Furthermore, since a user agent is not required to support multiple top-level windows, a *target* attribute which refers to an unknown frame *F* need not cause a new window and frame to be created. In this case, the default behavior should be to interpret the *target* as if the value `"_self"` were specified.

5.1.1.6.1.1 href attribute

The semantics of the *a* (anchor) element are restricted in regard to the content type of the anchor's destination, i.e., the entity referenced by the *href* attribute, as well as the context in which a particular content type may be activated. The content type of an anchor's destination shall be one of the following types, as permitted by this specification:

- `application/dase`
- `application/xhtml+xml`
- `video/mpeg`, `video/mpv`
- `audio/basic`, `audio/ac3`

Note: In certain, limited circumstances, an *a* (anchor) element may reference a destination whose content is generated dynamically. See Section 5.1.1.6.1.1.5 below for further information about this usage.

5.1.1.6.1.1.1 Referencing Application Metadata Content

If an *a* (anchor) element's destination content type is `application/dase`, then anchor activation shall cause (1) termination of the current application and (2) instantiation of a new application instance where the anchor's destination resource is the application root entity of the new application instance.

The process of replacing an application by activation of an anchor which references application metadata content is referred to as *application replacement by anchor activation*.

Note: In this context, referencing an entity of content type `application/dase` is equivalent to referencing a DASE application as a whole.

Note: A DASE application is able to cause itself to be replaced by a new instance of itself by causing or permitting the activation of an anchor which refers to its own root entity.

If an entity is characterized as content type `application/dase`, then that entity shall adhere to [DASE], Section 6.1.1.

5.1.1.6.1.1.2 Referencing Markup Content

If an *a* (anchor) element's destination content type is `application/xhtml+xml`, and the anchor's destination resource is an entity of the current declarative application, then anchor activation shall not cause termination of the current declarative application; rather, the referenced markup content shall be loaded and presented in the targeted frame.

5.1.1.6.1.1.3 Referencing Video Content

If an *a* (anchor) element's destination content type is `video/mpeg` or `video/mpv` and (1) the current application is not a legacy declarative application, (2) the entity containing the *a* (anchor) element is part of a legacy application but does not contain a `Transcoded-From-Profile` *meta* element with the value `"SMPTE DDE-1"`, or (3) the target of the anchor is not the top-level frame (window), then anchor activation shall not cause termination of the current declarative application; rather, the referenced video content shall be rendered by the environment's video capabilities into the targeted frame, if such capabilities are present and enabled.

Note: See Section 5.1.1.6.7.3 for further information on the `Transcoded-From-Profile` *meta* element.

If an *a* (anchor) element's destination content type is `video/mpeg` or `video/mpv`, the containing entity is part of a legacy declarative application and contains a `Transcoded-From-Profile` *meta* element with the value `"SMPTE DDE-1"`, and the target of the anchor is the top-

level frame (window), then anchor activation shall cause (1) the referenced video content to be rendered by the environment's video capabilities into a full-screen video plane and (2) termination of the current declarative application in a manner consistent with the procedure prescribed for the `Window::close` method.

Note: See Section 5.3.1.2.9.4.3 for further information on the `Window::close` method.

If an entity is characterized as content type `video/mpeg`, then that entity shall adhere to [DASE], Section 6.5.1. If an entity is characterized as content type `video/mpv`, then that entity shall adhere to [DASE], Section 6.5.2.

5.1.1.6.1.1.4 Referencing Audio Content

If an *a* (anchor) element's destination content type is `audio/basic` or `audio/ac3`, and the anchor's destination resource is an entity of the current declarative application, then anchor activation shall not cause termination of the current declarative application; rather, the referenced audio content shall be rendered by the environment's audio capabilities, if such capabilities are present and enabled.

Rendering of audio content should be effected by mixing the referenced audio content with the active audio of any video content being simultaneously rendered.

If an entity is characterized as content type `audio/basic`, then that entity shall adhere to [DASE], Section 6.4.1. If an entity is characterized as content type `audio/ac3`, then that entity shall adhere to [DASE], Section 6.6.1.

5.1.1.6.1.1.5 Referencing Dynamically Generated Content

If an *a* (anchor) element employs a *ecmascript* URI, then it shall adhere to the constraints specified above in Section 4.7. Further, if the result of evaluating the statement list of the *ecmascript* URI produces the value *undefined*, then, other than a side-effect produced by this evaluation itself, no other side-effect shall occur.

Example: Given an *a* (anchor) element in a legacy declarative application as follows:

```
<a href="ecmascript:alert('Hello World!')">Say Hello</a>
```

then activation of this anchor would cause the display of an alert dialog containing the text `"Hello World!"`, and, when the dialog is closed, the original content (i.e., the document containing the above element) should remain displayed since the `alert()` method does not return a value (i.e., it evaluates to *undefined*).

In contrast, given the following *a* (anchor) element in a legacy declarative application:

```
<a href="ecmascript:'Hello World!'">Say Hello</a>
```

then activation of this anchor would cause the replacement of the content of the current document window with the text `"Hello World!"`.

5.1.1.6.1.1.6 Referencing Other Content

In the case that the content type of an anchor's destination entity is not supported by a content decoder, then the user agent shall either (1) ignore the anchor activation and produce no side effect, or (2) present feedback to the end-user that anchor activation will/does produce no effect. If an entity of a declarative application uses content type `application/xhtml+xml`, does reference a destination entity of a content type not listed above in an anchor element, and the anchor is activated, then the user agent shall not abort the application.

If an entity of a declaration application uses content type `application/xhtml+xml`, and an anchor's destination resource is neither an entity of the current declarative application nor the root entity of another DASE application, then the user agent shall either (1) ignore the anchor activation and produce no side effect, or (2) present feedback to the end-user that anchor activation will/does produce no effect.

5.1.1.6.2 *base* element

If a *base* element appears in an XHTML document instance, then it shall be interpreted as if the value of its *href* attribute were specified as the value of an *xml:base* attribute on the root *html* element.

If both an *xml:base* attribute appears on the root *html* element and a *base* element appears in an XHTML document instance, then the *base* element shall be ignored, and, the value of the *xml:base* attribute shall serve as the document instance's default base identifier.

> *Note*: The *base* element is provided for facilitating interoperability with legacy content. Content authors should use the *xml:base* attribute rather than the *base* element when authoring new XDML content.

5.1.1.6.3 *form* element

This specification does not define a mechanism for obtaining privacy of personal information obtained by means of a *form* element.

5.1.1.6.4 *img* element

The semantics of the *img* element are restricted in regard to the content type of the referenced image entity, i.e., the entity referenced by the *src* attribute. The content type of an image entity shall be one of the types permitted by [DASE], Section 6.2, Graphics Content.

If an entity of a declarative application uses content type `application/xhtml+xml`, does specify an *img* element whose referenced resource is a content type other than those permitted by the above paragraph or the entity is found to be invalid, then the user agent should allocate a presentation area and render into this area any associated alternative text or, if and only if no alternative text is available, an icon indicating the presence of unrenderable content.

If an entity referenced by an *img* element takes the form of a content type other than one of the above types or the entity is found to be invalid, then the user agent shall not abort the application.

> *Note*: The *img* element is provided for facilitating interoperability with legacy content. Content authors should use the *object* element rather than the *img* element when authoring new XDML content.

5.1.1.6.5 *input* element

The decoding semantics of the *input* element are restricted to interpreting only those *input* elements with certain values of the *type* attribute. Only the following values of the type attribute are defined for use by this specification. An *input* element with any other value of the *type* attribute may be ignored by the user agent. An *input* element which does not specify a *type* attribute shall be interpreted as if a *text* attribute with a value of `"text"` were specified.

- `button`
- `checkbox`
- `hidden`
- `image`
- `password`
- `radio`
- `reset`
- `submit`
- `text`

Users may enter password information in clear text into an *input* element of type password. According to constraints specified in Section 5.3.1.2.3.6.1, script content is not to be given direct access to the clear text of user entered passwords.

5.1.1.6.6 *link* element

The processing semantics of the *link* element are restricted to interpreting only those *link* elements with certain values of the *rel* attribute. The value of the *rel* attribute shall be one of the values specified by [HTML], Section 6.12, *Link Types*, or a non-standard value which starts with the prefix `"x-"`.

The value of the *rel* attribute shall be treated as case-insensitive for the purpose of determining value equality.

If the value of the *rel* attribute of a *link* element is `"stylesheet"`, then the *href* attribute shall reference a style sheet resource in accordance with Section 5.2, *Stylesheet Content*. In addition to those media types specified by [HTML], Section 6.13, *Media Types*, the value of the *media* attribute may specify the `"atsc-tv"` media type defined below in Section 5.2.1.7.1.

> *Note*: The *link* element is provided for facilitating interoperability with legacy content. Content authors should use the *xml-stylesheet* processing instruction rather than the *link* element to specify an external stylesheet when authoring new XDML content.

5.1.1.6.7 *meta* element

The semantics of the *meta* element are restricted to interpreting only those *meta* elements with certain values of the *name* attribute. The value of the *name* attribute shall be one of the values specified below or a non-standard value which starts with the prefix `"x-"`.

- `Content-Script-Type`
- `Content-Style-Type`
- `Transcoded-From-Type`
- `Transcoded-From-Profile`

> *Note:* Non-standard values of the *name* attribute are intended for use by DASE Application and DASE System implementers who employ a private agreement. When a non-standard value is used, its user should take care to avoid collision with other non-standard values, such as by making use of an organization unique infix.

The value of the *name* attribute shall be treated as case-insensitive for the purpose of determining value equality.

5.1.1.6.7.1 Default Script Content Type

The default content type for certain script content embedded in an XHTML document may be changed by using a *meta* element with a *name* attribute with a value `Content-Script-Type`. In this case, the value of the *content* attribute shall be a valid MIME media type indicating the script content type.

This default content type applies to all cases where script content is specified in a context where the script's content type is not explicitly stated.

If more than one *meta* element specifies a document's default script content type, then the most recently specified element applies.

If no default script content type is specified, the content type shall default to `text/ecmascript`.

Although the present level of declarative application functionality (DASE-1) only permits the use of one type of script content, content authors should explicitly specify the default script content type for enhanced interoperability with future levels which may permit multiple script content types.

5.1.1.6.7.2 Default Style Content Type

The default content type for certain style content embedded in an XHTML document may be changed by using a *meta* element with a *name* attribute with a value `Content-Style-Type`. In

this case, the value of the *content* attribute shall be a valid MIME media type indicating the style content type.

This default content type applies to all cases where style content is specified in a context where the style's content type is not explicitly stated.

If more than one *meta* element specifies a document's default style content type, then the most recently specified element applies.

If no default style content type is specified, the content type shall default to `text/css`.

Although the present level of declarative application functionality (DASE-1) only permits the use of one type of style content, content authors should explicitly specify the default style content type for enhanced interoperability with future levels which may permit the use of multiple style content types.

5.1.1.6.7.3 Transcoding Information

If the current document was transcoded from another content type, this information should be recorded in the current document by using a *meta* element with a *name* attribute with a value `Transcoded-From-Type`. In this case, the value of the *content* attribute shall be a valid MIME media type indicating the source document's content type.

If the current document was transcoded from another content type profile, this information should be recorded in the current document by using a *meta* element with a *name* attribute with a value `Transcoded-From-Profile`. In this case, the value of the *content* attribute shall specify the source document's content type profile. The following profiles are defined for use with source documents transcoded from content type `text/html`:

- `"Web"`
- `"SMPTE DDE-1"`

When specifying a transcoding profile, the value of the *content* attribute shall be treated as case-insensitive for the purpose of determining value equality.

> *Note:* The `"Web"` profile designates generic Web profiles based on Versions 3.X or 4.X of [HTML]. The `"SMPTE DDE-1"` profile designates profiles based on [DDE-1].

Content authors may extend the above set of transcoding profiles for `text/html` content type sources or any other content type source provided that the profile value is prefixed with the string `"x-"`.

> *Note:* Non-standard values of a transcoding profile are intended for use by DASE Application and DASE System implementers who employ a private agreement. When a non-standard value is used, its user should take care to avoid collision with other non-standard values, such as by making use of an organization unique infix.

5.1.1.6.8 *object* element

The semantics of the *object* element are restricted in regard to the content type of the object's data, i.e., the entity referenced by the *data* attribute. The content type of an object's data entity shall be either (1) one of the types permitted by [DASE], Section 6.2, Graphics Content, Section 6.3, Non-Streaming Video Content, Section 6.4, Non-Streaming Audio Content, Section 6.5, Streaming Video Content, and Section 6.6, Streaming Audio Content, or (2) an application specific content type to be interpreted by an application defined object implementation.

If an entity of a declarative application uses content type `application/xhtml+xml`, does specify an *object* element whose referenced data resource is a content type other than those permitted by the above paragraph or the entity is found to be invalid, then the user agent should allocate a presentation area and render into this area any associated alternative text or, if and only if no alternative text is available, an icon indicating the presence of unrenderable content.

If an entity referenced by an *object* element takes the form of a content type other than one of the above types or the entity is found to be invalid, then the user agent shall not abort the application.

The decoding semantics of the *object* element are restricted in regard to the content type of the object's implementation, i.e., the entity referenced by the *classid* attribute. The content type of an object's implementation shall be `application/javatv-xlet`, as permitted by [DASE], Section 4.1.3. If no *classid* attribute is specified, then the data entity referenced by the *object* element shall not be an application specific content type.

> *Note:* When no *classid* attribute is specified, the *object* element's data entity must be a built-in content type according to the constraints specified above in the first paragraph of this section.
>
> *Note:* A *classid* attribute may be specified to reference an object's implementation even in the case of a content type explicitly defined for use with the DASE Standard. This may be used to provide an application-defined decoder for a built-in content type which would otherwise use a built-in decoder.

If an entity of a declarative application uses content type `application/xhtml+xml`, does specify an *object* element whose referenced implementation resource is a content type other than those permitted by the above paragraph, and the entity is processed, then the user agent shall not abort the application.

See ANNEX H for examples of the *object* element.

5.1.1.6.8.1 Active Content Object Element

The *object* element type may be used to reference one or more active content objects from `application/xhtml+xml` content. By using this feature, a declarative application may make indirect use of the procedural application environment. This use is considered indirect since no mechanism is provided for direct interaction between the declarative application and this procedural application content. Interaction is limited to (1) the transfer of input focus to the active content object (so that this content may obtain end-user input events); (2) the display of active content graphical output in an area within the displayed declarative application document view; and (3) the rendering of audio content under control of the active content object.

Active object content used for an *object* element's immediate implementation shall be restricted to content type `application/javatv-xlet` as permitted by [DASE], Section 4.1.3. This implementation may reference and thereby make use of other procedural content types admitted by [DASE], Section 4.1.3.

Each *object* element whose *classid* attribute references an entity of content type `application/javatv-xlet` shall cause the instantiation of a distinct Java TV™ Xlet with a distinct class loader. Arguments to the Xlet are provided by zero or more *param* element children of the *object* element, where the name attributes of the *param* elements are of the form `arg.`*`n`*, with *n* being a non-negative integer. If the largest value of *n* among all param elements is *N*, then the Xlet shall be instantiated with *N+1* arguments, and the *n*th argument shall be the value of the *param* element whose *name* attribute is `arg.`*`n`*. If some `arg.`*`n`* is not specified, then a default value of the empty string shall be used as the value.

The archive attribute, if specified, shall be a space separated list of URIs each of which specifies an archive content entity according to [DASE], Section 6.8, or a Java archive content entity as permitted by [DASE], Section 4.1.3.

The *classid* attribute of an active content object element shall be a URI which specifies the Java class resource that implements the `javax.tv.xlet.Xlet` interface.

The *codebase* attribute, if specified, shall be an absolute URI or a relative URI, which shall be absolutized according to Section 4.6, Relative Identifier Resolution. The resulting absolute URI shall be used to absolutize relative URIs used with the *archive*, *classid*, and *data* attributes. If not

specified, the *codebase* shall default to the base URI of the document entity in which the *object* element is specified.

The *codetype* attribute, if specified, shall be `application/javatv-xlet`.

In addition to Xlet arguments of the form `arg.n`, which are derived from *param* element children of the *object* element, the following additional parameters shall be provided to the Xlet as Xlet context properties:

- `org.atsc.xlet.obj.codebase`
- `org.atsc.xlet.obj.data`
- `org.atsc.xlet.obj.type`

These additional Xlet arguments shall be computed as follows:

(1) the `org.atsc.xlet.obj.codebase` parameter's value shall be the URI specified by the *codebase* attribute, or, if no *codebase* attribute is specified, the base URI of the document entity in which the *object* element is specified;

(2) the `org.atsc.xlet.obj.data` parameter's value shall be the absolutized form of the URI specified by the *data* attribute, or, if no *data* attribute is specified, the empty string;

(3) the `org.atsc.xlet.obj.type` parameter's value shall be the value specified by the *type* attribute, or, if no *type* attribute is specified, the empty string.

5.1.1.6.8.1.1 Active Content Document Instance Access

When an embedded Xlet is in its active state, it shall be given access to the containing document's object instance hierarchy.

5.1.1.6.8.2 Trigger Object Element

Notwithstanding the above constraints placed on use of the *object* element, a special use of the *object* element is permitted. This special use is referred to as a trigger object element and is used with certain legacy content. A trigger object element is an *object* element which (1) specifies an *id* attribute, (2) specifies a *type* attribute with value `application/dase-trigger`, and (3) does not specify a *classid* or *data* attribute. No more than one instance of a trigger object element may appear at any given instant within all `application/xhtml+xml` entities that compose a declarative application's current document instance hierarchy. See Section 5.3.1.2.3.7 for further requirements regarding the instantiation of an associated document object for use by script content.

5.1.1.6.9 *script* element

The semantics of the *script* element are restricted in regard to the values of the element's *type* attribute. The value of the *type* attribute is restricted to one of the types permitted by Section 5.3, Script Content.

A user agent should ignore a *script* element whose *type* attribute does not satisfy the above constraints. If an entity of a declarative application uses content type `application/xhtml+xml`, does specify a *script* element with a *type* attribute whose value does not satisfy the above constraints, then the user agent shall not abort the application.

The *script* element's content is parsed as `#PCDATA`. As a consequence, any entity reference open (*ero*), start tag open (*stago*), end tag open (*etago*), and markup declaration open (*mdo*) delimiters are recognized as markup rather than script content. The practical impact of this is that any ampersand character `'&'` or less-than character `'<'` should be escaped using an appropriate character reference, e.g., `&` and `<`, respectively. Alternatively, the script content may be enclosed in a `CDATA` marked section provided that the marked section close (*msc*) delimiter `"]]>"` does not appear in the enclosed content.

5.1.1.6.10 *style* element

The decoding semantics of the *style* element are restricted in regard to the values of the element's *type* attribute. The value of the *type* attribute is restricted to one of the types permitted by Section 5.2, Stylesheet Content.

A user agent should ignore a *style* element whose *type* attribute does not satisfy the above constraints. If an entity of a declarative application uses content type `application/xhtml+xml`, does specify a *style* element with a *type* attribute whose value does not satisfy the above constraints, then the user agent shall not abort the application.

The *style* element's content is parsed as `#PCDATA`. As a consequence, any entity reference open (*ero*), start tag open (*stago*), end tag open (*etago*), and markup declaration open (*mdo*) delimiters are recognized as markup rather than script content. The practical impact of this is that any ampersand character `'&'` or less-than character `'<'` should be escaped using an appropriate character reference, e.g., `&` and `<`, respectively. Alternatively, the style content may be enclosed in a `CDATA` marked section provided that the marked section close (*msc*) delimiter `"]]>"` does not appear in the enclosed content.

5.1.1.7 External Stylesheets

One or more external stylesheets may be associated with an XHTML document instance by using either the `xml-stylesheet` processing instruction in accordance with [XMLSTYLE] or the *link* element with a *rel* attribute equal to `"stylesheet"`.

A user agent which implements this facility shall support [XMLSTYLE] semantics.

If multiple external stylesheets are associated with an XHTML document instance, then the rules for their application precedence are determined by stylesheet specific semantics. The order of their appearance in an XHTML document instance may be considered significant when resolving style rules that would apply to the same content.

5.2 Stylesheet Content

This facility consists of a stylesheet language content type. A declarative application may contain one or more application entities which take the form of this stylesheet language. Furthermore, fragments of this stylesheet language may be embedded directly into markup language content as well as script language content.

A stylesheet is used to express how some other content is to be presented. In the context of this specification, that other content is the markup language content specified above (see Section 5.1).

5.2.1 text/css

Stylesheet content shall adhere to *Cascading Style Sheets, Level 2, 12 May 1998* [CSS2], as extended and restricted below. This content shall be identified as content type `text/css`.

> *Note:* See *Errata in REC-CSS2-19980512* [CSS2-ERR] for corrections and clarifications regarding [CSS2].

A user agent which implements this facility shall satisfy the CSS Level 2 User Agent Conformance requirements specified in [CSS2], Section 3.2, items (2)-(5).

> *Note:* Such a user agent *is not* strictly CSS Level 2 Conformant unless it supports one of the pre-defined media types specified by CSS Level 2. This specification does not require that any of the pre-defined media types be supported.

An application resource which represents an application entity of this content type should be accompanied by content type metadata which specifies a *charset* parameter indicating the character encoding which applies to the resource.

If the content type metadata for an application resource of type `text/css` contains a `charset` parameter, then (1) the value of that parameter shall be either `UTF-8` or `ISO-8859-1`, and (2) the character encoding of the resource shall employ the specified encoding system as its actual character encoding system.

5.2.1.1 Validity

An application entity which employs this content type shall be valid. A stylesheet content entity is valid if either (1) it is valid according to [CSS2], Section 3.1, or (2) after removing all references to non-standard style properties and property values, it is valid according to [CSS2], Section 3.1, and these references adhere to the syntax for the non-standard style properties and property values defined below.

If an entity of a declarative application uses content type `text/css`, is not valid, and the entity is processed, then the user agent shall not abort the application.

> *Note*: All stylesheet content entities which are valid according to [CSS2], Section 3.1, are valid according to this specification; however, the converse does not hold since this specification permits certain non-standard style properties in stylesheet content entities.

If an entity of this content type is not valid, then the user agent shall ignore certain invalid constructs according to [CSS2], Section 4.2, *Rules for Handling Parsing Errors*. A user agent may report violations of validity criteria set forth in [CSS2] and in this specification.

> *Note:* Certain non-standard extensions to [CSS2] are defined by this specification. In all cases, these extensions take the form of new identifiers which are permitted by [CSS2], Appendix D, *The Grammar of CSS2*. Furthermore, in each of these cases, the identifier is prefixed by the string `"atsc-"` in order to prevent collision with future W3C revisions of this content type.

5.2.1.2 Selectors

A user agent which implements this facility shall support the semantics of both (1) selector grouping as prescribed by [CSS2], Section 5.2.1, and (2) the following selector types:

Table 1 CSS Selector Support

universal	[CSS2], Section 5.3
type	[CSS2], Section 5.4
descendant	[CSS2], Section 5.5
child	[CSS2], Section 5.6
adjacent sibling	[CSS2], Section 5.7
attribute and attribute value	[CSS2], Section 5.8.1
class	[CSS2], Section 5.8.3
id	[CSS2], Section 5.9
:first-child pseudo-class	[CSS2], Section 5.11.1
link pseudo-classes	[CSS2], Section 5.11.2
:hover pseudo-class	[CSS2], Section 5.11.3
:active pseudo-class	[CSS2], Section 5.11.3
:focus pseudo-class	[CSS2], Section 5.11.3
:lang pseudo-class	[CSS2], Section 5.11.4
pseudo-elements	[CSS2], Section 5.12

> *Note:* The above selector types constitute a proper superset of the selector types specified by *CSS Level 1* [CSS1].

If a rule uses a valid [CSS2] selector type which is recognized by, but whose semantics are not supported by a user agent, then the user agent shall ignore the rule.

Note: Content authors should use simple selectors instead of complex selectors. This permits avoiding the inherent processing overhead required for complex selectors. Complex selectors include *descendant*, *attribute*, and *attribute value* selector types.

If a pointer device is not supported by a declarative application environment, then the semantics of the `:hover` pseudo-class selector need not be supported.

5.2.1.3 Character Set Rules

A user agent which implements this facility shall support the semantics of @*charset* rules [CSS2], Section 4.4.

A non-embedded stylesheet entity shall specify a @*charset* rule according to one of the following character encoding systems, and, furthermore, the stylesheet entity's representation shall employ the specified encoding system as its actual character encoding system:

- `"UTF-8"`
- `"ISO-8859-1"`

If an entity of this content type does not specify a @*charset* rule and no metadata about the character encoding system of the entity is otherwise available, then the user agent should process the entity's representation as if a @*charset* rule specifying `"UTF-8"` were present. If an entity of a declarative application uses content type `text/css`, does not specify a @*charset* rule with one of the preceding character encoding systems according to the actual employed character encoding system, and the entity is processed, then the user agent shall not abort the application.

If the content type metadata for the application resource which embodies an entity of type `text/css` specifies a `charset` parameter, than the character encoding system specified by the @*charset* rule shall be the same as the `charset` parameter's value.

5.2.1.4 Import Rules

A user agent which implements this facility shall support the semantics of @*import* rules [CSS2], Section 6.3.

If stylesheet content uses an @*import* rule, then the absolutized form of the URI argument to the rule shall adhere to constraints specified by Section 5.1.1.5.1 and this URI shall reference an entity of content type `text/css`.

If an entity of a declarative application uses content type `text/css`, either in a distinct resource or in stylesheet embedded in `application/xhtml+xml` content, does specify an @*import* rule with a URI for which (1) the scheme part of its absolutized form does not adhere to constraints specified by Section 5.1.1.5.1 or (2) the referenced entity is not content type `text/css`, and the import rule is processed for resource access, then the user agent shall not abort the application.

If an entity of this content type does specify an @import rule with a URI which cannot be resolved to a resource of this content type, then the user agent should ignore the @import rule.

5.2.1.5 Font Face Rules

A user agent which implements this facility shall support a subset of the semantics of @*font-face* rules [CSS2], Section 15.3, including (1) the font matching algorithm specified by [CSS2], Section 15.5, and (2) the following descriptors:

Table 2 CSS Font Face Rule Support

`'font-family'` descriptor	[CSS2], Section 15.3.2
`'font-style'` descriptor	[CSS2], Section 15.3.2
`'font-variant'` descriptor	[CSS2], Section 15.3.2

`'font-weight'` descriptor	[CSS2], Section 15.3.2
`'font-stretch'` descriptor	[CSS2], Section 15.3.2
`'font-size'` descriptor	[CSS2], Section 15.3.2
`'unicode-range'` descriptor	[CSS2], Section 15.3.3
`'src'` descriptor	[CSS2], Section 15.3.5

If a *@font-face* rule which uses an unsupported, but valid [CSS2] *@font-face* rule descriptor type is recognized by a user agent, then the user agent should ignore the descriptor. If an entity of a declarative application uses content type `text/css`, does specify a *@font-face* rule with a descriptor not listed above within an `atsc-tv` *@media* rule, and the entity is processed, then the user agent shall not abort the application.

5.2.1.5.1 'src' descriptor

If a `'src'` descriptor is specified in a *@font-face* rule, then the value of the descriptor shall consist of a single `url()` specification optionally followed by a `format()` specification. The `url()` specification shall specify a URI which adheres to constraints specified by Section 5.1.1.5.1 and which references a resource of a content type permitted by [DASE], Section 6.7, Font Content.

If a *@font-face* rule uses a valid `'src'` descriptor which does not satisfy the above constraints, then the user agent should either (1) use the first `url()` specification in the descriptor which does adhere to the above constraints or (2) ignore the descriptor. If an entity of a declarative application uses content type `text/css`, does specify a `'src'` descriptor which does not satisfy the above constraints, and the entity is processed, then the user agent shall not abort the application.

> *Note:* The `local()` specification of a font resource source provided by [CSS2], Section 15.3.5, is not supported by this specification.

A user agent should ignore any valid `local()` specifications present in a valid `'src'` descriptor.

If a `format()` specification is provided, it should be compatible with the content type of the resource referenced by the URI value of the `url()` specification.

If a `format()` specification is provided, then the user agent may interpret it as a hint of the referenced font resource's content type.

5.2.1.6 Page Rules

A user agent which implements this facility need not support the semantics of *@page* rules, described by [CSS2], Section 13.2; furthermore, an application entity shall not rely upon user agent support for *@page* rules.

5.2.1.7 Media Types

A CSS *media type* is a collection of CSS style properties associated with a particular media form or a manner in which content is presented on a media form. A number of media types are pre-defined by [CSS2], including `screen`, `print`, `tv`, `aural`, and others. This specification does not make use of any of these pre-defined media types. Rather, this specification defines a new media type, `atsc-tv`, to be used as the default media type for stylesheet content.

A user agent which implements this facility shall support the `atsc-tv` media type defined below.

> *Note:* For the purpose of interpreting [CSS2] statements regarding user agent conformance, an XHTML user agent is to be construed as an HTML user agent.

5.2.1.7.1 'atsc-tv' media type

The `atsc-tv` media type includes the following style properties:

Table 3 CSS Property Support

atsc-dynamic-refresh	border-top-color	margin
atsc-nav-down	border-top-style	margin-bottom
atsc-nav-index	border-top-width	margin-left
atsc-nav-left	border-width	margin-right
atsc-nav-right	bottom	margin-top
atsc-nav-up	caption-side	outline
background	clear	outline-color
background-attachment	clip	outline-style
background-color	color	outline-width
background-image	content	overflow
background-position	counter-increment	padding
background-repeat	counter-reset	padding-bottom
border	display	padding-left
border-bottom	float	padding-right
border-bottom-color	font	padding-top
border-bottom-style	font-family	position
border-bottom-width	font-size	right
border-color	font-style	text-align
border-left	font-variant	text-decoration
border-left-color	font-weight	text-indent
border-left-style	height	text-transform
border-left-width	left	top
border-right	letter-spacing	vertical-align
border-right-color	line-height	visibility
border-right-style	list-style	white-space
border-right-width	list-style-image	width
border-style	list-style-position	word-spacing
border-top	list-style-type	z-index

Content authors should avoid uses of the *overflow* property which result in scrolling behavior.

5.2.1.8 Style Properties

For a user agent to support a media type, it must implement the semantics of each style property included in that media type. The semantics of each style property are specified by [CSS2] as extended and constrained by this specification. The semantics of style properties with extensions or constraints are specified below.

5.2.1.8.1 Alternative Semantic Interpretations

Certain [CSS2] style properties are afforded alternative interpretations by either [CSS2] or this specification. In general, an alternative interpretation of a style property entails the user agent optionally ignoring or producing a different implied side effect.

Content authors should take into account the possibility that a style property will be ignored or interpreted differently than implied by its explicit semantics.

> *Note:* It is not necessary for a declarative application to employ a guard in order to use style properties which may have an alternative interpretation. This is due to the requirement that a user agent be able to parse valid style properties for which alternative interpretations may apply irrespective of whether the user agent employs an alternative interpretation or not.

5.2.1.8.1.1 'background' property

A user agent may ignore a declaration which specifies this property if the `<'background-image'>` sub-construct of this shorthand property would be ignored in accordance with the alternative interpretation of the *background-image* property (see Section 5.2.1.8.1.3).

5.2.1.8.1.2 'background-attachment' property

A user agent may interpret the value `fixed` as the value `scroll`. A user agent should correctly interpret the semantics of `fixed` for the *body* element.

5.2.1.8.1.3 'background-image' property

A user agent may ignore a declaration which specifies this property if the background is a content type other than the graphic or streaming video content types permitted by this specification.

If the background is a streaming video content type, then a user agent should attempt to scale and translate the video to the size and position of the affected element.

5.2.1.8.1.4 'background-repeat' property

A user agent may ignore a declaration which specifies this property if the background is a streaming video content type.

5.2.1.8.1.5 'border-top-style', 'border-left-style', 'border-bottom-style', 'border-right-style', and 'border-style' properties

A user agent may interpret the values `dotted`, `dashed`, `double`, `groove`, `ridge`, `inset` and `outset` as the value `solid`.

5.2.1.8.1.6 'content' property

A user agent may ignore a declaration which specifies this property if it has a value which takes the form `<uri>` or if it has one of the following values: `open-quote`, `close-quote`, `no-open-quote`, or `no-close-quote`.

5.2.1.8.1.7 'display' property

A user agent may ignore a declaration which specifies this property if it has one of the following values: `run-in`, `compact`, `marker`, `table`, `inline-table`, `table-row-group`, `table-header-group`, `table-footer-group`, `table-row`, `table-column-group`, `table-column`, `table-cell`, or `table-caption`.

If a user agent ignores a declaration which specifies this property, it shall apply the implicit display semantics for the selected element according to [HTML].

5.2.1.8.1.8 'font' property

A user agent may ignore a declaration which specifies this property if it has one of the following values: `caption`, `icon`, `menu`, `message-box`, `small-caption`, and `status-bar`.

5.2.1.8.1.9 'font-variant' property

A user agent may interpret the value `small-caps` as the value `normal` for characters that do not have prescribed case-conversion rules.

5.2.1.8.1.10 'letter-spacing' property

A user agent may interpret a value of the form `<length>` as the value `normal`.

5.2.1.8.1.11 'list-style-type' property

A user agent may interpret the values `armenian`, `cjk-ideographic`, `georgian`, `hebrew`, `hiragana`, `hiragana-iroha`, `katakana`, `katakana-iroha`, and `lower-greek` as the value `decimal`.

5.2.1.8.1.12 'overflow' property

A user agent may interpret the value `scroll` as the value `auto`.

Note: A user agent is not required to support visible scroll indicators or scroll functionality when clipping occurs on an overflowing box.

5.2.1.8.1.13 'text-align' property

A user agent may interpret the value `justify` as the value `left` or `right`, depending upon whether the element's default writing direction is left-to-right or right-to-left, respectively.

5.2.1.8.1.14 'text-decoration' property

A user agent may interpret the value `blink` as the value `none`.

5.2.1.8.1.15 'white-space' property

A user agent may ignore a declaration which specifies this property.

If a user agent ignores a declaration which specifies this property, it shall apply the implicit white-space semantics for the selected element according to [HTML].

5.2.1.8.1.16 'word-spacing' property

A user agent may interpret a value of the form `<length>` as the value `normal`.

5.2.1.8.2 Standard Property Values

This section specifies additional constraints on standard [CSS2] style property values.

5.2.1.8.2.1 'font-size' property, '<relative-size>' values

The *<relative-size>* values of the *font-size* property shall be interpreted as the following font-relative sizes according to a reference font of size 24 pixels on a reference display of 480 pixels in height:

Table 4 CSS Relative Font Size Mappings

xx-small	0.33em
x-small	0.50em
small	0.75em
medium	1.00em
large	1.33em
x-large	2.00em
xx-large	2.67em

5.2.1.8.3 Non-Standard Properties

This section specifies non-standard, ATSC specific style properties.

Note: In this context, *non-standard* means not specified by the W3C for use with CSS.

5.2.1.8.3.1 'atsc-dynamic-refresh' property

The `atsc-dynamic-refresh` property may be used to specify that an XHTML document, image, or an embedded object in an XHTML document (i.e., an instance of an *object* element) is to be dynamically refreshed (redisplayed) when the underlying application resource is updated.

The following constraints apply to this property:

Values: auto | none | inherit
Initial: none
Applies to: *html*, *img*, *object*
Inherited: no

Percentages: n/a
Media: atsc-tv

If used with any other element type, this property shall be ignored.

If the value is `auto` and the underlying application resource is updated, then the document, image, or object shall be dynamically updated by re-decoding and re-presenting the contents of the updated resource. Otherwise, if the value is `none`, a change to the underlying application resource shall have no affect on previously decoded and presented document, image, or object content through means of this property.

> *Note*: In the case the value of this property is `none`, a refresh may still be effected by programmatic means by registering an `org.atsc.resourcechanged` event listener on an element of interest and then using the `DocumentViewExt::refresh` method to cause a refresh to occur. See Sections 5.3.1.2.8.3.1 and 5.3.1.2.4.1.10 for further information.

Only changes to the application resource referenced by the *data* attribute of an object element shall be construed as a dynamic update. Changes to an application resource referenced by the *classid* attribute of an object shall not cause a dynamic update.

The dynamic update of an XHTML document instance shall cause all script state associated with its prior document instance to be discarded.

> *Note*: The ability for an XHTML document instance, image, or object data instance to be updated is dependent upon the application delivery system used to deliver an application's resources. The property specified here is intended to support those application resource delivery mechanisms that support automatic updating (versioning) of resources, e.g., a broadcast data or object carousel.

5.2.1.8.3.2 'atsc-nav-index' property

The `atsc-nav-index` property may be used to specify that an element is the target of a directional navigation property (see `atsc-nav-left`, `atsc-nav-right`, `atsc-nav-up`, `atsc-nav-down` properties).

The following constraints apply to this property:

Values: auto | <number>
Initial: auto
Applies to: all focusable elements
Inherited: no
Percentages: n/a
Media: atsc-tv

When the value is `auto`, directional navigation behavior is not defined by this specification, but is implementation dependent.

When a *<number>* value is used to express an integer index value, no more than one element in a document instance shall be associated with the index.

5.2.1.8.3.3 'atsc-nav-{left,right,up,down}' properties

The `atsc-nav-{left,right,up,down}` properties may be used to specify an element to which focus should be moved when responding to focus movement events.

The following constraints apply to this property:

Values: auto | <number> | <string> | inherit
Initial: auto
Applies to: all focusable elements
Inherited: no
Percentages: n/a
Media: atsc-tv

When the value is `auto`, directional navigation behavior is not defined by this specification, but is implementation dependent.

When a *<number>* value form is used, it shall denote an integer index value that corresponds to the index value associated with an element by the `atsc-nav-index` property.

When a *<string>* value form is used, it shall adhere to the following syntax:

```
navigation-target:    [ frame-ref ] '#' element-ref
frame-ref:            idref
element-ref:          idref | '$' integer
```

The value of *frame-ref*, if specified, shall be a valid identifier reference associated with some frame in the current document instance frame hierarchy. The value of *element-ref* shall be either a valid identifier reference to some element or an integer reference to an index value associated with an element by an `atsc-nav-index` property. The scope of element reference shall be limited by the frame reference, if one is specified, or the current frame, if none is specified.

The syntax of *idref* shall adhere to the set of legal values of the XML *id* attribute.

> *Note:* Care should be taken by content authors to ensure that uses of these properties do not lead to unexpected navigation behavior. In particular, it is possible that, through variations in content layout, an element which might appear on the left or right in one layout context may appear on the right or left in another layout context; e.g., due to differences in default fonts, font sizes, text layout algorithms, etc.

5.2.1.8.4 Non-Standard Property Values

This section specifies non-standard, ATSC style property values.

> *Note:* In this context, *non-standard* means not specified by the W3C for use with CSS.

5.2.1.8.4.1 'atsc-rgba' function

The `atsc-rgba()` function may be used to specify an RGB color with alpha as a property value. The function takes four arguments, in the following order, indicating the red, green, blue, and alpha components of the color. Each argument shall be either an integer in the range 0 through 255 (inclusive) or a percentage. In this context, alpha is to be interpreted as opacity; that is, values of 255 or 100% shall be interpreted as opaque.

The `atsc-rgba()` function may appear anywhere that the `rgb()` function is permitted by [CSS2], Section 4.3.6.

5.2.1.9 Inline Stylesheets

When an inline stylesheet is specified in `application/xhtml+xml` content by means of the *style* attribute and the default style content type is `text/css`, then the normalized value of the *style* attribute shall adhere to the syntax of the following `inline-stylesheet` non-terminal:

```
inline-stylesheet
  : S* [ declarations | inline-rulesets ]?

declarations
  : declaration [ ';' S* declaration ]*

inline-rulesets
  : inline-ruleset*

inline-ruleset
  : [ pseudo S* [ ',' S* pseudo S* ]* ]? '{' S* declarations '}' S*
```

See [CSS2], Appendix D, *The Grammar of CSS2*, for the definitions of `declaration` and `pseudo` non-terminals.

The semantics of the above inline style specification are as follows: (1) declarations not associated with a pseudo selector apply to the element on which the inline style attribute occurs and have a cascading order equivalent to that of a rule-set specified at the end of the author's style sheet with a specificity of `{a=1,b=0,c=0}`; (2) declarations associated with a pseudo-class and not a pseudo-element selector apply to all document elements for which the pseudo-class applies and have a cascading order equivalent to that of a rule-set specified at the end of the author's style sheet with a specificity of `{a=1,b=1,c=0}`; (3) declarations associated with a pseudo-element selector and not a pseudo-class selector apply to the element on which this inline style attribute occurs and have a cascading order equivalent to that of a rule-set specified at the end of the author's style sheet with a specificity of `{a=1,b=0,c=1}`; and (4) declarations associated with both a pseudo-class and a pseudo-element selector apply to all document elements for which the pseudo-class applies and have a cascading order equivalent to that of a rule-set specified at the end of the author's style sheet with a specificity of `{a=1,b=1,c=1}`. See [CSS2], Section 6.4, *The Cascade*, for information on cascade specificity determination.

5.3 Script Content

This facility consists of a script language content type. A declarative application may contain one or more application entities which take the form of this script language. Furthermore, fragments of this script language may be embedded directly into markup language content.

5.3.1 text/ecmascript

Script content shall adhere to *ECMAScript, 3rd Edition* [ECMASCRIPT], as extended and restricted below, and shall be identified as content type `text/ecmascript`.

A user agent which implements this facility shall satisfy the ECMAScript Conformance requirements specified in [ECMASCRIPT], Section 2.

An application entity or fragment which employs this content type shall be valid. A script content entity or fragment is valid if it (1) adheres to the grammar specified by [ECMASCRIPT] and (2) makes reference only to application-defined host objects and to native objects and host objects permitted by this specification.

If an entity of a declarative application uses content type `text/ecmascript`, is not valid, and the entity or fragment is processed, then the user agent may abort the application.

An application resource which represents an application entity of this content type should be accompanied by content type metadata which specifies a *charset* parameter indicating the character encoding which applies to the resource.

If the content type metadata for an application resource of type `text/ecmascript` contains a `charset` parameter, then (1) the value of that parameter shall be either `UTF-8` or `ISO-8859-1`, and (2) the character encoding of the resource shall employ the specified encoding system as its actual character encoding system.

5.3.1.1 Native Objects

An ECMAScript *native object* is an object whose form and semantics are defined by [ECMASCRIPT].

5.3.1.1.1 Non-Instantiable Built-In Objects

A script content entity or fragment may reference the following non-instantiable, built-in, native objects as defined by [ECMASCRIPT].

Table 5 ECMAScript Non-Instantiable Native Object Support

Global	Math	

5.3.1.1.1.1 Global Object

The `Global` object shall implement the `Window` interface specified in Section 5.3.1.2.9.4.

5.3.1.1.2 Instantiable Built-In Objects

A script content entity or fragment may reference and instantiate the following instantiable, built-in native objects, as defined by [ECMASCRIPT].

Table 6 ECMAScript Instantiable Native Object Support

`Array`	`Boolean`	`Date`
`Error`	`Function`	`Number`
`Object`	`Regexp`	`String`

5.3.1.1.2.1 Date Object

When a Date object is constructed so as to indicate the current time of day, it should derive the time of day from accurate time information available from the application delivery system.

5.3.1.1.2.2 String Object

In addition to implementing the semantics defined by [ECMASCRIPT], Section 15.5, each instance of a `String` object shall implement the following interface:

```
interface StringExt
{
  /* methods */
  String anchor(in String name);
  String big();
  String blink();
  String bold();
  String fixed();
  String fontcolor(in String color);
  String fontsize(in String size);
  String italics();
  String link(in String href);
  String small();
  String strike();
  String sub();
  String sup();
};
```

The following subsections describe the entirety of `StringExt` interface semantics. Use of this interface is deprecated in non-legacy content.

> *Note:* This extended interface is provided to obtain lexical interoperability with certain legacy script content.

5.3.1.1.2.2.1 `StringExt::anchor`

This string valued method shall return a string representing the current `String` object surrounded by a start and end tag as follows:

```
<a name="name">original string</a>
```

where the *name* argument is used to specify the value of the *a* (anchor) element's *name* attribute.

5.3.1.1.2.2.2 `StringExt::big`

This string valued method shall return a string representing the current `String` object surrounded by a start and end tag as follows

```
<big>original string</big>
```

5.3.1.1.2.2.3 `StringExt::blink`

This string valued method shall return a string representing the current `String` object surrounded by a start and end tag as follows:

```
<span style="text-decoration: blink">original string</span>
```

5.3.1.1.2.2.4 `StringExt::bold`

This string valued method shall return a string representing the current `String` object surrounded by a start and end tag as follows

```
<b>original string</b>
```

5.3.1.1.2.2.5 `StringExt::fixed`

This string valued method shall return a string representing the current `String` object surrounded by a start and end tag as follows

```
<tt>original string</tt>
```

5.3.1.1.2.2.6 `StringExt::fontcolor`

This string valued method shall return a string representing the current `String` object surrounded by a start and end tag as follows:

```
<span style="color: color">original string</span>
```

where the *color* argument is used to specify the value of the *color* style property.

5.3.1.1.2.2.7 `StringExt::fontsize`

This string valued method shall return a string representing the current `String` object surrounded by a one or more *span* element start and end tags.

If the value of the *size* argument is an integer in the range [1-7], then the following shall be returned:

```
<span style="font-size: absolute size">original string</span>
```

where the value of *absolute size* is determined according to the value of the *size* argument as follows:

Table 7 Legacy Font Size Attribute Mappings

1	xx-small
2	x-small
3	small
4	medium
5	large
6	x-large
7	xx-large

If the value of the *size* argument is an integer in the range [1-7] preceded by either the `'+'` (plus sign) delimiter or the `'-'` (minus sign) delimiter, then the following shall be returned:

```
...<span style="font-size: relative size">original string</span>...
```

where (1) the value of *relative size* is either `larger` or `smaller` depending upon whether the preceding delimiter is `'+'` or `'-'`, respectively, and (2) `...` denotes additional surrounding *span* element start and end tags such that the number of surrounding span elements is equal to the integer value of the *size* argument.

The following script fragment, and, in particular, the *fontsize* function, provides a sample implementation of this method's semantics:

```
function mapsize ( size )
{
  var map = [ "xx-small", "x-small", "small", "medium", "large", "x-large", "xx-large" ];
  if ( isNaN ( parseInt ( size ) ) )
      size = 4;
  else if ( size < 1 )
      size = 1;
  else if ( size > 7 )
      size = 7;
  return map [ size - 1 ];
}

function fontsize ( string, size )
{
  if ( size.substring ( 0, 1 ) == '+' ) {
    if ( size.substring ( 1 ) > 0 )
      return "<" + "span style=\"font-size: larger\">" +
             fontsize ( string, "+" + ( size.substring ( 1 ) - 1 ) ) +
             "</" + "span>";
    else
      return string;
  } else if ( size.substring ( 0, 1 ) == '-' ) {
    if ( size.substring ( 1 ) > 0 )
      return "<" + "span style=\"font-size: smaller\">" +
             fontsize ( string, "-" + ( size.substring ( 1 ) - 1 ) ) +
             "</" + "span>";
    else
      return string;
  } else if ( ( size >= 1 ) && ( size <= 7 ) ) {
    return "<" + "span style=\"font-size: " + mapsize ( size ) + "\">" +
           string +
           "</" + "span>";
  } else
    return string;
};
```

5.3.1.1.2.2.8 `StringExt::italics`

This string valued method shall return a string representing the current `String` object surrounded by a start and end tag as follows

```
<i>original string</i>
```

5.3.1.1.2.2.9 `StringExt::link`

This string valued method shall return a string representing the current `String` object surrounded by a start and end tag as follows:

```
<a href="href">original string</a>
```

where the *href* argument is used to specify the value of the *a* (anchor) element's *href* attribute.

5.3.1.1.2.2.10 `StringExt::small`

This string valued method shall return a string representing the current `String` object surrounded by a start and end tag as follows

```
<small>original string</small>
```

5.3.1.1.2.2.11 `StringExt::strike`

This string valued method shall return a string representing the current `String` object surrounded by a start and end tag as follows:

```
<span style="text-decoration: line-through">original string</span>
```

5.3.1.1.2.2.12 `StringExt::sub`

This string valued method shall return a string representing the current `String` object surrounded by a start and end tag as follows

```
<sub>original string</sub>
```

5.3.1.1.2.2.13 `StringExt::sup`

This string valued method shall return a string representing the current `String` object surrounded by a start and end tag as follows

```
<sup>original string</sup>
```

5.3.1.2 Host Objects

An ECMAScript *host object* is an object whose meta-behavior (as object) is defined by [ECMASCRIPT], Section 8.6, *The Object Type*, but whose particular form and semantics are not defined by the language. Host objects extend the functionality of ECMAScript beyond that functionality provided by native objects.

This specification permits the use of a number of host objects in order to support (1) access to and modification of the decoded representation of a declarative application and (2) access to and modification of the state of the user agent. The specifications of these two types of host objects are referred to as the *Document Object Model* (DOM) and the *Environment Object Model* (EOM), respectively.

> *Note:* Both document and environment object models are frequently conflated. This specification does not formally distinguish between the DOM and EOM, and uses the common term DOM to refer to both.

A user agent which supports this facility shall implement the host objects specified by the following subsections in accordance with *Document Object Model, Level 2* [DOM2], and as extended and restricted below:

- Core
- Core Extensions
- HTML
- Views
- StyleSheets
- CSS
- Event Sets
- Environment

> *Note:* The purpose of the Core, Core Extensions, Views, StyleSheets, CSS, and Event Sets objects is to satisfy general requirements for access to, manipulation of, and interaction with a declarative application by procedural means within the context of script content.

> *Note:* The purpose of the HTML and Environment host objects, defined below in Sections 5.3.1.2.3, *HTML Module Objects*, and 5.3.1.2.9, *Environment Module Objects*, is to satisfy interoperability requirements for legacy script content usage as supported by [DOM0].

> *Note:* See *Errata of Document Object Model Level 2 Specifications* [DOM2-ERR] for corrections and clarifications regarding [DOM2], [DOM2-EVENTS], [DOM2-HTML], [DOM2-STYLE], and [DOM2-VIEWS].

A user agent shall implement these objects and their extended interfaces in such a manner that if a valid script content entity or fragment which references only the objects, object properties, and object methods of [DOM0] is evaluated, then that evaluation will not produce an error condition for reasons of either (1) reference to an unknown property or (2) reference to an unknown method. However, evaluation may produce an error condition for reasons of semantic

restrictions placed on properties or methods as specified below. When such a restriction does produce an error condition, it shall do so by means of throwing an ECMAScript exception.

> *Note:* In the following subsections, the term *interface* is used to refer to both (1) a collection of object properties and methods and (2) an IDL (interface definition language) specification of such a collection. In addition, the term *property* is used to refer to both (1) a state variable of an object, whether mutable (read-write) or immutable (read-only), and (2) an IDL attribute specification.

> *Note:* IDL (interface definition language) uses the keyword *attribute* to designate an object property. To avoid confusion between IDL attributes and XML attributes, we use the term *property* when referring to interface functionality expressed as an IDL attribute. This is consistent with an ECMAScript binding of IDL, since ECMAScript uses the term *property* as well.

Except for those cases which would not otherwise conform to [ECMASCRIPT] semantics, the *DontEnum* attribute shall not be present on properties which correspond to the methods and properties of all host objects permitted by this specification; that is, enumeration of these methods and properties shall be supported.

> *Note:* See [ECMASCRIPT], Sections 8.6 and 12.6.4, for further information on use of the *DontEnum* property attribute.

All host objects shall implement the internal `[[HasInstance]]` method in order to support the `instanceof` operator.

> *Note:* See [ECMASCRIPT], Section 15.3.5.3., for further information on use of the internal `[[HasInstance]]` method.

5.3.1.2.1 Core Module Objects

A script content entity or fragment may reference and a user agent which supports this facility shall implement the following host objects, defined by [DOM2], Section 1.2, *Fundamental Interfaces*, in conjunction with [DOM2], Appendix E, *ECMAScript Language Binding*.

With the exception of the `DOMException` object, none of the following objects is directly instantiable; that is, the internal `[[Construct]]` property for the object shall remain undefined. Any attempt to construct an object using the `new` operator shall cause an ECMAScript *TypeError* to be raised.

> *Note:* See [ECMASCRIPT], Section 11.2.2, for further information on use of the internal `[[Construct]]` property.

Table 8 DOM Core Interface Support

`Attr`	`CharacterData`	`Comment`
`Document`	`DocumentFragment`	`DOMException`
`DOMExceptionExt`	`DOMImplementation`	`Element`
`NamedNodeMap`	`Node`	`NodeList`
`Text`		

Certain methods and properties exposed by means of these host objects may be invoked or mutated in such a manner as to cause a valid, decoded document instance or fragment to become invalid.

Script content shall not cause a valid decoded document instance or fragment to become invalid by invoking the methods or mutating the properties of these host objects.

> *Note:* This constraint applies only to document instances or fragments produced as a result of decoding a source document. It does not apply to document instances or fragments synthesized by script content.

If use of this property causes a valid decoded document instance or fragment to become invalid, then the user agent should cause a `VALIDATION_ERR` exception to be raised. If an entity of

a declarative application uses content type `text/ecmascript` and invokes a method or mutates a property of these host objects in such a manner as to cause a valid, decoded document instance or fragment to become invalid, then the user agent may abort the application.

> *Note:* See Section 5.3.1.2.1.3.5 for more information on `VALIDATION_ERR`.

The methods and properties of *Core* host objects which permit the violation of validity are specified explicitly in the following subsections.

5.3.1.2.1.1 Attr Interface

5.3.1.2.1.1.1 `Attr::ownerElement`

If the `Attr::specified` property has the value `false`, then the `Attr::ownerElement` property may have the value `null` even when the attribute is in use.

> *Note:* By permitting the value of this property to remain `null`, an implementation of a declarative application environment may share instances of default attributes, thus achieving considerable memory savings.

5.3.1.2.1.1.2 `Attr::specified`

The `Attr::specified` property shall have the value `false` if the attribute has no assigned value in the document and has a default value in the DTD.

5.3.1.2.1.1.3 `Attr::value`

The `Attr::value` property may be used to mutate the value of an attribute `Node` which would invalidate a valid document instance or fragment. For example, the value of the *type* attribute of an *input* element could be changed to `file`.

5.3.1.2.1.2 Document Interface

5.3.1.2.1.2.1 `Document::createEntityReference`

If a user agent does not support the *EntityReference* interface and this method is invoked, then the user agent shall cause a `NOT_SUPPORTED_ERR` exception to be raised.

> *Note:* See Section 5.3.1.2.2.3 for further information on the *EntityReference* interface.

5.3.1.2.1.3 DOMException Interface

The `DOMException` interface shall be implemented by the `DOMException` exception object according to [DOM2], Appendix E, *ECMAScript Language Binding*.

Exceptional conditions defined by [DOM2] or by this specification shall be raised by use of an explicit or implied *ThrowStatement* as specified by [ECMASCRIPT], Section 12.13. The value of the throw statement's exception argument shall be a `DOMException` object instance whose `code` property is either (1) set to the specific exception type code defined by [DOM2] or (2) set to the special value 65535 (0xFFFF) in the case of an extended exception code as defined below.

In addition to implementing the `DOMException` exception interface as defined by [DOM2], Section 1.2, each instance of a `DOMException` exception object shall implement the following exception interface:

```
exception DOMExceptionExt
{
  unsigned short codeExtension;
};

// Extended Exception Codes
const unsigned short VALIDATION_ERR = 1;
const unsigned short NO_CLOSE_ALLOWED_ERR = 2;
```

The following subsections describe constraints on `DOMException` exception interface semantics as well as the entirety of `DOMExceptionExt` exception interface semantics.

5.3.1.2.1.3.1 `DOMException::[[Construct]]`

The internal `[[Construct]]` method for the `DOMException` object shall be implemented by a user agent which supports this facility. This method shall take two arguments: (1) an argument used to initialize `DOMException::code`, and (2) an argument used to initialize `DOMExceptionExt::codeExtension`. If the second argument is non-zero, the first argument shall be 65535 (0xFFFF).

> *Note:* See [ECMASCRIPT], Section 13.2.2, for further information on the internal `[[Construct]]` property.

5.3.1.2.1.3.2 `NO_MODIFICATION_ALLOWED_ERR`

A NO_MODIFICATION_ALLOWED_ERR exception shall be raised upon any attempt to modify a read-only property.

5.3.1.2.1.3.3 `DOMExceptionExt::codeExtension`

This immutable, non-negative integral number valued property denotes an extended exception code. For extended exceptions defined by this specification, this property shall have a non-zero, positive value.

5.3.1.2.1.3.4 `NO_CLOSE_ALLOWED_ERR`

A `NO_CLOSE_ALLOWED_ERR` exception shall be raised upon certain attempts to close a `Window` object. See Section 5.3.1.2.9.4.3 for further information.

5.3.1.2.1.3.5 `VALIDATION_ERR`

A `VALIDATION_ERR` exception should be raised upon any attempt to modify a source document instance or one of its nodes in such a manner as to cause a valid document instance to become invalid.

5.3.1.2.1.4 DOMImplementation Interface

5.3.1.2.1.4.1 `DOMImplementation::hasFeature`

This method shall return *true* for the following standard feature strings defined by [DOM2], [DOM2-VIEWS], [DOM2-STYLE], and [DOM2-EVENTS]: `"Core"`, `"Views"`, `"StyleSheets"`, `"CSS"`, `"Events"`, `"UIEvents"`, `"MouseEvents"`, `"MutationEvents"`, and `"HTMLEvents"`.

In addition, this method shall return *true* for the feature string `"XML"` if the user agent supports internal declaration subsets.

In addition, this method shall return *true* for the following feature strings defined by this specification: `"org.atsc.dom.core"`, `"org.atsc.dom.environment"`, `"org.atsc.dom.events"`, `"org.atsc.dom.html"`, `"org.atsc.dom.native"` and `"org.atsc.dom.views"`.

5.3.1.2.1.4.2 `DOMImplementation::createDocumentType`

If the public identifier supplied as an argument to this method is not one of those specified in Section 5.1.1.4 Document Type Declaration, then the user agent may cause a `NOT_SUPPORTED_ERR` exception to be raised.

5.3.1.2.1.4.3 `DOMImplementation::createDocument`

If the public identifier of the document type supplied as an argument to this method is not one of those specified in Section 5.1.1.4 Document Type Declaration, then the user agent may cause a `NOT_SUPPORTED_ERR` exception to be raised.

5.3.1.2.1.5 Element Interface

5.3.1.2.1.5.1 `Element::removeAttribute`

The `Element::removeAttribute` method may be used to remove an attribute from an `Element` which would invalidate a valid document instance or fragment. For example, the *type* attribute could be removed from a *script* `Element`.

5.3.1.2.1.5.2 `Element::removeAttributeNS`

The `Element::removeAttributeNS` method may be used to remove an attribute from an `Element` which would invalidate a valid document instance or fragment. For example, the *type* attribute could be removed from a *script* `Element`.

5.3.1.2.1.5.3 `Element::removeAttributeNode`

The `Element::removeAttributeNode` method may be used to remove an attribute from an `Element` which would invalidate a valid document instance or fragment. For example, the *type* attribute could be removed from a *script* `Element`.

5.3.1.2.1.5.4 `Element::setAttribute`

The `Element::setAttribute` method may be used to mutate the value of an attribute of an `Element` which would invalidate a valid document instance or fragment. For example, the value of the *type* attribute of an *input* element could be changed to `file`.

5.3.1.2.1.5.5 `Element::setAttributeNS`

The `Element::setAttributeNS` method may be used to mutate the value of an attribute of an `Element` which would invalidate a valid document instance or fragment. For example, the value of the *type* attribute of an *input* element could be changed to `file`.

5.3.1.2.1.5.6 `Element::setAttributeNode`

The `Element::setAttributeNode` method may be used to mutate the value of an attribute of an `Element` which would invalidate a valid document instance or fragment. For example, the value of the *type* attribute of an *input* element could be changed to `file`.

5.3.1.2.1.5.7 `Element::setAttributeNodeNS`

The `Element::setAttributeNodeNS` method may be used to mutate the value of an attribute of an `Element` which would invalidate a valid document instance or fragment. For example, the value of the *type* attribute of an *input* element could be changed to `file`.

5.3.1.2.1.6 NamedNodeMap Interface

5.3.1.2.1.6.1 `NamedNodeMap::removeNamedItem`

The `NamedNodeMap::removeNamedItem` method may be used to mutate the value of certain `NamedNodeMap` instances which would invalidate a valid document instance or fragment. For example, the value the `Node::attributes` property could be mutated so as to remove a required attribute.

5.3.1.2.1.6.2 `NamedNodeMap::removeNamedItemNS`

The `NamedNodeMap::removeNamedItemNS` method may be used to mutate the value of certain `NamedNodeMap` instances which would invalidate a valid document instance or fragment. For example, the value the `Node::attributes` property could be mutated so as to remove a required attribute.

5.3.1.2.1.6.3 `NamedNodeMap::setNamedItem`

The `NamedNodeMap::setNamedItem` method may be used to mutate the value of certain `NamedNodeMap` instances which would invalidate a valid document instance or fragment. For example, the value the `Node::attributes` property could be mutated so as to replace an attribute with an invalid attribute.

5.3.1.2.1.6.4 `NamedNodeMap::setNamedItemNS`

The `NamedNodeMap::setNamedItemNS` method may be used to mutate the value of certain `NamedNodeMap` instances which would invalidate a valid document instance or fragment. For example, the value the `Node::attributes` property could be mutated so as to replace an attribute with an invalid attribute.

5.3.1.2.1.7 Node Interface

5.3.1.2.1.7.1 `Node::appendChild`

The `Node::appendChild` method may be used to insert a child `Node` which would invalidate a valid document instance or fragment. For example, a *body* element node could be inserted as a child of another *body* element node.

5.3.1.2.1.7.2 `Node::attributes`

The `Node::attributes` property provides access to the *attribute* nodes of an element *node*. In such a case, the `NamedNodeMap` typed value of this property may be mutated in such a manner as to invalidate a valid document instance or fragment. For example, a required attribute `Node` could be removed, leaving an element `Node` without a required attribute.

5.3.1.2.1.7.3 `Node::insertBefore`

The `Node::insertBefore` method may be used to insert a child `Node` which would invalidate a valid document instance or fragment. For example, a *body* element node could be inserted as a child of another *body* element node.

5.3.1.2.1.7.4 `Node::nodeValue`

The `Node::nodeValue` property may be used to mutate the value of an attribute `Node` which would invalidate a valid document instance or fragment. For example, the value of the *type* attribute of an *input* element could be changed to `file`.

5.3.1.2.1.7.5 `Node::prefix`

The `Node::prefix` property may be used to mutate the XML namespace prefix of an element or attribute `Node` which would invalidate a valid document instance or fragment. For example, the XML namespace prefix of the *a* element could be changed to `none`.

5.3.1.2.1.7.6 `Node::removeChild`

The `Node::removeChild` method may be used to remove a child `Node` which would invalidate a valid document instance or fragment. For example, a *body* element node could be removed as the child of an *html* element node.

5.3.1.2.1.7.7 `Node::replaceChild`

The `Node::replaceChild` method may be used to replace a child `Node` which would invalidate a valid document instance or fragment. For example, the *body* element node child of an *html* element node could be replaced with a *head* element node.

5.3.1.2.2 Core Extension (XML) Module Objects

A script content entity or fragment may reference and a user agent which supports this facility shall implement the following host objects, defined by [DOM2], Section 1.3, *Extended Interfaces*, in conjunction with [DOM2], Appendix E, *ECMAScript Language Binding*.

None of the following objects is directly instantiable; that is, the internal `[[Construct]]` property for the object shall remain undefined. Any attempt to construct an object using the `new` operator shall cause an ECMAScript *TypeError* to be raised.

> *Note:* See [ECMASCRIPT], Section 11.2.2, for further information on use of the internal `[[Construct]]` property.

Table 9 DOM Core Extensions Interface Support

`CDATASection`	`DocumentType`	`Entity`
`EntityReference`	`Notation`	`ProcessingInstruction`

5.3.1.2.2.1 DocumentType Interface

5.3.1.2.2.1.1 `DocumentType::entities`

If a user agent does not support internal declaration subsets (see Section 5.1.1.4) and does not parse an external declaration subset, then this property's value shall be an empty `NamedNodeMap`.

5.3.1.2.2.1.2 `DocumentType::internalSubset`

If a user agent does not support internal declaration subsets (see Section 5.1.1.4), then this property's value shall be an empty string.

5.3.1.2.2.1.3 `DocumentType::notations`

If a user agent does not support internal declaration subsets and does parse the external declaration subset, then this property's value shall be an empty `NamedNodeMap`.

Note: See Section 5.1.1.4 for more information on support of internal declaration subsets.

5.3.1.2.2.2 Entity Interface

If a user agent does not support internal declaration subsets and does not parse the external declaration subset, then the user agent need not implement this interface or its associated object.

Note: See Section 5.1.1.4 for more information on support of internal declaration subsets.

5.3.1.2.2.3 EntityReference Interface

If a user agent does not support internal declaration subsets and does not parse the external declaration subset, then the user agent need not implement this interface or its associated object.

Note: See Section 5.1.1.4 for more information on support of internal declaration subsets.

5.3.1.2.2.4 Notation Interface

If a user agent does not support internal declaration subsets and does not parse the external declaration subset, then the user agent need not implement this interface or its associated object.

Note: See Section 5.1.1.4 for more information on support of internal declaration subsets.

5.3.1.2.3 HTML Module Objects

A script content entity or fragment may reference and a user agent which supports this facility shall implement the following host objects and interfaces, defined by [DOM2-HTML].

None of the following objects is directly instantiable; that is, the internal `[[Construct]]` property for the object shall remain undefined. Any attempt to construct an object using the `new` operator shall cause an ECMAScript *TypeError* to be raised.

Note: See [ECMASCRIPT], Section 11.2.2, for further information on use of the internal `[[Construct]]` property.

Table 10 DOM HTML Interface Support

`HTMLAnchorElement`	`HTMLAnchorElementExt`	`HTMLBodyElement`
`HTMLCollection`	`HTMLDocument`	`HTMLDocumentExt`

HTMLElement	HTMLFormElement	HTMLFormElementExt
HTMLImageElement	HTMLImageElementExt	HTMLInputElement
HTMLObjectElement	HTMLObjectElementExt	HTMLOptionElement
HTMLOptionsCollection	HTMLSelectElement	HTMLTextAreaElement
HTMLTriggerObjectElementExt		

Note: These objects represent a subset of the interfaces defined by [DOM2-HTML]. Some of these objects are additionally extended by the interfaces defined in the following sections. The purpose of these extended interfaces is to provide strict lexical compatibility with [DOM0] usage.

The presence of support for this set of interfaces is indicated by means of a feature string `"org.atsc.dom.html"`, which may be used with the `DOMImplementation::hasFeature` method.

When a source document entity containing `application/xhtml+xml` content is decoded, the user agent will instantiate these objects in order to populate a document object instance hierarchy. These objects may be referenced and mutated during the decoding process by script content embedded in the document itself.

As an optimization, a user agent may defer the instantiation of a document object instance hierarchy until a reference to the instance occurs. This optimization permits the decoding of source documents that do not make reference to the document object instance without incurring the overhead of creating a document object instance hierarchy that is never used.

5.3.1.2.3.1 HTMLAnchorElement Object

In addition to implementing the `HTMLAnchorElement` interface as defined by [DOM2-HTML], each instance of an `HTMLAnchorElement` object shall implement the following interface:

```
interface HTMLAnchorElementExt
{
  /* read-write properties */
  attribute DOMString hash;
  attribute DOMString host;
  attribute DOMString hostname;
  attribute DOMString pathname;
  attribute DOMString port;
  attribute DOMString protocol;
  attribute DOMString search;
};
```

The following subsections describe constraints on `HTMLAnchorElement` interface semantics as well as the entirety of `HTMLAnchorElementExt` interface semantics.

5.3.1.2.3.1.1 `HTMLAnchorElement::[[DefaultValue]]`

The internal `[[DefaultValue]]` method for the `HTMLAnchorElement` object shall return the value of `HTMLAnchorElement::href` in the form of a primitive string value irrespective of the value of *hint*. See [ECMASCRIPT], Section 8.6.2.6.

5.3.1.2.3.1.2 `HTMLAnchorElementExt::hash`

This mutable, string valued property denotes the *fragment* portion of the parsed representation of the anchor element's *href* attribute value. See [URI], Section 4.1.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `HTMLAnchorElement::href`.

5.3.1.2.3.1.3 `HTMLAnchorElementExt::host`

This mutable, string valued property denotes the *hostport* portion of the parsed representation of the anchor element's *href* attribute value. See [URI], Section 3.2.2.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `HTMLAnchorElement::href`.

5.3.1.2.3.1.4 `HTMLAnchorElementExt::hostname`

This mutable, string valued property denotes the *host* portion of the parsed representation of the anchor element's *href* attribute value. See [URI], Section 3.2.2.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `HTMLAnchorElement::href` and `HTMLAnchorElementExt::host`.

5.3.1.2.3.1.5 `HTMLAnchorElementExt::pathname`

This mutable, string valued property denotes the *abs_path* or *rel_path* portion of the parsed representation of the anchor element's *href* attribute value, depending on whether the URI is an *absoluteURI* or *relativeURI*, respectively. See [URI], Sections 3 and 5.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `HTMLAnchorElement::href`.

5.3.1.2.3.1.6 `HTMLAnchorElementExt::port`

This mutable, string valued property denotes the *port* portion of the parsed representation of the anchor element's *href* attribute value. See [URI], Section 3.2.2.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `HTMLAnchorElement::href` and `HTMLAnchorElementExt::host`.

5.3.1.2.3.1.7 `HTMLAnchorElementExt::protocol`

This mutable, string valued property denotes the *scheme* portion of the parsed representation of the anchor element's *href* attribute value, including the `':'` delimiter (colon) that follows the *scheme*. See [URI], Section 3.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `HTMLAnchorElement::href`.

5.3.1.2.3.1.8 `HTMLAnchorElementExt::search`

This mutable, string valued property denotes the *query* portion of the parsed representation of the anchor element's *href* attribute value, including the `'?'` delimiter (question mark) that precedes the *query*. See [URI], Sections 3 and 3.4.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `HTMLAnchorElement::href` and `HTMLAnchorElementExt::pathname`.

5.3.1.2.3.2 HTMLBodyElement Object

None of the properties of the `HTMLBodyElement` interface corresponds to XDML or XHTML Basic attributes; rather, the functionality of these properties is specified by using stylesheet content. Notwithstanding the inability to directly set these properties by using attributes, these properties shall support their [HTML] presentation semantics. In particular, mutation of their values shall cause a presentation side effect as if the element's style were changed by mutating the `CSSStyleDeclaration` value of the `ElementCSSInlineStyle::style` property of the `HTMLBodyElement` object.

> *Note:* XHTML Host Language Conformant Document Types other than XDML and XHTML Basic may support the legacy attributes which correspond to the properties of `HTMLBodyElement`.

5.3.1.2.3.3 HTMLDocument Object

In addition to implementing the `HTMLDocument` interface as defined by [DOM2-HTML], each instance of an `HTMLDocument` object shall implement the following interface:

```
interface HTMLDocumentExt
{
  /* read-only properties */
  readonly attribute DOMString location;
  readonly attribute DOMString lastModified;
  readonly attribute Window window;
  /* read-write properties */
  attribute DOMString alinkColor;
  attribute DOMString vlinkColor;
  attribute DOMString linkColor;
  attribute DOMString bgColor;
  attribute DOMString fgColor;
  /* methods */
  void clear();
};
```

The following subsections describe constraints on `HTMLDocument` interface semantics as well as the entirety of `HTMLDocumentExt` interface semantics.

5.3.1.2.3.3.1 `HTMLDocument::anchors`

This immutable, object value property shall reference an immutable `HTMLCollection` object that, in turn, references zero or more mutable `HTMLAnchorElement` objects.

An `HTMLAnchorElement` shall be instantiated and added to the `HTMLCollection` object referenced by this property for each *a* (anchor) element decoded, provided the element specifies a *name* attribute.

If an *a* (anchor) element specifies both *name* and *href* attributes, then only one instance of `HTMLAnchorElement` shall be instantiated, and that instance shall be added to both `HTMLCollection` objects referenced by this property and `HTMLDocument::links`. See Section 5.3.1.2.3.3.8 for further information.

> *Note:* Although script content may not mutate the referenced `HTMLCollection` object, the user agent will mutate this object as a side effect of loading the document and decoding individual *a* (anchor) elements. This mutation of the referenced `HTMLCollection` object by the user agent will occur asynchronously with respect to script content evaluation.

5.3.1.2.3.3.2 `HTMLDocument::applets`

This immutable, object value property shall either (1) reference an immutable `HTMLCollection` object that, in turn, references zero or more mutable `HTMLObjectElement` objects or (2) have a null value according to whether the user agent supports active object content or not, respectively.

If the user agent supports active object content, then an `HTMLObjectElement` shall be instantiated and added to the `HTMLCollection` object referenced by this property for each *object* element that references active object content.

> *Note:* Although script content may not mutate the referenced `HTMLCollection` object, the user agent will mutate this object as a side effect of loading the document and decoding individual *object* elements. This mutation of the referenced `HTMLCollection` object by the user agent will occur asynchronously with respect to script content evaluation.

For each named `HTMLObjectElement` object the user agent adds to the referenced `HTMLCollection` object, a new property shall be added to the current `HTMLDocument` object, where the name of the property is the value of the *name* property of that `HTMLObjectElement` object and the value of the property is the `HTMLObjectElement` object itself. If an identically named property already exists on the current `HTMLDocument` object, then a property shall not be added and the value of the pre-existing, identically named property shall not be changed.

5.3.1.2.3.3.3 `HTMLDocument::body`

A user agent may disallow mutation of this mutable property, in which case an attempt to mutate this property shall cause a `NO_MODIFICATION_ALLOWED_ERR` exception to be raised.

5.3.1.2.3.3.4 `HTMLDocument::cookie`

This mutable, string valued property denotes persistent state information that (1) is associated with the application resource which represents the current frame or current document entity and (2) is composed of information described by the *cookies* non-terminal of [HTTP-STATE], Section 4.2.2.

If no persistent state information is available for the application resource which represents the current frame or current document entity, then this property's value shall be an empty string.

> *Note:* In the following, the phrase *persistent state item* refers to a single *cookie*. The value of this property denotes zero or more persistent state items.

When this property is accessed by a read operation, all persistent state items whose *domain* and *path* attributes are unifiable with an absolutized form of the resource identifier (URI) of the application resource which represents the current frame or current document entity shall be returned as a single string, with each persistent state item's name-value pair concatenated into a list of name-value pairs, each list item being separated by a `';'` (semicolon) delimiter. Any attribute-value pairs associated with each persistent state item are not returned in the resulting value.

When accessed by a write operation, the value to be written shall be a string that adheres to the *cookie* non-terminal of [HTTP-STATE]; that is, it shall be a single name-value pair followed by zero or more optional attribute-value pairs. If no *domain* attribute is specified, then the domain attribute for the new value shall default to the *host* portion of an absolutized form of the resource identifier of the application resource which represents the current frame or current document entity. If no *path* attribute is specified, then the path attribute for the new value shall default to the *abs_path* portion of an absolutized form of the resource identifier of the application resource which represents the current frame or current document entity. If no *max-age* attribute is specified, then the max-age attribute for the new value shall default to 60 (seconds). If a persistent state item with the specified name is already associated with the current frame or current document entity's associated application resource, then the new value as well as the new attributes shall replace the old value and attributes. If a *max-age* attribute of zero is specified for the new value, then any existing persistent state item of the specified name shall be removed from the persistent state item store.

When mutating the value of this property, if the new value does not adhere to the *cookie* syntax specified by [HTTP-STATE], then the user agent shall cause a `SYNTAX_ERR` exception to be raised.

A user agent shall support the semantics of the persistent state attribute-value pairs specified by [HTTP-STATE]. In addition, if a declarative application is designated as a legacy application, then the application may employ and a user agent shall support an attribute-value pair whose name is *expires* and whose value is an *rfc1123-date*, in accordance with the syntax prescribed by [DASE], Section 6.8.1. Otherwise, if an application is not a declarative application or is not designated as a legacy application, then the presence of an *expires* attribute shall cause a `SYNTAX_ERR` exception to be raised.

For the initial (first) instantiation of a particular declarative application by a user agent, persistent state data shall not be initially available; however, data may be available in a subsequent processing of the document in a single user agent session if this property has been mutated by script content.

> *Note:* The precise nature of a *user agent session* is not defined by this specification.

A user agent should provide storage for at least one cookie during a single session such that at least 4096 (4K) bytes of persistent store items may be retained during the session. A user

agent should attempt to retain at least twenty (20) such cookies in non-volatile store between user agent sessions.

A user agent should digest or checksum each retained persistent store item in order to prevent unintended modification of the item. Retained items which fail a subsequent digest or checksum test should be deleted from the persistent state store.

A user agent shall take measures to prevent unexpected sharing of persistent store items as described in [HTTP-STATE], Section 7.3.

Script content shall not rely upon a user agent being able to store a particular persistent storage item between two instantiations of the same application, even in the context of a single user agent session.

If a user agent makes use of a local file system to store a persistent store item, then the portion of the local file system used for this purpose shall be made inaccessible to any functionality of a DASE Application which may attempt to directly access the item's store.

5.3.1.2.3.3.5 `HTMLDocument::domain`

This immutable, string valued property denotes the domain name of the server that issued the application resource that represents the current document entity, if known, or shall be null.

For applications whose initial entity resource is delivered by a broadcast channel, this property shall be the domain name of the resource reference of the application's initial entity.

5.3.1.2.3.3.6 `HTMLDocument::forms`

This immutable, object value property shall reference an immutable `HTMLCollection` object that, in turn, references zero or more mutable `HTMLFormElement` objects.

An `HTMLFormElement` shall be instantiated and added to the `HTMLCollection` object referenced by this property for each *form* element decoded.

> *Note:* Although script content may not mutate the referenced `HTMLCollection` object, the user agent will mutate this object as a side effect of loading the document and decoding individual *form* elements. This mutation of the referenced `HTMLCollection` object by the user agent will occur asynchronously with respect to script content evaluation.

For each named `HTMLFormElement` object the user agent adds to the referenced `HTMLCollection` object, a new property shall be added to the current `HTMLDocument` object, where the name of the property is the value of the *name* property of that `HTMLFormElement` object and the value of the property is the `HTMLFormElement` object itself. If an identically named property already exists on the current `HTMLDocument` object, then a property shall not be added and the value of the pre-existing, identically named property shall not be changed.

5.3.1.2.3.3.7 `HTMLDocument::images`

This immutable, object value property shall reference an immutable `HTMLCollection` object that, in turn, references zero or more mutable `HTMLImageElement` or `HTMLObjectElement` objects.

An `HTMLImageElement` shall be instantiated and added to the `HTMLCollection` object referenced by this property for each *img* element decoded.

An `HTMLObjectElement` shall be instantiated and added to the `HTMLCollection` object referenced by this property for each *object* element decoded if (1) the *classid* attribute is not specified and (2) the *data* attribute is specified and references content which may be interpreted as a graphical image.

> *Note:* Although script content may not mutate the referenced `HTMLCollection` object, the user agent will mutate this object as a side effect of loading the document and decoding individual *img* (image) or *object* elements. This mutation of the referenced

`HTMLCollection` object by the user agent will occur asynchronously with respect to script content evaluation.

For each named `HTMLImageElement` or `HTMLObjectElement` object the user agent adds to the referenced `HTMLCollection` object, a new property shall be added to the current `HTMLDocument` object, where the name of the property is the value of the *name* property of that `HTMLImageElement` or `HTMLObjectElement` object and the value of the property is the `HTMLImageElement` or `HTMLObjectElement` object itself. If an identically named property already exists on the current `HTMLDocument` object, then a property shall not be added and the value of the pre-existing, identically named property shall not be changed.

5.3.1.2.3.3.8 `HTMLDocument::links`

This immutable, object value property shall reference an immutable `HTMLCollection` object that, in turn, references zero or more mutable `HTMLAnchorElement` or `HTMLElement` objects.

An `HTMLAnchorElement` or an `HTMLElement` shall be instantiated and added to the `HTMLCollection` object referenced by this property for each *a* (anchor) and *area* element decoded, respectively, provided the element specifies an *href* attribute.

If an *a* (anchor) element specifies both *name* and *href* attributes, then only one instance of `HTMLAnchorElement` shall be instantiated, and that instance shall be added to both `HTMLCollection` objects referenced by this property and `HTMLDocument::anchors`. See Section 5.3.1.2.3.3.1 for further information.

Note: Although script content may not mutate the referenced `HTMLCollection` object, the user agent will mutate this object as a side effect of loading the document and decoding individual *a* (anchor) or *area* elements. This mutation of the referenced `HTMLCollection` object by the user agent will occur asynchronously with respect to script content evaluation.

5.3.1.2.3.3.9 `HTMLDocument::open`

A user agent shall ignore any arguments specified by the caller of this method.

5.3.1.2.3.3.10 `HTMLDocument::title`

A user agent may disallow mutation of this mutable property, in which case an attempt to mutate this property shall cause a `NO_MODIFICATION_ALLOWED_ERR` exception to be raised.

5.3.1.2.3.3.11 `HTMLDocument::write`

If a declarative application is a legacy application, this void valued method shall cause the insertion of *text* into the current entity's input stream immediately subsequent to the close tag of the *script* element in which this method is invoked; otherwise, if not a legacy application, a user agent may abort the application. If this method is invoked multiple times during the evaluation of script content in a single script element, then the *text* inserted by each invocation shall be serialized into the input stream in evaluation order.

If this method is invoked in any execution context other than during the processing of a *script* element, then no side effect shall result.

Note: This method has no effect if invoked within an event handler or scheduled statement timer context.

The aggregate text to be inserted into the input stream by zero or more invocations of this method within the context of a single script element shall be valid `text/html` content according to either the traditional or frameset document types specified by [HTML]. Furthermore, this aggregate text shall be a well-formed document entity fragment; namely, it shall be either element or mixed element content and any element contained therein shall be complete (i.e., contain the entirety of its start tag, content, and end tag, if required).

If an entity of a declarative application uses content type `text/ecmascript` and invokes this method in such a manner as to generate content that is not valid HTML Transitional or HTML Frameset content, then the user agent may abort the application.

When a user agent decodes content inserted by this method, it shall do so by first transcoding this content from `text/html` to `application/xhtml+xml`. The resulting modified document entity shall be valid `application/xhtml+xml` content. See Section 5.1.1.2 for further information.

> *Note:* It is possible to simply transcode a valid, well-formed `text/html` document entity fragment to `application/xhtml+xml` by performing the process described in ANNEX F. When performing this transcoding, the source fragment is interpreted either as an HTML Transitional or HTML Frameset fragment according to the elements referenced by the fragment.

This method shall not be used to insert `application/xhtml+xml` content into the current entity's input stream.

> *Note:* The functionality of generating `application/xhtml+xml` content may be achieved by other means; for example, by using `Node::appendChild`.

5.3.1.2.3.3.12 `HTMLDocument::writeln`

This method shall have the same semantics as `HTMLDocument::write` except that a newline character shall be appended to the inserted content.

5.3.1.2.3.3.13 `HTMLDocumentExt::alinkColor`

This mutable, string valued property shall have the same value as the `HTMLBodyElement::aLink` property. Mutation of this property shall be reflected in any subsequent reference to the value of `HTMLBodyElement::aLink`.

Use of this property is deprecated. The `HTMLBodyElement::aLink` property should be used instead.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.3.14 `HTMLDocumentExt::bgColor`

This mutable, string valued property shall have the same value as the `HTMLBodyElement::background` property. Mutation of this property shall be reflected in any subsequent reference to the value of `HTMLBodyElement::background`.

Use of this property is deprecated. The `HTMLBodyElement::background` property should be used instead.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.3.15 `HTMLDocumentExt::clear`

This void valued method shall have the same semantics as the `HTMLDocument::open` method.

Use of this property is deprecated. The `HTMLDocument::open` method should be used instead.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.3.16 `HTMLDocumentExt::fgColor`

This mutable, string valued property shall have the same value as the `HTMLBodyElement::text` property. Mutation of this property shall be reflected in any subsequent reference to the value of `HTMLBodyElement::text`.

Use of this property is deprecated. The `HTMLBodyElement::text` property should be used instead.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.3.17 `HTMLDocumentExt::lastModified`

This immutable, string valued property denotes the last modified date and time of the underlying document. If no last modified information is available, then the value of this property shall be an empty string. If last modified information is available by means of application signaling metadata, then the value of this property shall be a primitive String value which adheres to the *rfc1123-date* non-terminal of [HTTP], Section 3.3.1.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.3.18 `HTMLDocumentExt::linkColor`

This mutable, string valued property shall have the same value as the `HTMLBodyElement::link` property. Mutation of this property shall be reflected in any subsequent reference to the value of `HTMLBodyElement::link`.

Use of this property is deprecated. The `HTMLBodyElement::link` property should be used instead.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.3.19 `HTMLDocumentExt::location`

This immutable, string valued property shall have the same value as the `HTMLDocument::URL` property.

Use of this property is deprecated. The `HTMLDocument::URL` property should be used instead.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.3.20 `HTMLDocumentExt::vlinkColor`

This mutable, string valued property shall have the same value as the `HTMLBodyElement::vLink` property. Mutation of this property shall be reflected in any subsequent reference to the value of `HTMLBodyElement::vLink`.

Use of this property is deprecated. The `HTMLBodyElement::vLink` property should be used instead.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.3.21 `HTMLDocumentExt::window`

This immutable, object value property shall reference the `Window` object within which the object which implements the `HTMLDocument` interface is being or is to be displayed. See Section 5.3.1.2.9.4 for information about the `Window` object.

> *Note:* If a single document instance may be associated with multiple `Window` instances, then the choice of which `Window` instance this property references is implementation dependent.

> *Note:* No explicit binding is specified for this property for ECMAScript since the *Global* object is bound to an object implementing the `Window` interface which provides a *window* property. See Section 5.3.1.2.9.4.21 for further information. This property is specified in order to support access to a `Window` instance from a `Document` instance in those language bindings employed with procedural application content that do not support a *Global* object.

5.3.1.2.3.4 HTMLFormElement Object

In addition to implementing the `HTMLFormElement` interface as defined by [DOM2-HTML], each instance of an `HTMLFormElement` object shall implement the following interface:

```
interface HTMLFormElementExt
{
  /* read-write properties */
  attribute DOMString encoding;
};
```

The following subsections describe constraints on `HTMLFormElement` interface semantics as well as the entirety of `HTMLFormElementExt` interface semantics.

5.3.1.2.3.4.1 `HTMLFormElement::elements`

This immutable, object value property shall reference an immutable `HTMLCollection` object that, in turn, references zero or more mutable `HTMLInputElement`, `HTMLSelectElement`, or `HTMLTextAreaElement` objects.

An `HTMLInputElement`, `HTMLSelectElement`, or `HTMLTextAreaElement` shall be instantiated and added to the `HTMLCollection` object referenced by this property for each *input*, *select*, and *textarea* element decoded.

> *Note:* Although script content may not mutate the referenced `HTMLCollection` object, the user agent will mutate this object as a side effect of loading the document and decoding individual *input*, *select*, and *textarea* elements. This mutation of the referenced `HTMLCollection` object by the user agent will occur asynchronously with respect to script content evaluation.

For each named `HTMLInputElement`, `HTMLSelectElement`, and `HTMLTextAreaElement` object the user agent adds to the referenced `HTMLCollection` object, a new property shall be added to the current `HTMLFormElement` object, where the name of the property is the value of the *name* property of that `HTMLInputElement`, `HTMLSelectElement`, or `HTMLTextAreaElement` object and the value of the property is the `HTMLInputElement`, `HTMLSelectElement`, or `HTMLTextAreaElement` object itself. If an identically named property already exists on the current `HTMLFormElement` object, then a property shall not be added and the value of the pre-existing, identically named property shall not be changed.

5.3.1.2.3.4.2 `HTMLFormElementExt::encoding`

This mutable, string valued property shall have the same value as the `HTMLFormElement::enctype` property. Mutation of this property shall be reflected in any subsequent reference to the value of `HTMLFormElement::enctype`.

Use of this property is deprecated. The `HTMLFormElement::enctype` property should be used instead.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.5 HTMLImageElement Object

In addition to implementing the `HTMLImageElement` interface as defined by [DOM2-HTML], each instance of an `HTMLImageElement` object shall implement the following interface:

```
interface HTMLImageElementExt
{
  /* read-only properties */
  readonly attribute boolean complete;
  /* read-write properties */
  attribute DOMString lowsrc;
};
```

Note: This interface is provided to obtain lexical interoperability with certain legacy content. Its use is deprecated for non-legacy applications.

5.3.1.2.3.5.1 `HTMLImageElementExt::complete`

This immutable, boolean valued property shall be `false` during the time the image is loading and shall transition to `true` when the image is loaded or when a load error occurs.

Note: This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.5.2 `HTMLImageElementExt::lowsrc`

This mutable, string valued property shall be either (1) an empty string or (2) a URI which denotes a low resolution, alternate image that may be displayed as alternate content.

Use of this property is deprecated.

Note: This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.6 HTMLInputElement Object

The following subsections describe constraints on `HTMLInputElement` interface semantics.

5.3.1.2.3.6.1 `HTMLInputElement::value`

If an `HTMLInputElement` is instantiated as a result of decoding an *input* element in the source document and the input element has a *type* attribute with the value `password`, then read access to this mutable, string valued property shall return a nonce value consisting of *N* asterisk `'*'` characters, where *N* is the length of the actual string value.

Note: This constraint provides additional security with respect to script content access to user entered passwords.

5.3.1.2.3.7 HTMLObjectElement Object

In addition to implementing the `HTMLObjectElement` interface as defined by [DOM2-HTML], each instance of an `HTMLObjectElement` object shall implement the following interface:

```
interface HTMLObjectElementExt
{
  /* read-only properties */
  readonly attribute boolean complete;
  /* read-write properties */
  attribute DOMString lowsrc;
  attribute DOMString src;
};
```

Note: This interface is provided to obtain lexical interoperability with certain legacy content. Its use is deprecated for non-legacy applications.

In addition, instances of an `HTMLObjectElement` object which are instantiated as a result of decoding a trigger object element shall implement the following interface:

```
interface HTMLTriggerObjectElementExt
{
  /* read-only properties */
  readonly attribute DOMString backChannel;
  readonly attribute DOMString contentLevel;
  readonly attribute DOMString sourceId;
  /* read-write properties */
  attribute boolean enabled;
  attribute boolean releasable;
};
```

No more than one instance of an `HTMLElement` object shall implement this interface within the context of all of the active document object instance hierarchies associated with a single declarative application.

> *Note:* This interface is provided to obtain lexical interoperability with certain legacy content. Its use is deprecated for non-legacy applications.
>
> *Note:* See Section 5.1.1.6.8.2 for more information on use of a trigger object element.

The following subsections describe constraints on `HTMLObjectElement` interface semantics as well as the entirety of `HTMLObjectElementExt` and `HTMLTriggerObjectElementExt` interface semantics.

5.3.1.2.3.7.1 `HTMLObjectElementExt::complete`

This immutable, boolean valued property shall be `false` during the time the object is loading and shall transition to `true` when the object is loaded or when a load error occurs.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.7.2 `HTMLObjectElementExt::lowsrc`

This mutable, string valued property shall be either (1) an empty string or (2) a URI which denotes a low resolution, alternate image that may be displayed as alternate content.

Use of this property is deprecated.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.3.7.3 `HTMLObjectElementExt::src`

This mutable, string valued property shall have the same value as the `data` property of this object. Mutation of this property shall be reflected in any subsequent reference to the value of `data` property.

Use of this property is deprecated.

5.3.1.2.3.7.4 `HTMLTriggerObjectElementExt::backChannel`

This immutable, string valued property shall have the same value as the `Navigator::ddeBackChannel` property.

5.3.1.2.3.7.5 `HTMLTriggerObjectElementExt::contentLevel`

This immutable, string valued property shall have the same value as the `Navigator::ddeContentLevel` property.

5.3.1.2.3.7.6 `HTMLTriggerObjectElementExt::sourceId`

This immutable, string valued property shall have the same value as the `Navigator::ddeSourceId` property.

5.3.1.2.3.7.7 `HTMLTriggerObjectElementExt::enabled`

This mutable, boolean valued property shall have the same value as the `Navigator::ddeEnabled` property. Mutation of this property shall be reflected in the value of as the `Navigator::ddeEnabled` property.

5.3.1.2.3.7.8 `HTMLTriggerObjectElementExt::releasable`

This mutable, boolean valued property shall have the same value as the `Navigator::ddeReleasable` property. Mutation of this property shall be reflected in the value of as the `Navigator::ddeReleasable` property.

5.3.1.2.4 Views Module Objects

A script content entity or fragment may reference and a user agent which supports this facility shall implement the following host objects, defined by [DOM2-VIEWS], Section 1.2, *Interfaces*, in conjunction with [DOM2-VIEWS], Appendix C, *ECMAScript Language Binding*.

None of the following objects is directly instantiable; that is, the internal `[[Construct]]` property for the object shall remain undefined. Any attempt to construct an object using the `new` operator shall cause an ECMAScript *TypeError* to be raised.

> *Note:* See [ECMASCRIPT], Section 11.2.2, for further information on use of the internal `[[Construct]]` property.

Table 11 DOM Views Interface Support

`AbstractView`	`DocumentView`	`DocumentViewExt`

5.3.1.2.4.1 DocumentView Object

In addition to implementing the `DocumentView` interface as defined by [DOM2-VIEWS], Section 1.2, *Interfaces*, each instance of a `DocumentView` object shall implement the following interface:

```
interface DocumentViewExt
{
  /* read-only properties */
  readonly attribute double refWidthMm;
  readonly attribute double refHeightMm;
  readonly attribute double gfxPosX;
  readonly attribute double gfxPosY;
  readonly attribute double gfxWidth;
  readonly attribute double gfxHeight;
  readonly attribute unsigned long gfxWidthPx;
  readonly attribute unsigned long gfxHeightPx;
  readonly attribute double vidPosX;
  readonly attribute double vidPosY;
  readonly attribute double vidWidth;
  readonly attribute double vidHeight;
  readonly attribute unsigned long vidWidthPx;
  readonly attribute unsigned long vidHeightPx;
  readonly attribute unsigned long sampleBitsR;
  readonly attribute unsigned long sampleBitsG;
  readonly attribute unsigned long sampleBitsB;
  readonly attribute unsigned long sampleBitsA;
  /* read-write properties */
  attribute boolean refreshOnChange;
  /* methods */
  boolean refresh(in DOMString id);
};
```

The following subsections describe the entirety of `DocumentViewExt` interface semantics.

In order to fully specify the relationship between the graphics and video planes in which a document's view is rendered, a normalized (reference) coordinate space is used, as depicted below in Figure 1 Display Reference Model. Coordinates in this space are specified using real valued numbers, where the top-left point of the reference space is designated as position (0,0) and the bottom-right point designated as position (1,1).

> *Note:* This reference space will generally correspond to the full display screen; however, the graphics and video planes need not align with each other or the reference space. Furthermore, the pixels of the graphics and video plane need not be of identical size or aspect ratio.

5.3.1.2.4.1.1 `DocumentViewExt::gfxHeight`

This immutable, real number valued property shall have a value which specifies the height of the graphics plane in normalized coordinates.

The value of this property shall be determined by the user agent implementer.

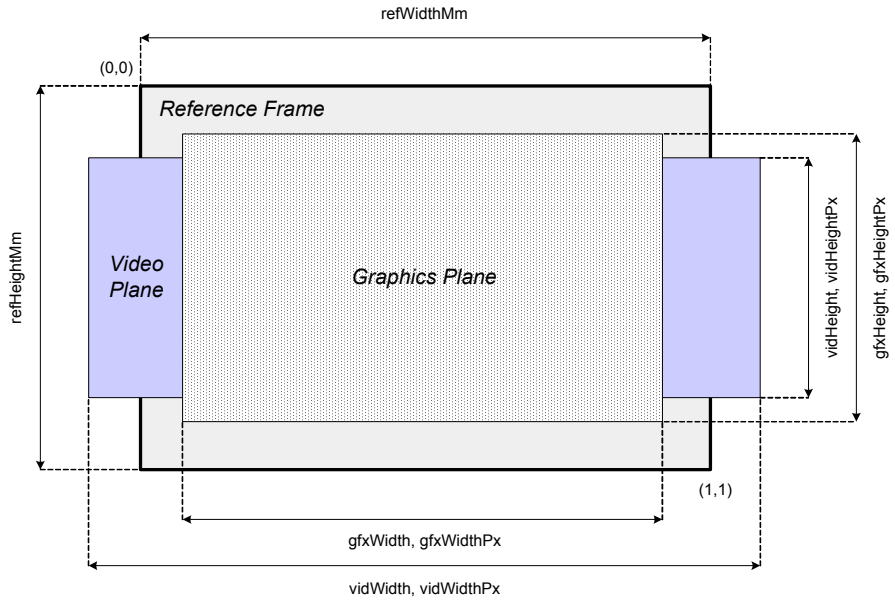

Figure 1 Display Reference Model

5.3.1.2.4.1.2 `DocumentViewExt::gfxHeightPx`

This immutable, non-negative integral number valued property shall have a value which specifies the height of the graphics plane in pixels.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.3 `DocumentViewExt::gfxPosX`

This immutable, real number valued property shall have a value which specifies the leftmost position of the graphics plane in normalized coordinates.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.4 `DocumentViewExt::gfxPosY`

This immutable, real number valued property shall have a value which specifies the topmost position of the video plane in normalized coordinates.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.5 `DocumentViewExt::gfxWidth`

This immutable, real number valued property shall have a value which specifies the width of the graphics plane in normalized coordinates.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.6 `DocumentViewExt::gfxWidthPx`

This immutable, non-negative integral number valued property shall have a value which specifies the width of the graphics plane in pixels.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.7 `DocumentViewExt::refHeightMm`

This immutable, real valued property shall have a value which specifies the height of normalized coordinate space in millimeters as realized on the physical display device; that is, the absolute distance between points (0,0) and (0,1).

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.8 `DocumentViewExt::refWidthMm`

This immutable, real valued property shall have a value which specifies the width of normalized coordinate space in millimeters as realized on the physical display device; that is, the absolute distance between points (0,0) and (1,0).

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.9 `DocumentViewExt::refreshOnChange`

This mutable, boolean valued property shall have a value which specifies whether any change (mutation) of the document instance should cause refresh (i.e., reformat and redraw) to occur. If set to the value `false`, then refresh should not occur when the document instance is changed. If set to `true`, then any change of the document instance should cause a refresh. Additionally, if the value of this property is changed from `false` to `true` and any change has occurred since the value of the property was set to `false`, then a refresh should occur.

The default value of this property shall be `true` unless the application was signaled with an application parameter `deferDisplay`, in which case the default value shall be `false`.

> Note: See [DASE], Section 6.1.1.6.13.3, for more information on how this application parameter is specified.

5.3.1.2.4.1.10 `DocumentViewExt::refresh`

This boolean valued method shall cause either the entire view or a portion of the view to be refreshed while taking into account changes in one or more rendered elements. The *id* argument to this method shall specify either (1) *null* or an empty string or (2) a non-empty string whose value references some element in accordance to the element's *id* attribute. If the *id* argument is *null* or an empty string, then refresh scope applies to the entire document rendered in the view; otherwise, refresh scope applies to the identified element and any of its child elements, if present.

If no element or renderable content referenced by an element in the refresh scope has changed since the prior (i.e., currently depicted) rendition, then no side-effect shall obtain, and the boolean value `false` shall be returned; otherwise, the affected portion(s) of the view shall be re-rendered using the changed element or referenced renderable content, and the value `true` shall be returned.

If a document is refreshed by means of this method, then any dynamic script object state associated with the document shall be discarded. In addition, the values of any references to the former document object from the properties `Window::document` and `Window::frames` shall be replaced by references to a new document object. Subsequent references to the former document object by user-defined script objects are not defined.

> Note: This method is intended to satisfy the need for programmatic control of element rendition in the presence of dynamic updates to underlying image, object, and document content, where such updates may occur as a result of asynchronous changes to content.
>
> Note: User defined script objects must take adequate care that references to document instances which are undergoing dynamic updates are replaced by references to the updated document instances.

5.3.1.2.4.1.11 `DocumentViewExt::sampleBitsA`

This immutable, non-negative integral number valued property shall have a value which specifies the bits per pixel of each view pixel's alpha sample.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.12 `DocumentViewExt::sampleBitsB`

This immutable, non-negative integral number valued property shall have a value which specifies the bits per pixel of each view pixel's blue sample.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.13 `DocumentViewExt::sampleBitsG`

This immutable, non-negative integral number valued property shall have a value which specifies the bits per pixel of each view pixel's green sample.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.14 `DocumentViewExt::sampleBitsR`

This immutable, non-negative integral number valued property shall have a value which specifies the bits per pixel of each view pixel's red sample.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.15 `DocumentViewExt::vidHeight`

This immutable, real number valued property shall have a value which specifies the height of the video plane in normalized coordinates.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.16 `DocumentViewExt::vidHeightPx`

This immutable, non-negative integral number valued property shall have a value which specifies the height of the video plane in pixels.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.17 `DocumentViewExt::vidPosX`

This immutable, real number valued property shall have a value which specifies the leftmost position of the video plane in normalized coordinates.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.18 `DocumentViewExt::vidPosY`

This immutable, real number valued property shall have a value which specifies the topmost position of the video plane in normalized coordinates.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.19 `DocumentViewExt::vidWidth`

This immutable, real number valued property shall have a value which specifies the width of the video plane in normalized coordinates.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.4.1.20 `DocumentViewExt::vidWidthPx`

This immutable, non-negative integral number valued property shall have a value which specifies the width of the video plane in pixels.

The value of this property shall be determined by the user agent implementer.

5.3.1.2.5 StyleSheets Module Objects

A script content entity or fragment may reference and a user agent which supports this facility shall implement the following host objects, defined by [DOM2-STYLE], Section 1.2, *Style Sheet Interfaces*, and Section 1.3, *Document Extensions*, in conjunction with [DOM2-STYLE], Appendix C, *ECMAScript Language Binding*.

None of the following objects is directly instantiable; that is, the internal `[[Construct]]` property for the object shall remain undefined. Any attempt to construct an object using the `new` operator shall cause an ECMAScript *TypeError* to be raised.

> *Note:* See [ECMASCRIPT], Section 11.2.2, for further information on use of the internal `[[Construct]]` property.

Table 12 DOM Stylesheet Interface Support

DocumentStyle	LinkStyle	MediaList
StyleSheet	StyleSheetList	

5.3.1.2.6 CSS Module Objects

A script content entity or fragment may reference and a user agent which supports this facility shall implement the following host objects, defined by [DOM2-STYLE], Section 2.2, *CSS Fundamental Interfaces*, in conjunction with [DOM2-STYLE], Appendix C, *ECMAScript Language Binding*.

None of the following objects is directly instantiable; that is, the internal `[[Construct]]` property for the object shall remain undefined. Any attempt to construct an object using the `new` operator shall cause an ECMAScript *TypeError* to be raised.

> *Note:* See [ECMASCRIPT], Section 11.2.2, for further information on use of the internal `[[Construct]]` property.

Table 13 DOM Cascading Stylesheet Interface Support

Counter	CSSCharsetRule	CSSFontFaceRule
CSSImportRule	CSSMediaRule	CSSPrimitiveValue
CSSRule	CSSRuleList	CSSStyleDeclaration
CSSStyleRule	CSSStyleSheet	CSSUnknownRule
CSSValue	CSSValueList	DocumentCSS
DOMImplementationCSS	ElementCSSInlineStyle	Rect
RGBColor	ViewCSS	

5.3.1.2.7 Events Module Objects

A script content entity or fragment may reference and a user agent which supports this facility shall implement the following host objects, defined by [DOM2-EVENTS], Section 1.3 through 1.5, in conjunction with [DOM2-EVENTS], Appendix C, *ECMAScript Language Binding*.

None of the following objects is directly instantiable; that is, the internal `[[Construct]]` property for the object shall remain undefined. Any attempt to construct an object using the `new` operator shall cause an ECMAScript *TypeError* to be raised.

> *Note:* See [ECMASCRIPT], Section 11.2.2, for further information on use of the internal `[[Construct]]` property.

Table 14 DOM Event Interface Support

DocumentEvent	Event	EventException
EventListener	EventTarget	

5.3.1.2.8 Event Set Module Objects

A script content entity or fragment may reference and a user agent which supports this facility shall implement the following host objects, defined herein or defined by [DOM2-EVENTS], Section 1.6, *Event Module Definitions*, in conjunction with [DOM2-EVENTS], Appendix C, *ECMAScript Language Binding*.

None of the following objects is directly instantiable; that is, the internal `[[Construct]]` property for the object shall remain undefined. Any attempt to construct an object using the `new` operator shall cause an ECMAScript *TypeError* to be raised.

> *Note:* See [ECMASCRIPT], Section 11.2.2, for further information on use of the internal `[[Construct]]` property.

Table 15 DOM Event Set Interface Support

KeyEvent	KeyModifiers	MouseEvent
MutationEvent	UIEvent	VirtualKeys

5.3.1.2.8.1 KeyEvent Object

A `KeyEvent` object provides a means to deliver events generated by a text or key input device. A `KeyEvent` object shall implement the `KeyEvent`, `KeyModifiers`, and `VirtualKeys` interfaces specified below.

```
interface KeyEvent : UIEvent
{
  /* read-write properties */
  attribute DOMString outputString;
  attribute unsigned long keyVal;
  attribute unsigned long virtKeyVal;
  attribute boolean visibleOutputGenerated;
  attribute boolean numPad;
  /* methods */
  void initKeyEvent(
       in DOMString typeArg,
       in boolean canBubbleArg,
       in boolean cancelableArg,
       in views::AbstractView viewArg,
       in long detailArg,
       in DOMString outputString,
       in unsigned long keyValArg,
       in unsigned long virtKeyValArg,
       in boolean visibleOutputGeneratedArg,
       in boolean numPadArg);
  void initModifier(in unsigned long modifier, in boolean value);
  boolean checkModifier(in unsigned long modifier);
};

interface KeyModifiers
{
  const unsigned long MOD_NONE                      = 0;
  const unsigned long MOD_SHIFT                     = 1;
  const unsigned long MOD_CONTROL                   = 2;
  const unsigned long MOD_META                      = 4;
  const unsigned long MOD_ALT                       = 8;
};

interface VirtualKeys
{
  const unsigned long VK_UNDEFINED                  = 0;
  const unsigned long VK_CANCEL                     = 3;
  const unsigned long VK_BACK_SPACE                 = 8;
  const unsigned long VK_TAB                        = 9;
  const unsigned long VK_ENTER                      = 10;
  const unsigned long VK_CLEAR                      = 12;
  const unsigned long VK_SHIFT                      = 16;
  const unsigned long VK_CONTROL                    = 17;
  const unsigned long VK_ALT                        = 18;
  const unsigned long VK_PAUSE                      = 19;
```

```
const unsigned long VK_CAPS_LOCK                    = 20;
const unsigned long VK_KANA                         = 21;
const unsigned long VK_FINAL                        = 24;
const unsigned long VK_KANJI                        = 25;
const unsigned long VK_ESCAPE                       = 27;
const unsigned long VK_CONVERT                      = 28;
const unsigned long VK_NONCONVERT                   = 29;
const unsigned long VK_ACCEPT                       = 30;
const unsigned long VK_MODECHANGE                   = 31;
const unsigned long VK_SPACE                        = 32;
const unsigned long VK_PAGE_UP                      = 33;
const unsigned long VK_PAGE_DOWN                    = 34;
const unsigned long VK_END                          = 35;
const unsigned long VK_HOME                         = 36;
const unsigned long VK_LEFT                         = 37;
const unsigned long VK_UP                           = 38;
const unsigned long VK_RIGHT                        = 39;
const unsigned long VK_DOWN                         = 40;
const unsigned long VK_COMMA                        = 44;
const unsigned long VK_PERIOD                       = 46;
const unsigned long VK_SLASH                        = 47;
const unsigned long VK_0                            = 48;
const unsigned long VK_1                            = 49;
const unsigned long VK_2                            = 50;
const unsigned long VK_3                            = 51;
const unsigned long VK_4                            = 52;
const unsigned long VK_5                            = 53;
const unsigned long VK_6                            = 54;
const unsigned long VK_7                            = 55;
const unsigned long VK_8                            = 56;
const unsigned long VK_9                            = 57;
const unsigned long VK_SEMICOLON                    = 59;
const unsigned long VK_EQUALS                       = 61;
const unsigned long VK_A                            = 65;
const unsigned long VK_B                            = 66;
const unsigned long VK_C                            = 67;
const unsigned long VK_D                            = 68;
const unsigned long VK_E                            = 69;
const unsigned long VK_F                            = 70;
const unsigned long VK_G                            = 71;
const unsigned long VK_H                            = 72;
const unsigned long VK_I                            = 73;
const unsigned long VK_J                            = 74;
const unsigned long VK_K                            = 75;
const unsigned long VK_L                            = 76;
const unsigned long VK_M                            = 77;
const unsigned long VK_N                            = 78;
const unsigned long VK_O                            = 79;
const unsigned long VK_P                            = 80;
const unsigned long VK_Q                            = 81;
const unsigned long VK_R                            = 82;
const unsigned long VK_S                            = 83;
const unsigned long VK_T                            = 84;
const unsigned long VK_U                            = 85;
const unsigned long VK_V                            = 86;
const unsigned long VK_W                            = 87;
const unsigned long VK_X                            = 88;
const unsigned long VK_Y                            = 89;
const unsigned long VK_Z                            = 90;
const unsigned long VK_OPEN_BRACKET                 = 91;
const unsigned long VK_BACK_SLASH                   = 92;
const unsigned long VK_CLOSE_BRACKET                = 93;
const unsigned long VK_NUMPAD0                      = 96;
```

```
const unsigned long VK_NUMPAD1                    = 97;
const unsigned long VK_NUMPAD2                    = 98;
const unsigned long VK_NUMPAD3                    = 99;
const unsigned long VK_NUMPAD4                    = 100;
const unsigned long VK_NUMPAD5                    = 101;
const unsigned long VK_NUMPAD6                    = 102;
const unsigned long VK_NUMPAD7                    = 103;
const unsigned long VK_NUMPAD8                    = 104;
const unsigned long VK_NUMPAD9                    = 105;
const unsigned long VK_MULTIPLY                   = 106;
const unsigned long VK_ADD                        = 107;
const unsigned long VK_SEPARATER                  = 108;
const unsigned long VK_SUBTRACT                   = 109;
const unsigned long VK_DECIMAL                    = 110;
const unsigned long VK_DIVIDE                     = 111;
const unsigned long VK_F1                         = 112;
const unsigned long VK_F2                         = 113;
const unsigned long VK_F3                         = 114;
const unsigned long VK_F4                         = 115;
const unsigned long VK_F5                         = 116;
const unsigned long VK_F6                         = 117;
const unsigned long VK_F7                         = 118;
const unsigned long VK_F8                         = 119;
const unsigned long VK_F9                         = 120;
const unsigned long VK_F10                        = 121;
const unsigned long VK_F11                        = 122;
const unsigned long VK_F12                        = 123;
const unsigned long VK_DELETE                     = 127;
const unsigned long VK_NUM_LOCK                   = 144;
const unsigned long VK_SCROLL_LOCK                = 145;
const unsigned long VK_PRINTSCREEN                = 154;
const unsigned long VK_INSERT                     = 155;
const unsigned long VK_HELP                       = 156;
const unsigned long VK_META                       = 157;
const unsigned long VK_BACK_QUOTE                 = 192;
const unsigned long VK_QUOTE                      = 222;
const unsigned long VK_COLORED_KEY_0              = 403;
const unsigned long VK_COLORED_KEY_1              = 404;
const unsigned long VK_COLORED_KEY_2              = 405;
const unsigned long VK_COLORED_KEY_3              = 406;
const unsigned long VK_COLORED_KEY_4              = 407;
const unsigned long VK_COLORED_KEY_5              = 408;
const unsigned long VK_POWER                      = 409;
const unsigned long VK_DIMMER                     = 410;
const unsigned long VK_WINK                       = 411;
const unsigned long VK_REWIND                     = 412;
const unsigned long VK_STOP                       = 413;
const unsigned long VK_EJECT_TOGGLE               = 414;
const unsigned long VK_PLAY                       = 415;
const unsigned long VK_RECORD                     = 416;
const unsigned long VK_FAST_FWD                   = 417;
const unsigned long VK_PLAY_SPEED_UP              = 418;
const unsigned long VK_PLAY_SPEED_DOWN            = 419;
const unsigned long VK_PLAY_SPEED_RESET           = 420;
const unsigned long VK_RECORD_SPEED_NEXT          = 421;
const unsigned long VK_GO_TO_START                = 422;
const unsigned long VK_GO_TO_END                  = 423;
const unsigned long VK_TRACK_PREV                 = 424;
const unsigned long VK_TRACK_NEXT                 = 425;
const unsigned long VK_RANDOM_TOGGLE              = 426;
const unsigned long VK_CHANNEL_UP                 = 427;
const unsigned long VK_CHANNEL_DOWN               = 428;
const unsigned long VK_STORE_FAVORITE_0           = 429;
```

```
  const unsigned long VK_STORE_FAVORITE_1            = 430;
  const unsigned long VK_STORE_FAVORITE_2            = 431;
  const unsigned long VK_STORE_FAVORITE_3            = 432;
  const unsigned long VK_RECALL_FAVORITE_0           = 433;
  const unsigned long VK_RECALL_FAVORITE_1           = 434;
  const unsigned long VK_RECALL_FAVORITE_2           = 435;
  const unsigned long VK_RECALL_FAVORITE_3           = 436;
  const unsigned long VK_CLEAR_FAVORITE_0            = 437;
  const unsigned long VK_CLEAR_FAVORITE_1            = 438;
  const unsigned long VK_CLEAR_FAVORITE_2            = 439;
  const unsigned long VK_CLEAR_FAVORITE_3            = 440;
  const unsigned long VK_SCAN_CHANNELS_TOGGLE        = 441;
  const unsigned long VK_PINP_TOGGLE                 = 442;
  const unsigned long VK_SPLIT_SCREEN_TOGGLE         = 443;
  const unsigned long VK_DISPLAY_SWAP                = 444;
  const unsigned long VK_SCREEN_MODE_NEXT            = 445;
  const unsigned long VK_VIDEO_MODE_NEXT             = 446;
  const unsigned long VK_VOLUME_UP                   = 447;
  const unsigned long VK_VOLUME_DOWN                 = 448;
  const unsigned long VK_MUTE                        = 449;
  const unsigned long VK_SURROUND_MODE_NEXT          = 450;
  const unsigned long VK_BALANCE_RIGHT               = 451;
  const unsigned long VK_BALANCE_LEFT                = 452;
  const unsigned long VK_FADER_FRONT                 = 453;
  const unsigned long VK_FADER_REAR                  = 454;
  const unsigned long VK_BASS_BOOST_UP               = 455;
  const unsigned long VK_BASS_BOOST_DOWN             = 456;
  const unsigned long VK_INFO                        = 457;
  const unsigned long VK_GUIDE                       = 458;
  const unsigned long VK_TELETEXT                    = 459;
  const unsigned long VK_SUBTITLE                    = 460;
};
```

5.3.1.2.8.1.1 `KeyEvent::outputString`

This mutable, string valued property shall have a value denoting the text output generated by the key event. The value may be a single Unicode character or a string of multiple characters. If no output is generated by the event, then this property shall have a `null` value.

5.3.1.2.8.1.2 `KeyEvent::keyVal`

This mutable, integral valued property shall have a value denoting the Unicode character associated with the depressed key. If the key has no Unicode representation, then the value shall be zero (0).

5.3.1.2.8.1.3 `KeyEvent::virtKeyVal`

When the key associated with a key event is not representable with a Unicode character, this mutable, integral valued property shall denote a virtual key code associated with the depressed key. If no Unicode character is available and no virtual key code is associated with the key, then the value shall be zero (0).

In the case that a key is associated with both a Unicode character and a virtual key, then this property shall denote the virtual key.

The set of values permitted for this property shall be restricted to those constants defined by the `VirtualKeys` interface above.

> *Note:* The virtual key codes defined by the `VirtualKeys` interface are explicitly chosen to match the virtual key codes employed by DASE-1 Part 3 active object content facilities. In particular, these values correspond to those defined for use with the `java.awt.event.KeyEvent` and `org.havi.ui.event.HRcEvent` types.

5.3.1.2.8.1.4 `KeyEvent::visibleOutputGenerated`

This mutable, boolean valued property indicates whether the key event will normally cause visible output. If the key event does not generate any visible output, such as the use of a function key or the combination of certain modifier keys used in conjunction with another key, then the value will be `false`. If visible output is normally generated by the key event then the value will be `true`.

The value of `visibleOutputGenerated` does not guarantee the creation of a character. If a key event causing visible output is cancelable it may be prevented from causing visible output. This attribute is intended primarily to differentiate between keys events which may or may not produce visible output depending on the system state.

5.3.1.2.8.1.5 `KeyEvent::numPad`

This mutable, boolean valued property indicates whether or not the key event was generated on the number pad section of the keyboard. If the number pad was used to generate the key event the value is `true`, otherwise the value is `false`.

5.3.1.2.8.1.6 `KeyEvent::initKeyEvent`

The `initKeyEvent` method is used to initialize the value of a `KeyEvent` created through the `DocumentEvent` interface. This method may only be called before the `KeyEvent` has been dispatched via the `dispatchEvent` method, though it may be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence. This method has no effect if called after the event has been dispatched.

5.3.1.2.8.1.7 `KeyEvent::initModifier`

The `initModifier` method is used to initialize the values of any modifiers associated with a `KeyEvent` created through the `DocumentEvent` interface. This method may only be called before the `KeyEvent` has been dispatched via the `dispatchEvent` method, though it may be called multiple times during that phase if necessary. If called multiple times with the same *modifier* property the final invocation takes precedence. Unless explicitly give a value of `true`, all modifiers have a value of `false`. This method has no effect if called after the event has been dispatched. The list of values below, a subset of those defined by the `KeyModifiers` interface, represents the allowable *modifier* parameters for this method:

- `MOD_NONE`
- `MOD_SHIFT`
- `MOD_CONTROL`
- `MOD_META`
- `MOD_ALT`

Note: The modifier codes defined by the `KeyModifierss` interface are explicitly chosen to match the modifier codes employed by DASE-1 Part 3 active object content facilities. In particular, these values correspond to those defined for use with the `java.awt.event.InputEvent` type.

5.3.1.2.8.1.8 `KeyEvent::checkModifier`

The `checkModifier` method is used to check the status of a single modifier key associated with a `KeyEvent`. The identifier of the modifier in question is passed into the `checkModifier` method. If the modifier is triggered it will return `true`. If not, it will return `false`. The value of the *modifier* parameter to this method is limited to those values taken by the *modifier* parameter to the `initModifier` method.

5.3.1.2.8.2 Key Event Types

The value of `Event::type` for a `KeyEvent` object shall be one of the event types described in the following sub-sections.

5.3.1.2.8.2.1 org.atsc.textinput

The `org.atsc.textinput` event indicates that text information has been entered. The text information entered can originate from a variety of sources. It could, for example, be a character resulting from a key press. It could also be a string resulting from an input method. The `detail` attribute inherited from `UIEvent` is used to indicated the number of key presses which have occurred during key repetition. If this information is not available, this value should be zero (0).

Name: `org.atsc.textinput`
Bubbles: yes
Cancelable: yes
Context: view, detail, visibleOutputGenerated, outputString, keyVal, virtKeyVal, numPad

5.3.1.2.8.2.2 org.atsc.keydown

The `keydown` event indicates a physical or logical key motion down (engage).

Name: `org.atsc.keydown`
Bubbles: yes
Cancelable: yes
Context: view, keyVal, virtKeyVal, numPad

5.3.1.2.8.2.3 org.atsc.keyup

The `keyup` event indicates a physical or logical key motion up (dis-engage).

Name: `org.atsc.keyup`
Bubbles: yes
Cancelable: yes
Context: view, keyVal, virtKeyVal, numPad

5.3.1.2.8.3 Mutation Event Types

In addition to those mutation event types defined by [DOM2-EVENTS], Section 1.6.4, the following mutation event shall be generated by a user agent.

5.3.1.2.8.3.1 org.atsc.resourcechanged

The `org.atsc.resourcechanged` event indicates that an application resource referenced explicitly or implicitly by an element has changed. If the document application resource changes, then the *relatedNode* shall reference the `Document` node whose resource has changed. If an image or object data resource changes, then the *relatedNode* shall reference the associated `Element` node. Changes to application resources which do not represent a document, image, or object need not cause generation of this event.

Name: `org.atsc.resourcechanged`
Bubbles: no
Cancelable: no
Context: relatedNode

5.3.1.2.9 Environment Module Objects

A script content entity or fragment may reference and a user agent which supports this facility shall implement the following host objects, defined in the following sections.

None of the following objects is directly instantiable; that is, the internal `[[Construct]]` property for the object shall remain undefined. Any attempt to construct an object using the `new` operator shall cause an ECMAScript *TypeError* to be raised.

> *Note:* See [ECMASCRIPT], Section 11.2.2, for further information on use of the internal `[[Construct]]` property.

Table 16 DOM Environment Interface Support

`History`	`Location`	`Navigator`
`Window`		

The presence of support for this set of interfaces is indicated by means of a feature string `"org.atsc.dom.environment"`, which may be used with the `DOMImplementation::hasFeature` method.

5.3.1.2.9.1 History Object

A `History` object provides a means to navigate among recently navigated content. It implements the following interface:

```
interface History
{
  /* read-only properties */
  readonly attribute long length;
  /* methods */
  void back();
  void forward();
  void go(in long index);
  void go(in DOMString uri);
};
```

A user agent which implements this facility shall support a global navigation history list length of at least four entries. Mutation of this list is effected by the user agent during the process of content navigation, whether navigation is initiated by the end-user or by script content.

Script content shall not rely upon the user agent retaining more than four global navigation history entries.

A user agent may cache the decoded representation of documents referenced by non-current entries in the global navigation history list; alternatively, it may retain only the resource reference (i.e., the URI) of the document, in which case navigation to an entry will require reloading the document.

Script content shall not rely upon the user agent to retain the decoded representation of document referenced by non-current entries in the global navigation history list.

> *Note:* The following paragraph provides an expository description of the internal semantics of the `History` object's state. Elements of this description are further referenced below in the description of this object's properties and methods.

When the user agent is initialized, this global navigation history list is set to an empty state and an internal state variable, here termed *currentIndex*, is set to `null`, indicating that there is no current entry. When navigation occurs to a new document by replacing (1) the current document in the case of a non-frameset document, (2) the top-level frameset of a frameset document, or (3) a frame of a frameset document, then either the first entry or the entry at *currentIndex* is replaced by a reference to the new document, according to whether *currentIndex* is null or non-null, respectively. If *currentIndex* is null, then it is set to an unspecified, but non-null value indicating that the current entry is the first entry; otherwise, *currentIndex* is not modified. If *currentIndex* was not modified by this operation and *currentIndex* does not refer to the last entry, then all entries after *currentIndex* are removed, making *currentIndex* refer to the last entry.

The following subsections describe the entirety of `History` interface semantics.

5.3.1.2.9.1.1 `History::back`

This void valued method shall cause navigation to the previous entry in the global navigation history list.

In the case that the global navigation history list is non-empty and the current document is not the first entry in this list, then the internal state variable *currentIndex* is modified to refer to the

previous entry and the document referenced by this previous entry replaces the current document or frame. Otherwise, if the list is empty or the current document is the first entry in the list, then no side effect is produced.

5.3.1.2.9.1.2 `History::forward`

This void valued method shall cause navigation to the next entry in the global navigation history list.

In the case that the global navigation history list is non-empty and the current document is not the first last in this list, then the internal state variable *currentIndex* is modified to refer to the next entry and the document referenced by this next entry replaces the current document or frame. Otherwise, if the list is empty or the current document is the last entry in the list, then no side effect is produced.

5.3.1.2.9.1.3 `History::go`

This void valued method shall cause navigation to a previous or subsequent entry in the global navigation history list or shall cause the current document to be reinstantiated (i.e., reloaded), where the *index* parameter specifies a relative offset in the global navigation history list with respect to the internal state variable *currentIndex*.

If the value of *index* is zero, then it refers to the current entry in the list. If it is negative, it refers to the *N*th prior entry in the list, where *N* is the absolute value of *index*. If it is positive, it refers to the *N*th subsequent entry in the list. If *N* refers to an entry prior to the first entry or subsequent to the last entry, then no side effect is produced. Otherwise, the internal state variable *currentIndex* is modified to reference the specified entry and the document referenced by this entry replaces the current document or frame. If *currentIndex* is not modified due to *index* being zero, then the current document is reinstantiated (reloaded).

If the argument to this method is string valued and does not have a valid interpretation as a non-negative integer value, then the argument shall be interpreted as a URI. In this case, this method shall have the same effect as if the value of `top.location.href` were set to this URI.

5.3.1.2.9.1.4 `History::length`

This immutable, number valued property shall be either zero, if the global navigation history list is empty, or a positive integer denoting the length of the non-empty list.

> *Note:* Although script content may not mutate this property directly, the user agent will mutate this property as a side effect of content navigation operations. This mutation by the user agent will occur asynchronously with respect to script content evaluation.

5.3.1.2.9.2 Location Object

A `Location` object provides information about the constituents of a URI. It implements the following interface:

```
interface Location
{
  /* read-write properties */
  attribute DOMString hash;
  attribute DOMString host;
  attribute DOMString hostname;
  attribute DOMString href;
  attribute DOMString pathname;
  attribute DOMString port;
  attribute DOMString protocol;
  attribute DOMString search;
};
```

The following subsections describe the entirety of `Location` interface semantics.

5.3.1.2.9.2.1 `Location::[[DefaultValue]]`

The internal `[[DefaultValue]]` method for the `Location` object shall return the value of `Location::href` in the form of a primitive string value irrespective of the value of *hint*. See [ECMASCRIPT], Section 8.6.2.6.

5.3.1.2.9.2.2 `Location::hash`

This mutable, string valued property denotes the *fragment* portion of the parsed representation of the `Location` object's *href* property value, including the `'#'` delimiter (pound sign) that precedes the *fragment*. See [URI], Section 4.1.

When mutating the value of this property, if the new value does not adhere to the *fragment* syntax specified by [URI] including the `'#'` delimiter (pound sign) that precedes *fragment*, then the user agent shall cause a `SYNTAX_ERR` exception to be raised.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `Location::href`.

If the `Location` object is bound to `Window::location`, then mutation of this property's value shall additionally produce a side effect of navigating to the specified fragment, provided the fragment exists in the `Window` object's document instance.

> *Note:* In the preceding paragraph, *navigating* means to cause the referenced fragment to be displayed and moving focus (if active) to the referenced fragment.

5.3.1.2.9.2.3 `Location::host`

This mutable, string valued property denotes the *hostport* portion of the parsed representation of the `Location` object's *href* property value. See [URI], Section 3.2.2.

When mutating the value of this property, if the new value does not adhere to the *hostport* syntax specified by [URI], then the user agent shall cause a `SYNTAX_ERR` exception to be raised.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `Location::href`.

If the `Location` object is bound to `Window::location`, then mutation of this property's value shall additionally produce a side effect of navigating to the newly constructed value of `Location::href`.

> *Note:* In the preceding paragraph, *navigating* means to cause the application resource referenced by the URI value of `Location::href` to replace the `Window` object's current document instance.

If this property is mutated so as to produce an identical value of `Location::href` as existed prior to mutation, then no navigation side effect should result.

5.3.1.2.9.2.4 `Location::hostname`

This mutable, string valued property denotes the *host* portion of the parsed representation of the `Location` object's *href* property value. See [URI], Section 3.2.2.

When mutating the value of this property, if the new value does not adhere to the *host* syntax specified by [URI], then the user agent shall cause a `SYNTAX_ERR` exception to be raised.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `Location::href` and `Location::host`.

If the `Location` object is bound to `Window::location`, then mutation of this property's value shall additionally produce a side effect of navigating to the newly constructed value of `Location::href`.

> *Note:* In the preceding paragraph, *navigating* means to cause the application resource referenced by the URI value of `Location::href` to replace the `Window` object's current document instance.

If this property is mutated so as to produce an identical value of `Location::href` as existed prior to mutation, then no navigation side effect should result.

5.3.1.2.9.2.5 `Location::href`

This mutable, string valued property denotes the entire URI as represented by the `Location` object. See [URI].

When mutating the value of this property, if the new value does not adhere to the URI syntax specified by [URI], then the user agent shall cause a `SYNTAX_ERR` exception to be raised.

Mutation of this property's value shall be reflected in any dependent property of the `Location` object.

If the `Location` object is bound to `Window::location`, then mutation of this property's value shall additionally produce a side effect of navigating to the new value of `Location::href`.

> *Note:* In the preceding paragraph, *navigating* means to cause the application resource referenced by the URI value of `Location::href` to replace the `Window` object's current document instance.

If this property is mutated so as to produce an identical value of `Location::href` as existed prior to mutation, then no navigation side effect should result.

5.3.1.2.9.2.6 `Location::pathname`

This mutable, string valued property denotes the *abs_path* or *rel_path* portion of the parsed representation of the `Location` object's *href* property value, depending on whether the URI is an *absoluteURI* or *relativeURI*, respectively. See [URI], Sections 3 and 5.

When mutating the value of this property, if the new value does not adhere to the *abs_path* or *rel_path* syntax specified by [URI], then the user agent shall cause a `SYNTAX_ERR` exception to be raised.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `Location::href`.

If the `Location` object is bound to `Window::location`, then mutation of this property's value shall additionally produce a side effect of navigating to the newly constructed value of `Location::href`.

> *Note:* In the preceding paragraph, *navigating* means to cause the application resource referenced by the URI value of `Location::href` to replace the `Window` object's current document instance.

If this property is mutated so as to produce an identical value of `Location::href` as existed prior to mutation, then no navigation side effect should result.

5.3.1.2.9.2.7 `Location::port`

This mutable, string valued property denotes the *port* portion of the parsed representation of the `Location` object's *href* property value. See [URI], Section 3.2.2.

When mutating the value of this property, if the new value does not adhere to the *port* syntax specified by [URI], then the user agent shall cause a `SYNTAX_ERR` exception to be raised.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `Location::href` and `Location::host`.

If the `Location` object is bound to `Window::location`, then mutation of this property's value shall additionally produce a side effect of navigating to the newly constructed value of `Location::href`.

> *Note:* In the preceding paragraph, *navigating* means to cause the application resource referenced by the URI value of `Location::href` to replace the `Window` object's current document instance.

If this property is mutated so as to produce an identical value of `Location::href` as existed prior to mutation, then no navigation side effect should result.

5.3.1.2.9.2.8 `Location::protocol`

This mutable, string valued property denotes the *scheme* portion of the parsed representation of the `Location` object's *href* property value, including the `':'` delimiter (colon) that follows the *scheme*. See [URI], Section 3.

When mutating the value of this property, if the new value does not adhere to the *scheme* syntax specified by [URI] including the `':'` delimiter (colon) that follows *scheme*, then the user agent shall cause a `SYNTAX_ERR` exception to be raised.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `Location::href`.

If the `Location` object is bound to `Window::location`, then mutation of this property's value shall additionally produce a side effect of navigating to the newly constructed value of `Location::href`.

> *Note:* In the preceding paragraph, *navigating* means to cause the application resource referenced by the URI value of `Location::href` to replace the `Window` object's current document instance.

If this property is mutated so as to produce an identical value of `Location::href` as existed prior to mutation, then no navigation side effect should result.

5.3.1.2.9.2.9 `Location::search`

This mutable, string valued property denotes the *query* portion of the parsed representation of the `Location` object's *href* property value, including the `'?'` delimiter (question mark) that precedes the *query*. See [URI], Sections 3 and 3.4.

When mutating the value of this property, if the new value does not adhere to the *query* syntax specified by [URI] including the `'?'` delimiter (question mark) that precedes *query*, then the user agent shall cause a `SYNTAX_ERR` exception to be raised.

Mutation of this property's value shall be reflected in any subsequent reference to the value of `Location::href` and `Location::pathname`.

If the `Location` object is bound to `Window::location`, then mutation of this property's value shall additionally produce a side effect of navigating to the newly constructed value of `Location::href`.

> *Note:* In the preceding paragraph, *navigating* means to cause the application resource referenced by the URI value of `Location::href` to replace the `Window` object's current document instance.

If this property is mutated so as to produce an identical value of `Location::href` as existed prior to mutation, then no navigation side effect should result.

5.3.1.2.9.3 Navigator Object

A `Navigator` object provides information about the user agent. It implements the following interface:

```
interface Navigator
{
  /* read-only properties */
  readonly attribute DOMString appName;
  readonly attribute DOMString appVersion;
  readonly attribute DOMString appCodeName;
  readonly attribute DOMString userAgent;
  readonly attribute DOMString ddeBackChannel;
  readonly attribute DOMString ddeContentLevel;
```

```
  readonly attribute DOMString ddeSourceId;
  /* read-write properties */
  attribute boolean ddeEnabled;
  attribute boolean ddeReleasable;
};
```

The following subsections describe the entirety of `Navigator` interface semantics.

5.3.1.2.9.3.1 `Navigator::appCodeName`

This immutable, string valued property shall have a value specified by the application environment implementer.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.9.3.2 `Navigator::appName`

This immutable, string valued property shall have a value specified by the application environment implementer.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.9.3.3 `Navigator::appVersion`

This immutable, string valued property shall have a value specified by the application environment implementer.

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.9.3.4 `Navigator::ddeBackChannel`

This immutable, string valued property shall have a value as follows: if the DASE Application is not a legacy application or does not contain a `Transcoded-From-Profile` *meta* element with the value `"SMPTE DDE-1"`, then the value shall be null; otherwise, the value shall be one of the following: `"permanent"`, `"connected"`, `"disconnected"`, or `"unavailable"`, in which case the default value of this property shall be `"unavailable"` unless prescribed otherwise by a DASE Extension.

> *Note:* This property is provided to obtain interoperability with certain legacy content. Its use is deprecated for non-legacy applications.

5.3.1.2.9.3.5 `Navigator::ddeContentLevel`

This immutable, string valued property shall have a value `"1.0"`.

> *Note:* This property is provided to obtain interoperability with certain legacy content. Its use is deprecated for non-legacy applications.

5.3.1.2.9.3.6 `Navigator::ddeEnabled`

This mutable, boolean valued property shall have a default value of `true`. If the DASE Application is not a legacy application or does not contain a `Transcoded-From-Profile` *meta* element with the value `"SMPTE DDE-1"`, then the value of this property shall not affect the declarative application environment behavior; otherwise, if the value of this property is set to `false`, a declarative application environment shall not dispatch script triggers to the DASE Application.

> *Note:* See Section 4.5.2 for further information on script trigger processing.
>
> *Note:* This property is provided to obtain interoperability with certain legacy content. Its use is deprecated for non-legacy applications.

5.3.1.2.9.3.7 `Navigator::ddeSourceId`

This immutable, string valued property property shall have a value as follows: if the DASE Application is not a legacy application or does not contain a `Transcoded-From-Profile` *meta*

element with the value "SMPTE DDE-1", then the value shall be null; otherwise, the value shall be identical to the value of the *uuid* attribute of the *identifier* element of the DASE Application's metadata resource in accordance with [DASE], Section 6.1.1.6.10.

> *Note:* This property is provided to obtain interoperability with certain legacy content. Its use is deprecated for non-legacy applications.

5.3.1.2.9.3.8 Navigator::ddeReleasable

This mutable, boolean valued property shall have a default value of false. If the DASE Application is not a legacy application or does not contain a Transcoded-From-Profile *meta* element with the value "SMPTE DDE-1", then the value of this property shall not affect the declarative application environment behavior; otherwise, if the value of this property is set to true, then the declarative application environment shall behave as if an atsc-dynamic-refresh style property with the value of auto had been associated with the root *html* element of the document.

> *Note:* See Section 5.2.1.8.3.1 for further information on the atsc-dynamic-refresh style property.

> *Note:* This property is provided to obtain interoperability with certain legacy content. Its use is deprecated for non-legacy applications.

5.3.1.2.9.3.9 Navigator::userAgent

This immutable, string valued property shall have a value that adheres to the syntax prescribed by [HTTP], Section 14.43, for the value of the User-Agent request header. Furthermore, this value shall contain the product token "DASE-1/1.0".

> *Note:* This property is provided to obtain lexical interoperability with legacy [DOM0] content.

5.3.1.2.9.4 Window Object

The Window object serves as the Global object for all script content evaluation. Each frameset and frame in a multi-frame document is associated with a distinct Window object. In the case of a non-frame document, only one Window object is instantiated. A Window object implements the following interface:

```
typedef unsigned long TimerId;

interface Window
{
  /* read-only properties */
  readonly attribute Navigator navigator;
  readonly attribute History history;
  readonly attribute Window window;
  readonly attribute Window self;
  readonly attribute Window parent;
  readonly attribute Window top;
  readonly attribute HTMLCollection frames;
  readonly attribute long length;
  readonly attribute DOMString name;
  readonly attribute HTMLDocument document;
  /* read-write properties */
  attribute Location location;
  attribute DOMString status;
  attribute DOMString defaultStatus;
  attribute Window opener;
  /* methods */
  void alert(in DOMString message);
  boolean confirm(in DOMString message);
  DOMString prompt(in DOMString message, in DOMString defaultResponse);
  TimerId setTimeout(in DOMString statement, in unsigned long delay);
```

```
  void clearTimeout(in TimerId timerId);
  void close();
  Window open(in DOMString url, in DOMString name, in DOMString features);
};
```

The following subsections describe the entirety of `Window` interface semantics.

5.3.1.2.9.4.1 `Window::alert`

This void valued method shall cause the presentation of *message* to the end-user.

This method may be implemented as a modal dialog with a single OK button.

5.3.1.2.9.4.2 `Window::clearTimeout`

This void valued method shall cause the scheduled statement timer identified by *timerId* to be removed from the set of scheduled statement timers if the timer has not already fired and the statement evaluated; otherwise, if the timer has already fired and the statement evaluated, then no side effect shall be produced.

5.3.1.2.9.4.3 `Window::close`

This null valued method shall cause a top-level `Window` object to be destroyed. If invoked on a `Window` object which is not a top-level `Window` object, then no side effect shall result.

If a top-level `Window` object is closed by this method, then (1) all user interactivity with the `Window` and its descendant `Window` objects shall be disabled; (2) all automatic reformatting and redisplay functions for the `Window` object and its descendant `Window` objects shall be disabled; (3) an unload event shall be dispatched to each document associated with the Window object and its descendant `Window` objects; (4) all scheduled statement timers associated with the `Window` and its descendant `Window` objects shall be cleared; and (5) if the `Window` object is the current top-level `Window` object associated with an application, then the underlying application shall be terminated.

A user agent may disallow the semantics of this method (i.e., not close a window) if the current `Window` object was not created by or for the current application. In this case, the user agent shall cause a `NO_CLOSE_ALLOWED_ERR` exception to be raised.

> *Note:* This specification does not require a user agent to support multiple simultaneous applications.

5.3.1.2.9.4.4 `Window::confirm`

This boolean valued method shall cause the presentation of *message* to the end-user, returning `true` if the user selects OK, otherwise returning `false`.

This method may be implemented as a modal dialog with an OK button and a CANCEL button.

5.3.1.2.9.4.5 `Window::defaultStatus`

This mutable, string valued property denotes a default status message intended to be presented to the end-user in a status bar when no other status message is available.

If a user agent implements this facility and provides a status bar or status bar like functionality, then the user agent should present this property's value to the end-user when no other status message is available. The manner in which this value is presented to the end-user is not defined by this specification.

In a frameset document, the frame which has active focus should be used to determine the default status message to be presented to the end-user.

Script content shall not rely on a user agent to present the value of this property to the end-user.

5.3.1.2.9.4.6 `Window::document`

This immutable, object valued property shall reference a mutable HTMLDocument object that represents the `Window` object's current document instance. See Section 5.3.1.2.3.3.

> *Note:* Although script content may not mutate this property, the user agent will mutate this property as a side effect of replacing the `Window` object's current document instance, e.g., as a result of navigating to a new document by mutating `Window::location`.

5.3.1.2.9.4.7 `Window::frames`

This immutable, object value property shall reference an immutable `HTMLCollection` object that, in turn, references zero or more mutable Window objects. In the case that the current `Window` object's document instance is a frameset document, then this collection of `Window` objects shall be the frames of that frameset document in content order; otherwise, this collection shall be empty.

> *Note:* Although script content may not mutate the referenced `HTMLCollection` object, the user agent will mutate this object as a side effect of loading the frameset document and the individual frames referenced by this document. This mutation of the referenced `HTMLCollection` object by the user agent will occur asynchronously with respect to script content evaluation.

Script content which references the `HTMLCollection` object referenced by this property may do so reliably only after the *onload* event has been fired for the frameset.

For each named `Window` object the user agent adds to the referenced `HTMLCollection` object, a new property shall be added to the current `Window` object, where the name of the property is the value of the *name* property of that `Window` object and the value of the property is the `Window` object itself. If an identically named property already exists on the current `Window` object, then a property shall not be added and the value of the pre-existing, identically named property shall not be changed.

5.3.1.2.9.4.8 `Window::history`

This immutable, object valued property shall reference an immutable `History` object shared by all `Window` objects within a top-level viewing window. See Section 5.3.1.2.9.1.

5.3.1.2.9.4.9 `Window::location`

This mutable, object valued property shall reference a mutable `Location` object. See Section 5.3.1.2.9.2.

The semantics of mutation of this property shall be as follows: (1) the new value shall be subjected to *ToString* type conversion (see [ECMASCRIPT], Section 9.8); (2) the resulting string shall be used to mutate the value of `Window::location.href` producing navigation side effects as described in Section 5.3.1.2.9.2.5.

> *Note:* When mutating the value of this property, the reference to the `Location` object may or may not change in accordance with implementation dependent conditions. That is, the user agent implementation may choose to reuse the existing `Location` object and simply change its properties as opposed to instantiating a new `Location` object.

5.3.1.2.9.4.10 `Window::length`

This immutable, number valued property shall have the same value as `Window::-frames.length`. For non-frameset documents, its value shall be zero.

> *Note:* Although script content may not mutate this property, the user agent will mutate this property as a side effect of loading a frameset document and the individual frames referenced by the document. This mutation by the user agent will occur asynchronously with respect to script content evaluation.

Script content which references this property may do so reliably only after the *onload* event has been fired for the frameset.

5.3.1.2.9.4.11 `Window::name`

This immutable, string valued property denotes the name of a frame in a frameset document. The value of this property shall be the normalized value of the *name* attribute of the associated *frame* element of a frameset document, if present, otherwise it shall be the empty string.

5.3.1.2.9.4.12 `Window::navigator`

This immutable, object valued property shall reference an immutable, shared `Navigator` object. See Section 5.3.1.2.9.3.

5.3.1.2.9.4.13 `Window::open`

This object valued method shall either (1) in the case that *name* identifies an existing `Window` object (i.e., if *name* is the same as `Window::name` for some `Window` object instance), cause navigation to the document referenced by *url* in that existing `Window` object, or (2) in the case that *name* does not identify an existing `Window` object, cause loading of the document referenced by *url* into a newly created `Window` object associated with a new top-level viewing window. A reference to the existing or newly created `Window` object is returned.

The *name* argument shall be restricted to tokens consisting of the Latin alphabetic letters (a-zA-Z) or any of the following specially named frame targets according to [HTML], Section 6.16:

- `_blank`
- `_self`
- `_parent`
- `_top`

If *url* is `null` or the empty string, then an empty document (i.e., a `Document` instance with no nodes) shall be associated with the Window object. If *url* is a non-empty string, then it shall adhere to the form of the parameter entity `%URI.datatype;` as described in Section A.1.

A user agent need not support the instantiation of multiple viewing windows. In the case that a user agent does not support instantiation of multiple viewing windows and this method is invoked with a *name* which does not identify an existing `Window` object, then the user agent shall cause a `NOT_SUPPORTED_ERR` exception to be raised.

> *Note:* This specification does not define the manner in which multiple viewing windows are presented to an end-user in the case that a user agent does support multiple viewing windows.

If a user agent supports multiple viewing windows, then it should maintain a distinct `History` object for each top-level viewing window.

5.3.1.2.9.4.14 `Window::opener`

This mutable, object valued property shall initially reference the `Window` object which caused this `Window` object to be created by means of calling `Window::open()`; otherwise, it shall be null if (1) created by other means than calling `Window::open()` or (2) `window.self` is not equal to `window.top`.

This property may be set to null. An attempt to set this property to a value other than null shall cause a NO_MODIFICATION_ALLOWED_ERR exception to be raised.

5.3.1.2.9.4.15 `Window::parent`

This immutable, object valued property shall reference the parent `Window` object, in the case that the current `Window` object is not the top-most frame in a frameset document, or the `Window` object itself, in the case that the current `Window` object is the top-most frame in a frameset document or the document is not a frameset document.

5.3.1.2.9.4.16 `Window::prompt`

This string valued method shall cause the presentation of *message* to and solicit a response from the end-user where the default response is *defaultResponse*, returning the actual response. If this operation is cancelled by the user or the user does not modify the default response, then the default response shall be returned; otherwise, the modified response is returned.

This method may be implemented as a modal dialog with an OK button and a CANCEL button.

5.3.1.2.9.4.17 `Window::self`

This immutable, object valued property shall reference the `Window` object itself.

5.3.1.2.9.4.18 `Window::setTimeout`

This number valued method shall cause the one-time evaluation of *statement* to occur in *delay* milliseconds, returning a unique, opaque numeric identifier which may be subsequently used by `Window::clearTimeout`.

The form of *statement* shall be in accordance with the *StatementList* non-terminal of [ECMASCRIPT], Section 12.1; in particular, *statement* shall be syntactically valid for use as the content of the *Block* non-terminal.

The execution context for evaluating *statement* shall be in accordance with the global code type of [ECMASCRIPT], Section 10.2.1, with the global object being the current `Window` object in whose context this method is invoked.

If one or more statements scheduled by this method remain outstanding at the time the current `Window` is destroyed, then those statements shall be descheduled; i.e., an implicit `Window::clearTimeout` shall be applied to each outstanding scheduled statement timer.

> *Note:* Certain constraints apply to the semantics of host objects when those objects are accessed or their methods invoked by a scheduled statement timer; in particular, see Section 5.3.1.2.3.3.11.

Script content shall not rely on the relative order or precise timing of the evaluation of scheduled statement timers. For example, if this method is invoked twice in immediate succession with the same *delay* parameter, then it cannot be assured that the first scheduled statement will be evaluated before the second scheduled statement.

5.3.1.2.9.4.19 `Window::status`

This mutable, string valued property denotes a status message intended to be presented to the end-user in a status bar.

If a user agent implements this facility and provides a status bar or status bar like functionality, then the user agent should present this property's value to the end-user. The manner in which this value is presented to the end-user is not defined by this specification.

If a user agent presents this status message to the end-user, then mutation of this property in any `Window` object of a frameset document should be presented to the end-user. No serialization order for the presentation of this value among multiple `Window` objects is defined by this specification.

Script content shall not rely on a user agent to present the value of this property to the end-user.

5.3.1.2.9.4.20 `Window::top`

This immutable, object valued property shall reference the top-most `Window` object, in the case that the current Window object is not the top-most frame in a frameset document, or the `Window` object itself, in the case that the current `Window` object is the top-most frame in a frameset document or the document is not a frameset document.

5.3.1.2.9.4.21 `Window::window`

This immutable, object valued property shall reference the `Window` object itself.

ANNEX A. DOCUMENT TYPE DEFINITIONS

The entirety of this section and its subsections is normative.

This annex specifies a document type definition according to [XTHMLMOD] for use with markup content.

A.1 XDML Document Type

This document type defines the declarative application markup language.

A.1.1 DTD Driver

```
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                          XDML 1.0 DTD Driver                          -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!--
     This is XDML 1.0, an XHTML Host Language Document Type specified for
     use with ATSC DASE Declarative Applications.

     This module shall be identified by the following formal public identifier:

     "-//ATSC//DTD XHTML XDML 1.0//EN"
-->

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                        XHTML Driver Parameters                        -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!ENTITY % XHTML.version "-//ATSC//DTD XHTML XDML 1.0//EN">
<!ENTITY % XHTML.profile "">

<!ENTITY % XHTML.dtd.sysid.base
           "http://www.w3.org/TR/xhtml-modularization/DTD/">

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                                Framework                              -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!-- Framework Parameter: Include BIDI support -->
<!ENTITY % XHTML.bidi "INCLUDE">

<!-- Framework Parameter: Include intrinsic event attributes -->
<!ENTITY % xhtml-events.module "INCLUDE">

<!-- Framework Parameter: Define Content Model -->
<!ENTITY % xhtml-model.mod
    PUBLIC  "-//ATSC//ENTITIES XDML Content Model 1.0//EN"
            "xdml-model-1.ent">

<!-- Framework Parameter: %Core.attrib; extensions -->
<!ENTITY % base.attrib "xml:base CDATA #IMPLIED">
<!ENTITY % style.attrib "style CDATA #IMPLIED">
<!ENTITY % Core.extra.attrib "%base.attrib; %style.attrib;">

<!-- Framework Parameter: missing qnames -->
<!ENTITY % frameset.qname "frameset">

<!-- Modular Framework Module -->
<!ENTITY % xhtml-framework.mod
    PUBLIC  "-//W3C//ENTITIES XHTML Modular Framework 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-framework-1.mod">
%xhtml-framework.mod;

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                             Element Modules                           -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
```

```
<!-- Basic Text Module -->
<!ENTITY % xhtml-text.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Text 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-text-1.mod">
%xhtml-text.mod;

<!-- Hypertext Module -->
<!ENTITY % xhtml-hypertext.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Hypertext 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-hypertext-1.mod">
%xhtml-hypertext.mod;

<!-- Lists Module -->
<!ENTITY % xhtml-list.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Lists 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-list-1.mod">
%xhtml-list.mod;

<!-- Presentation Module -->
<!ENTITY % xhtml-pres.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Presentation 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-pres-1.mod">
%xhtml-pres.mod;

<!-- BIDI Override Element Module -->
<!ENTITY % xhtml-bdo.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML BIDI Override Element 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-bdo-1.mod">
%xhtml-bdo.mod;

<!-- Forms Module -->
<!ENTITY % xhtml-form.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Forms 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-form-1.mod">
%xhtml-form.mod;

<!-- Tables Module -->
<!ENTITY % xhtml-table.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Tables 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-table-1.mod">
%xhtml-table.mod;

<!-- Param Element Module -->
<!ENTITY % xhtml-param.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Param Element 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-param-1.mod">
%xhtml-param.mod;

<!-- Object Element Module -->
<!ENTITY % xhtml-object.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Embedded Object 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-object-1.mod">
%xhtml-object.mod;

<!-- Image Element Module -->
<!ENTITY % xhtml-image.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Images 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-image-1.mod">
%xhtml-image.mod;

<!-- Client-side Image Map Module -->
<!ENTITY % xhtml-csismap.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Client-side Image Maps 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-csismap-1.mod">
%xhtml-csismap.mod;

<!-- Link Element Module -->
<!ENTITY % xhtml-link.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Link Element 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-link-1.mod">
%xhtml-link.mod;
```

```
<!-- Base Element Module -->
<!ENTITY % xhtml-base.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Base Element 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-base-1.mod">
%xhtml-base.mod;

<!-- Document Metainformation Module -->
<!ENTITY % xhtml-meta.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Metainformation 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-meta-1.mod">
%xhtml-meta.mod;

<!-- Scripting Module -->
<!ENTITY % xhtml-script.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Scripting 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-script-1.mod">
%xhtml-script.mod;

<!-- Stylesheets Module -->
<!ENTITY % xhtml-style.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Style Sheets 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-style-1.mod">
%xhtml-style.mod;

<!-- Target Attribute Module -->
<!ENTITY % xhtml-target.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Target 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-target-1.mod">
%xhtml-target.mod;

<!-- Frames Module -->
<!ENTITY % xhtml-frames.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Frames 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-frames-1.mod">
%xhtml-frames.mod;

<!-- Document Structure Module -->
<!ENTITY % xhtml-struct.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Document Structure 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-struct-1.mod">
%xhtml-struct.mod;

<!-- Name Identifier Module -->
<!ENTITY % xhtml-nameident.mod
    PUBLIC  "-//W3C//ELEMENTS XHTML Name Identifier 1.0//EN"
            "%XHTML.dtd.sysid.base;xhtml-nameident-1.mod">
%xhtml-nameident.mod;

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                      Non-Parameterized Extensions                   -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<![%legend.attlist;[
<!ATTLIST %legend.qname;
      align        ( top | bottom | left | right ) #IMPLIED
>
]]>

<![%param.attlist;[
<!ATTLIST %param.qname;
      %base.attrib;
>
]]>

<![%meta.attlist;[
<!ATTLIST %meta.qname;
      %base.attrib;
>
]]>
```

```
<![%script.attlist;[
<!ATTLIST %script.qname;
      %base.attrib;
>
]]>

<![%style.attlist;[
<!ATTLIST %style.qname;
      %base.attrib;
>
]]>

<![%title.attlist;[
<!ATTLIST %title.qname;
      %base.attrib;
>
]]>

<![%head.attlist;[
<!ATTLIST %head.qname;
      %base.attrib;
>
]]>

<![%html.attlist;[
<!ATTLIST %html.qname;
      %base.attrib;
>
]]>

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                              END END END                              -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
```

A.1.2 Document (Content) Model

```
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                   XDML 1.0 Document (Content) Model                   -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!--
     This is the XDML 1.0 Document (Content) Model Module for use with the
     XDML 1.0 Document Type specified for ATSC DASE Declarative Applications.

     This module declares certain parameter entities that define groupings
     of elements employed in the definition of XDML 1.0 content models.

     This module shall be idenitifed by the following formal public identifier:

     "-//ATSC//ENTITIES XDML Content Model 1.0//EN"
-->

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                    Non-Empty Content Group Classes                    -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!ENTITY % BlkStruct.class      "%p.qname;
                                 |%div.qname;">

<!ENTITY % InlStruct.class      "%br.qname;
                                 |%span.qname;">

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                 Optionally Empty Content Group Classes                 -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!ENTITY % Anchor.class         "|%a.qname;">

<!ENTITY % BlkPhras.class       "|%pre.qname;
                                 |%blockquote.qname;
                                 |%address.qname;">
```

```
<!ENTITY % BlkPres.class        "|%hr.qname;">

<!ENTITY % Heading.class        "|%h1.qname;
                                 |%h2.qname;
                                 |%h3.qname;
                                 |%h4.qname;
                                 |%h5.qname;
                                 |%h6.qname;">

<!ENTITY % I18n.class           "|%bdo.qname;">

<!ENTITY % InlPhras.class       "|%abbr.qname;
                                 |%acronym.qname;
                                 |%cite.qname;
                                 |%code.qname;
                                 |%dfn.qname;
                                 |%em.qname;
                                 |%kbd.qname;
                                 |%q.qname;
                                 |%samp.qname;
                                 |%strong.qname;
                                 |%var.qname;">

<!ENTITY % InlPres.class        "|%b.qname;
                                 |%big.qname;
                                 |%i.qname;
                                 |%small.qname;
                                 |%sub.qname;
                                 |%sup.qname;
                                 |%tt.qname;">

<!ENTITY % InlForm.class        "|%input.qname;
                                 |%select.qname;
                                 |%textarea.qname;
                                 |%label.qname;
                                 |%button.qname;">

<!ENTITY % InlSpecial.class     "|%map.qname;
                                 |%img.qname;
                                 |%object.qname;">

<!ENTITY % List.class           "|%ul.qname;
                                 |%ol.qname;
                                 |%dl.qname;">

<!ENTITY % Table.class          "|%table.qname;">

<!ENTITY % Misc.class           "|%script.qname;
                                 |%noscript.qname;">

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--              Optionally Empty Content Group Class Extras              -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!ENTITY % Block.extra          "%Table.class;
                                 |%form.qname;
                                 |%fieldset.qname;">

<!ENTITY % BlkNoForm.extra      "%Table.class;">

<!ENTITY % BlkNoTable.extra     "|%form.qname;
                                 |%fieldset.qname;">

<!ENTITY % Inline.extra         "">

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                    Aggregate Content Group Classes                    -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!ENTITY % Block.class          "%BlkStruct.class;
```

```
                                %BlkPhras.class;
                                %BlkPres.class;
                                %Block.extra;">

<!ENTITY % BlkNoForm.class      "%BlkStruct.class;
                                %BlkPhras.class;
                                %BlkPres.class;
                                %BlkNoForm.extra;">

<!ENTITY % BlkNoTable.class     "%BlkStruct.class;
                                %BlkPhras.class;
                                %BlkPres.class;
                                %BlkNoTable.extra;">

<!ENTITY % Inline.class         "%InlStruct.class;
                                %I18n.class;
                                %InlPhras.class;
                                %InlPres.class;
                                %InlSpecial.class;
                                %InlForm.class;
                                %Inline.extra;
                                %Anchor.class;">

<!ENTITY % InlNoAnchor.class    "%InlStruct.class;
                                %I18n.class;
                                %InlPhras.class;
                                %InlPres.class;
                                %InlSpecial.class;
                                %InlForm.class;
                                %Inline.extra;">

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                          Content Group Mixes                        -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!ENTITY % Block.mix            "%Block.class;
                                %List.class;
                                %Heading.class;
                                %Misc.class;">

<!ENTITY % BlkNoForm.mix        "%BlkNoForm.class;
                                %List.class;
                                %Heading.class;
                                %Misc.class;">

<!ENTITY % BlkNoTable.mix       "%BlkNoTable.class;
                                %List.class;
                                %Heading.class;
                                %Misc.class;">

<!ENTITY % HeadOpts.mix         "(%script.qname;
                                 |%style.qname;
                                 |%meta.qname;
                                 |%link.qname;
                                 |%object.qname;)*">

<!ENTITY % Flow.mix             "%Block.class;
                                %List.class;
                                %Heading.class;
                                |%Inline.class;
                                %Misc.class;">

<!ENTITY % FlowNoTable.mix      "%BlkNoTable.class;
                                %List.class;
                                %Heading.class;
                                |%Inline.class;
                                %Misc.class;">

<!ENTITY % Inline.mix           "%Inline.class;
                                %Misc.class;">
```

```
<!ENTITY % InlNoAnchor.mix      "%InlNoAnchor.class;
                                 %Misc.class;">

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                 Element Content Model Predeclarations                 -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!ENTITY % html.content         "( %head.qname;,
                                   ( %body.qname; | %frameset.qname; ) )">

<!ENTITY % noscript.content     "( #PCDATA | %Flow.mix; )*">

<!ENTITY % blockquote.content   "( #PCDATA | %Flow.mix; )*">

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
<!--                              END END END                              -->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->
```

ANNEX B. DEFAULT STYLESHEET

The entirety of this section and its subsections is normative.

This annex specifies a default stylesheet which is recommended for use by user agents in the presentation of XHTML content. The content type of this stylesheet is `text/css`.

B.1 Stylesheet

In the following stylesheet, the font-relative length unit *em* shall be interpreted according to a reference font of size 24 pixels on a reference display of 480 pixels in height.

```
@media atsc-tv {

/* non-contextual rules */
abbr        { font-variant: small-caps; letter-spacing: 0.1em }
acronym     { font-variant: small-caps; letter-spacing: 0.1em }
address     { display: block; font-style: italic }
a[href]     { text-decoration: underline }
b           { font-weight: bolder }
big         { font-size: 1.33em }
blockquote  { display: block; margin: 1.33em 12.70mm 0 }
body        { display: block; padding: 8px; line-height: 1.33 }
br:before   { content: "\a" }
caption     { text-align: center }
cite        { font-style: italic }
code        { font-family: monospace }
dd          { display: block; margin-left: 12.70mm }
div         { display: block }
dl          { display: block; margin: 1.33em 0 }
dt          { display: block }
em          { font-style: italic }
fieldset    { display: block; margin: 1.33em 0 }
form        { display: block; margin: 1.33em 0 }
frame       { display: block }
frameset    { display: block }
h1          { display: block; font-weight: bolder; font-size: 2.67em; margin: 0.67em 0 }
h2          { display: block; font-weight: bolder; font-size: 2.00em; margin: 0.83em 0 }
h3          { display: block; font-weight: bolder; font-size: 1.33em; margin: 1.00em 0 }
h4          { display: block; font-weight: bolder; font-size: 1.00em; margin: 1.33em 0 }
h5          { display: block; font-weight: bolder; font-size: 0.75em; margin: 1.67em 0 }
h6          { display: block; font-weight: bolder; font-size: 0.50em; margin: 2.33em 0 }
head        { display: none }
hr          { display: block; border: 2px inset }
i           { font-style: italic }
kbd         { font-family: monospace }
noframes    { display: block }
object      { display: block }
ol          { display: block; list-style-type: decimal; margin: 1.33em 12.70mm 0 }
p           { display: block; margin: 1.33em 0 }
pre         { display: block; font-family: monospace; white-space: pre }
samp        { font-family: monospace }
small       { font-size: 0.75em }
strong      { font-weight: bolder }
sub         { font-size: 0.75em; vertical-align: sub }
sup         { font-size: 0.75em; vertical-align: super }
th          { font-weight: bolder; text-align: center }
tt          { font-family: monospace }
ul          { display: block; list-style-type: disc; margin: 1.33em 12.70mm 0 }
var         { font-style: italic }

/* contextual rules */
ol ol       { margin-top: 0; margin-bottom: 0 }
ol ul       { margin-top: 0; margin-bottom: 0 }
ul ol       { margin-top: 0; margin-bottom: 0 }
ul ul       { margin-top: 0; margin-bottom: 0 }

}
```

B.2 Default Fonts

It is recommended that a user agent provide built-in fonts of *serif*, *sans-serif*, and *monospace* logical font families, with *plain*, *italic* (or *oblique*), *bold*, and *bold-italic* (or *bold-oblique*) faces, in sizes of 8, 12, 18, 24, 32, 48, and 64 *reference pixels*, where a *reference pixel* is a pixel on a reference display of 480 pixels in height.

It is recommended that a user agent employ [TIRESIAS] for sans serif, plain face usage.

At a minimum, built-in fonts shall support the display of all Unicode characters in the range of 0x0021 through 0x007E and 0x00A0 through 0x00FF (i.e., ISO Latin 1 printable characters).

ANNEX C. DOCUMENT OBJECT MODEL INTERFACES

The entirety of this section and its subsections is normative.

This annex contains the complete [IDL] specification for the document object model interface definitions specified in this document. The definitions are divided into *Native Extensions*, *Core Extensions*, *Environment Extensions*, *Events Extensions*, *HTML Extensions*, and *View Extensions* modules.

In addition to the following modules, this specification incorporates, by reference, the [IDL] specification of certain document object model interfaces defined by [DOM2]; in particular, the following modules specified by [DOM2], [DOM2-HTML], [DOM2-VIEWS], [DOM2-STYLE], and [DOM2-EVENTS] *IDL Definitions*. Inclusion of a particular [DOM2] interface module does not mean that all interfaces specified by that module must be supported by a user agent. See Section 5.3.1.2 for information on which interfaces must be supported by a user agent.

Note: See [DOM0] for further information regarding *Environment* module functionality.

C.1 Native Extensions Module

```
// File: native.idl
#ifndef _NATIVE_IDL_
#define _NATIVE_IDL_

#pragma prefix "org.atsc.dom"

module native
{

  interface String;

  interface StringExt
  {
    /* methods */
      String anchor(in String name);
      String big();
      String blink();
      String bold();
      String fixed();
      String fontcolor(in String color);
      String fontsize(in String size);
      String italics();
      String link(in String href);
      String small();
      String strike();
      String sub();
      String sup();
  };
};

#endif // _NATIVE_IDL_
```

C.2 Core Extensions Module

```
// File: core.idl
#ifndef _CORE_IDL_
#define _CORE_IDL_

#pragma prefix "org.atsc.dom"

#include "dom.idl"

module core
{

  const unsigned short VALIDATION_ERR = 1;
  const unsigned short NO_CLOSE_ALLOWED_ERR = 2;
```

```
  exception DOMExceptionExt
  {
    unsigned short codeExtension;
  };

};

#endif // _CORE_IDL_
```

C.3 Environment Extensions Module

```
// File: environment.idl
#ifndef _ENVIRONMENT_IDL_
#define _ENVIRONMENT_IDL_

#pragma prefix "org.atsc.dom"

#include "dom.idl"
#include "html.idl"

module environment
{

  typedef dom::DOMString DOMString;
  typedef html::HTMLCollection HTMLCollection;
  typedef html::HTMLDocument HTMLDocument;
  typedef long TimerId;

  interface History;
  interface Location;
  interface Navigator;
  interface Window;

  interface History
  {
    /* read-write properties */
      readonly attribute long length;
    /* methods */
      void back();
      void forward();
      void go(in long index);
      void go(in DOMString uri);
  };

  interface Location
  {
    /* read-write properties */
      attribute DOMString hash;
      attribute DOMString host;
      attribute DOMString hostname;
      attribute DOMString href;
      attribute DOMString pathname;
      attribute DOMString port;
      attribute DOMString protocol;
      attribute DOMString search;
  };

  interface Navigator
  {
    /* read-only properties */
      readonly attribute DOMString appName;
      readonly attribute DOMString appVersion;
      readonly attribute DOMString appCodeName;
      readonly attribute DOMString userAgent;
      readonly attribute DOMString ddeBackChannel;
      readonly attribute DOMString ddeContentLevel;
      readonly attribute DOMString ddeSourceId;
    /* read-write properties */
      attribute boolean ddeEnabled;
```

```
      attribute boolean ddeReleasable;
  };

  interface Window
  {
    /* read-only properties */
      readonly attribute Navigator navigator;
      readonly attribute History history;
      readonly attribute Window window;
      readonly attribute Window self;
      readonly attribute Window parent;
      readonly attribute Window top;
      readonly attribute HTMLCollection frames;
      readonly attribute long length;
      readonly attribute DOMString name;
      readonly attribute HTMLDocument document;
    /* read-write properties */
      attribute Location location;
      attribute DOMString status;
      attribute DOMString defaultStatus;
      attribute Window opener;
    /* methods */
      void alert(in DOMString message);
      boolean confirm(in DOMString message);
      DOMString prompt(in DOMString message, in DOMString defaultResponse);
      TimerId setTimeout(in DOMString statement, in unsigned long delay);
      void clearTimeout(in TimerId timerId);
      void close();
      Window open(in DOMString url, in DOMString name, in DOMString features);
  };

};

#endif // _ENVIRONMENT_IDL_
```

C.4 Events Extensions Module

```
// File: events.idl
#ifndef _EVENTS_IDL_
#define _EVENTS_IDL_

#pragma prefix "org.atsc.dom"

#include "dom.idl"

module events
{

  const unsigned short VALIDATION_ERR = 1;
  const unsigned short NO_CLOSE_ALLOWED_ERR = 2;

  interface KeyEvent
  {
    /* read-write properties */
      attribute DOMString outputString;
      attribute unsigned long keyVal;
      attribute unsigned long virtKeyVal;
      attribute boolean visibleOutputGenerated;
      attribute boolean numPad;
  /* methods */
      void initKeyEvent(
        in DOMString type,
        in boolean canBubble,
        in boolean cancelable,
        in views::AbstractView view,
        in long detail,
        in DOMString outputString,
        in unsigned long keyVal,
        in unsigned long virtKeyVal,
        in boolean visibleOutputGenerated,
        in boolean numPad);
```

```
    void initModifier(in unsigned long modifier, in boolean value);
  boolean checkModifier(in unsigned long modifier);
};

interface KeyModifiers
{
  const unsigned long MOD_NONE                    = 0;
  const unsigned long MOD_SHIFT                   = 1;
  const unsigned long MOD_CONTROL                 = 2;
  const unsigned long MOD_META                    = 4;
  const unsigned long MOD_ALT                     = 8;
};

interface VirtualKeys
{
  const unsigned long VK_UNDEFINED                = 0;
  const unsigned long VK_CANCEL                   = 3;
  const unsigned long VK_BACK_SPACE               = 8;
  const unsigned long VK_TAB                      = 9;
  const unsigned long VK_ENTER                    = 10;
  const unsigned long VK_CLEAR                    = 12;
  const unsigned long VK_SHIFT                    = 16;
  const unsigned long VK_CONTROL                  = 17;
  const unsigned long VK_ALT                      = 18;
  const unsigned long VK_PAUSE                    = 19;
  const unsigned long VK_CAPS_LOCK                = 20;
  const unsigned long VK_KANA                     = 21;
  const unsigned long VK_FINAL                    = 24;
  const unsigned long VK_KANJI                    = 25;
  const unsigned long VK_ESCAPE                   = 27;
  const unsigned long VK_CONVERT                  = 28;
  const unsigned long VK_NONCONVERT               = 29;
  const unsigned long VK_ACCEPT                   = 30;
  const unsigned long VK_MODECHANGE               = 31;
  const unsigned long VK_SPACE                    = 32;
  const unsigned long VK_PAGE_UP                  = 33;
  const unsigned long VK_PAGE_DOWN                = 34;
  const unsigned long VK_END                      = 35;
  const unsigned long VK_HOME                     = 36;
  const unsigned long VK_LEFT                     = 37;
  const unsigned long VK_UP                       = 38;
  const unsigned long VK_RIGHT                    = 39;
  const unsigned long VK_DOWN                     = 40;
  const unsigned long VK_COMMA                    = 44;
  const unsigned long VK_PERIOD                   = 46;
  const unsigned long VK_SLASH                    = 47;
  const unsigned long VK_0                        = 48;
  const unsigned long VK_1                        = 49;
  const unsigned long VK_2                        = 50;
  const unsigned long VK_3                        = 51;
  const unsigned long VK_4                        = 52;
  const unsigned long VK_5                        = 53;
  const unsigned long VK_6                        = 54;
  const unsigned long VK_7                        = 55;
  const unsigned long VK_8                        = 56;
  const unsigned long VK_9                        = 57;
  const unsigned long VK_SEMICOLON                = 59;
  const unsigned long VK_EQUALS                   = 61;
  const unsigned long VK_A                        = 65;
  const unsigned long VK_B                        = 66;
  const unsigned long VK_C                        = 67;
  const unsigned long VK_D                        = 68;
  const unsigned long VK_E                        = 69;
  const unsigned long VK_F                        = 70;
  const unsigned long VK_G                        = 71;
  const unsigned long VK_H                        = 72;
  const unsigned long VK_I                        = 73;
  const unsigned long VK_J                        = 74;
  const unsigned long VK_K                        = 75;
  const unsigned long VK_L                        = 76;
  const unsigned long VK_M                        = 77;
```

```
const unsigned long VK_N                        = 78;
const unsigned long VK_O                        = 79;
const unsigned long VK_P                        = 80;
const unsigned long VK_Q                        = 81;
const unsigned long VK_R                        = 82;
const unsigned long VK_S                        = 83;
const unsigned long VK_T                        = 84;
const unsigned long VK_U                        = 85;
const unsigned long VK_V                        = 86;
const unsigned long VK_W                        = 87;
const unsigned long VK_X                        = 88;
const unsigned long VK_Y                        = 89;
const unsigned long VK_Z                        = 90;
const unsigned long VK_OPEN_BRACKET             = 91;
const unsigned long VK_BACK_SLASH               = 92;
const unsigned long VK_CLOSE_BRACKET            = 93;
const unsigned long VK_NUMPAD0                  = 96;
const unsigned long VK_NUMPAD1                  = 97;
const unsigned long VK_NUMPAD2                  = 98;
const unsigned long VK_NUMPAD3                  = 99;
const unsigned long VK_NUMPAD4                  = 100;
const unsigned long VK_NUMPAD5                  = 101;
const unsigned long VK_NUMPAD6                  = 102;
const unsigned long VK_NUMPAD7                  = 103;
const unsigned long VK_NUMPAD8                  = 104;
const unsigned long VK_NUMPAD9                  = 105;
const unsigned long VK_MULTIPLY                 = 106;
const unsigned long VK_ADD                      = 107;
const unsigned long VK_SEPARATER                = 108;
const unsigned long VK_SUBTRACT                 = 109;
const unsigned long VK_DECIMAL                  = 110;
const unsigned long VK_DIVIDE                   = 111;
const unsigned long VK_F1                       = 112;
const unsigned long VK_F2                       = 113;
const unsigned long VK_F3                       = 114;
const unsigned long VK_F4                       = 115;
const unsigned long VK_F5                       = 116;
const unsigned long VK_F6                       = 117;
const unsigned long VK_F7                       = 118;
const unsigned long VK_F8                       = 119;
const unsigned long VK_F9                       = 120;
const unsigned long VK_F10                      = 121;
const unsigned long VK_F11                      = 122;
const unsigned long VK_F12                      = 123;
const unsigned long VK_DELETE                   = 127;
const unsigned long VK_NUM_LOCK                 = 144;
const unsigned long VK_SCROLL_LOCK              = 145;
const unsigned long VK_PRINTSCREEN              = 154;
const unsigned long VK_INSERT                   = 155;
const unsigned long VK_HELP                     = 156;
const unsigned long VK_META                     = 157;
const unsigned long VK_BACK_QUOTE               = 192;
const unsigned long VK_QUOTE                    = 222;
const unsigned long VK_COLORED_KEY_0            = 403;
const unsigned long VK_COLORED_KEY_1            = 404;
const unsigned long VK_COLORED_KEY_2            = 405;
const unsigned long VK_COLORED_KEY_3            = 406;
const unsigned long VK_COLORED_KEY_4            = 407;
const unsigned long VK_COLORED_KEY_5            = 408;
const unsigned long VK_POWER                    = 409;
const unsigned long VK_DIMMER                   = 410;
const unsigned long VK_REWIND                   = 412;
const unsigned long VK_STOP                     = 413;
const unsigned long VK_EJECT_TOGGLE             = 414;
const unsigned long VK_PLAY                     = 415;
const unsigned long VK_RECORD                   = 416;
const unsigned long VK_FAST_FWD                 = 417;
const unsigned long VK_PLAY_SPEED_UP            = 418;
const unsigned long VK_PLAY_SPEED_DOWN          = 419;
const unsigned long VK_PLAY_SPEED_RESET         = 420;
const unsigned long VK_RECORD_SPEED_NEXT        = 421;
```

```
    const unsigned long VK_GO_TO_START              = 422;
    const unsigned long VK_GO_TO_END                = 423;
    const unsigned long VK_TRACK_PREV               = 424;
    const unsigned long VK_TRACK_NEXT               = 425;
    const unsigned long VK_RANDOM_TOGGLE            = 426;
    const unsigned long VK_CHANNEL_UP               = 427;
    const unsigned long VK_CHANNEL_DOWN             = 428;
    const unsigned long VK_STORE_FAVORITE_0         = 429;
    const unsigned long VK_STORE_FAVORITE_1         = 430;
    const unsigned long VK_STORE_FAVORITE_2         = 431;
    const unsigned long VK_STORE_FAVORITE_3         = 432;
    const unsigned long VK_RECALL_FAVORITE_0        = 433;
    const unsigned long VK_RECALL_FAVORITE_1        = 434;
    const unsigned long VK_RECALL_FAVORITE_2        = 435;
    const unsigned long VK_RECALL_FAVORITE_3        = 436;
    const unsigned long VK_CLEAR_FAVORITE_0         = 437;
    const unsigned long VK_CLEAR_FAVORITE_1         = 438;
    const unsigned long VK_CLEAR_FAVORITE_2         = 439;
    const unsigned long VK_CLEAR_FAVORITE_3         = 440;
    const unsigned long VK_SCAN_CHANNELS_TOGGLE     = 441;
    const unsigned long VK_PINP_TOGGLE              = 442;
    const unsigned long VK_SPLIT_SCREEN_TOGGLE      = 443;
    const unsigned long VK_DISPLAY_SWAP             = 444;
    const unsigned long VK_SCREEN_MODE_NEXT         = 445;
    const unsigned long VK_VIDEO_MODE_NEXT          = 446;
    const unsigned long VK_VOLUME_UP                = 447;
    const unsigned long VK_VOLUME_DOWN              = 448;
    const unsigned long VK_MUTE                     = 449;
    const unsigned long VK_SURROUND_MODE_NEXT       = 450;
    const unsigned long VK_BALANCE_RIGHT            = 451;
    const unsigned long VK_BALANCE_LEFT             = 452;
    const unsigned long VK_FADER_FRONT              = 453;
    const unsigned long VK_FADER_REAR               = 454;
    const unsigned long VK_BASS_BOOST_UP            = 455;
    const unsigned long VK_BASS_BOOST_DOWN          = 456;
    const unsigned long VK_INFO                     = 457;
    const unsigned long VK_GUIDE                    = 458;
    const unsigned long VK_TELETEXT                 = 459;
    const unsigned long VK_SUBTITLE                 = 460;
  };

};

#endif // _EVENTS_IDL_
```

C.5 HTML Extensions Module

```
// File: html.idl
#ifndef _HTML_IDL_
#define _HTML_IDL_

#pragma prefix "org.atsc.dom"

#include "dom.idl"

module html
{

  typedef dom::DOMString DOMString;
  typedef environment::Window Window;

  interface HTMLAnchorElementExt
  {
    /* read-write properties */
      attribute DOMString hash;
      attribute DOMString host;
      attribute DOMString hostname;
      attribute DOMString pathname;
      attribute DOMString port;
      attribute DOMString protocol;
      attribute DOMString search;
```

```
  };

  interface HTMLDocumentExt
  {
    /* read-only properties */
      readonly attribute DOMString location;
      readonly attribute DOMString lastModified;
      readonly attribute Window window;
    /* read-write properties */
      attribute DOMString alinkColor;
      attribute DOMString vlinkColor;
      attribute DOMString linkColor;
      attribute DOMString bgColor;
      attribute DOMString fgColor;
    /* methods */
      void clear();
  };

  interface HTMLFormElementExt
  {
    /* read-write properties */
      attribute DOMString encoding;
  };

  interface HTMLImageElementExt
  {
    /* read-only properties */
      readonly attribute boolean complete;
    /* read-write properties */
      attribute DOMString lowsrc;
  };

  interface HTMLObjectElementExt
  {
    /* read-only properties */
      readonly attribute boolean complete;
    /* read-write properties */
      attribute DOMString lowsrc;
      attribute DOMString src;
  };

  interface HTMLTriggerObjectElementExt
  {
    /* read-only properties */
      readonly attribute DOMString backChannel;
      readonly attribute DOMString contentLevel;
      readonly attribute DOMString sourceId;
    /* read-write properties */
      attribute boolean enabled;
      attribute boolean releasable;
  };

};

#endif // _HTML_IDL_
```

C.6 Views Extensions Module

```
// File: views.idl
#ifndef _VIEWS_IDL_
#define _VIEWS_IDL_

#pragma prefix "org.atsc.dom"

#include "dom.idl"

module views
{

  interface DocumentViewExt
  {
```

```
    /* read-only properties */
      readonly attribute double refWidthMm;
      readonly attribute double refHeightMm;
      readonly attribute double gfxPosX;
      readonly attribute double gfxPosY;
      readonly attribute double gfxWidth;
      readonly attribute double gfxHeight;
      readonly attribute unsigned long gfxWidthPx;
      readonly attribute unsigned long gfxHeightPx;
      readonly attribute double vidPosX;
      readonly attribute double vidPosY;
      readonly attribute double vidWidth;
      readonly attribute double vidHeight;
      readonly attribute unsigned long vidWidthPx;
      readonly attribute unsigned long vidHeightPx;
      readonly attribute unsigned long sampleBitsR;
      readonly attribute unsigned long sampleBitsG;
      readonly attribute unsigned long sampleBitsB;
      readonly attribute unsigned long sampleBitsA;
    /* read-write properties */
      attribute boolean refreshOnChange;
    /* methods */
      boolean refresh(in DOMString id);
  };

};

#endif // _VIEWS_IDL
```

ANNEX D. DOCUMENT OBJECT MODEL BINDINGS

The entirety of this section and its subsections is normative.

This annex contains the complete ECMAScript binding for the document object model interface definitions specified in this document as well as certain [DOM2] interfaces. The bindings are divided according to the modules in which their [IDL] interfaces are defined.

In addition to the following bindings, this specification incorporates, by reference, the bindings of certain document object model interfaces defined by [DOM2]; in particular, the bindings for the following modules specified by [DOM2]], [DOM2-HTML], [DOM2-VIEWS], [DOM2-STYLE], and [DOM2-EVENTS] *ECMAScript Language Bindings*. Inclusion of a particular [DOM2] module binding does not mean that all objects, methods, and properties specified by that binding must be supported by a user agent. See Section 5.3.1.2 for information on which objects, methods, and properties must be supported by a user agent.

Note: See [DOM0] for further information regarding *Environment* module bindings.

D.1 Native Extensions Module Bindings

Object **String**

The **String** object has the following additional methods (in addition to native methods):

anchor(name)
This method returns a **String**. The **name** parameter is of type **String**.

big()
This method returns a **String**.

blink()
This method returns a **String**.

bold()
This method returns a **String**.

fixed()
This method returns a **String**.

fontcolor(color)
This method returns a **String**. The **color** parameter is of type **String**.

fontsize(size)
This method returns a **String**. The **size** parameter is of type **String**.

italics()
This method returns a **String**.

link(href)
This method returns a **String**. The **href** parameter is of type **String**.

small()
This method returns a **String**.

strike()
This method returns a **String**.

sub()
This method returns a **String**.

sup()
This method returns a **String**.

D.2 Core Extensions Module Bindings

Exception **DOMException**

The **DOMException** exception object has the following additional properties (in addition to those derived from [DOM2] bindings):

codeExtension
This property is of type **Number**.

Class **DOMException**

The **DOMException** object class (prototype object) has the following additional properties (in addition to those derived from [DOM2] bindings):

VALIDATION_ERR

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

NO_CLOSE_ALLOWED_ERR

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

D.3 Environment Extensions Module Bindings

Object **History**

The **History** object has the following methods and properties:

length

This property is of type **Number**.

back()

This method returns a **void**.

forward()

This method returns a **void**.

go(where)

This method returns a **void**. The **where** parameter is of type **Number** or **String**.

Object **Location**

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

hash

This property is of type **String**.

host

This property is of type **String**.

hostname

This property is of type **String**.

href

This property is of type **String**.

pathname

This property is of type **String**.

port

This property is of type **String**.

protocol

This property is of type **String**.

search

This property is of type **String**.

Object **Navigator**

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

appName

This property is of type **String**.

appVersion

This property is of type **String**.

appCodeName

This property is of type **String**.

userAgent

This property is of type **String**.

ddeBackChannel

This property is of type **String**.

ddeContentLevel

This property is of type **String**.

ddeSourceId

This property is of type **String**.

ddeEnabled
This property is of type **Boolean**.
ddeReleasable
This property is of type **Boolean**.

Object **Window**
The **Window** object has the following additional methods and properties:
navigator
This property is of type **Navigator**.
history
This property is of type **History**.
window
This property is of type **Window**.
self
This property is of type **Window**.
parent
This property is of type **Window** or **void**.
top
This property is of type **Window**.
opener
This property is of type **Window**.
frames
This property is of type **HTMLCollection**.
length
This property is of type **Number**.
name
This property is of type **String**.
document
This property is of type **HTMLDocument**.
status
This property is of type **String**.
defaultStatus
This property is of type **String**.
opener
This property is of type **Window**.
alert(message)
This method returns a **void**. The **message** parameter is of type **String**.
confirm(message)
This method returns a **boolean**. The **message** parameter is of type **String**.
prompt(message, defaultResponse)
This method returns a **String**. The **message** parameter is of type **String**. The **defaultResponse** parameter is of type **String**.
setTimeout(statement, delay)
This method returns a **Number**. The **statement** parameter is of type **String**. The **delay** parameter is of type **Number**.
clearTimeout(timerId)
This method returns a **void**. The **timerId** parameter is of type **Number**.
close()
This method returns a **void**.
open(url, name, features)
This method returns a **Window** or **void**. The **url** parameter is of type **String**. The **name** parameter is of type **String**. The **features** parameter is of type **String**.

D.4 Events Extensions Module Bindings

Object **KeyEvent**

The **KeyEvent** object has the following methods and properties (in addition to those derived from the [DOM2-EVENTS] `UIEvent` interface bindings):

outputString
This property is of type **String**.

keyVal
This property is of type **Number**.

virtKeyVal
This property is of type **Number**.

visibleOutputGenerated
This property is of type **Boolean**.

numPad
This property is of type **Boolean**.

initKeyEvent(type, canBubble, cancelable, view, detail, outputString, keyVal, virtKeyVal, visibleOutputGenerated, numPad)
This method returns a **void**. The **type** and **outputString** parameters are of type **String**. The **canBubble**, **cancelable**, **visibleOutputGenerated**, and **numPad** parameters are of type **Boolean**. The **detail**, **keyVal**, and **virtKeyVal** parameters are of type **Number**. The **view** parameter is of type **AbstractView**.

initModifier(modifier, value)
This method returns a **void**. The **modifier** parameter is of type **Number**. The **value** parameter is of type **Boolean**.

checkModifier(modifier)
This method returns a **Boolean**. The **modifier** parameter is of type **Number**.

Class **KeyModifiers**

The **KeyModifiers** object class (prototype object) has the following properties:

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

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

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

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

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

Class **VirtualKeys**

The **VirtualKeys** object class (prototype object) has the following properties:

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

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

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

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

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

VK_CLEAR
This constant property is of type **Number** and its value is **12;**

VK_SHIFT
This constant property is of type **Number** and its value is **16;**

VK_CONTROL
This constant property is of type **Number** and its value is **17;**
VK_ALT
This constant property is of type **Number** and its value is **18;**
VK_PAUSE
This constant property is of type **Number** and its value is **19;**
VK_CAPS_LOCK
This constant property is of type **Number** and its value is **20;**
VK_KANA
This constant property is of type **Number** and its value is **21;**
VK_FINAL
This constant property is of type **Number** and its value is **24;**
VK_KANJI
This constant property is of type **Number** and its value is **25;**
VK_ESCAPE
This constant property is of type **Number** and its value is **27;**
VK_CONVERT
This constant property is of type **Number** and its value is **28;**
VK_NONCONVERT
This constant property is of type **Number** and its value is **29;**
VK_ACCEPT
This constant property is of type **Number** and its value is **30;**
VK_MODECHANGE
This constant property is of type **Number** and its value is **31;**
VK_SPACE
This constant property is of type **Number** and its value is **32;**
VK_PAGE_UP
This constant property is of type **Number** and its value is **33;**
VK_PAGE_DOWN
This constant property is of type **Number** and its value is **34;**
VK_END
This constant property is of type **Number** and its value is **35;**
VK_HOME
This constant property is of type **Number** and its value is **36;**
VK_LEFT
This constant property is of type **Number** and its value is **37;**
VK_UP
This constant property is of type **Number** and its value is **38;**
VK_RIGHT
This constant property is of type **Number** and its value is **39;**
VK_DOWN
This constant property is of type **Number** and its value is **40;**
VK_COMMA
This constant property is of type **Number** and its value is **44;**
VK_PERIOD
This constant property is of type **Number** and its value is **46;**
VK_SLASH
This constant property is of type **Number** and its value is **47;**
VK_0
This constant property is of type **Number** and its value is **48;**
VK_1
This constant property is of type **Number** and its value is **49;**
VK_2
This constant property is of type **Number** and its value is **50;**
VK_3
This constant property is of type **Number** and its value is **51;**

VK_4
This constant property is of type **Number** and its value is **52;**
VK_5
This constant property is of type **Number** and its value is **53;**
VK_6
This constant property is of type **Number** and its value is **54;**
VK_7
This constant property is of type **Number** and its value is **55;**
VK_8
This constant property is of type **Number** and its value is **56;**
VK_9
This constant property is of type **Number** and its value is **57;**
VK_SEMICOLON
This constant property is of type **Number** and its value is **59;**
VK_EQUALS
This constant property is of type **Number** and its value is **61;**
VK_A
This constant property is of type **Number** and its value is **65;**
VK_B
This constant property is of type **Number** and its value is **66;**
VK_C
This constant property is of type **Number** and its value is **67;**
VK_D
This constant property is of type **Number** and its value is **68;**
VK_E
This constant property is of type **Number** and its value is **69;**
VK_F
This constant property is of type **Number** and its value is **70;**
VK_G
This constant property is of type **Number** and its value is **71;**
VK_H
This constant property is of type **Number** and its value is **72;**
VK_I
This constant property is of type **Number** and its value is **73;**
VK_J
This constant property is of type **Number** and its value is **74;**
VK_K
This constant property is of type **Number** and its value is **75;**
VK_L
This constant property is of type **Number** and its value is **76;**
VK_M
This constant property is of type **Number** and its value is **77;**
VK_N
This constant property is of type **Number** and its value is **78;**
VK_O
This constant property is of type **Number** and its value is **79;**
VK_P
This constant property is of type **Number** and its value is **80;**
VK_Q
This constant property is of type **Number** and its value is **81;**
VK_R
This constant property is of type **Number** and its value is **82;**
VK_S
This constant property is of type **Number** and its value is **83;**
VK_T
This constant property is of type **Number** and its value is **84;**

VK_U
This constant property is of type **Number** and its value is **85;**
VK_V
This constant property is of type **Number** and its value is **86;**
VK_W
This constant property is of type **Number** and its value is **87;**
VK_X
This constant property is of type **Number** and its value is **88;**
VK_Y
This constant property is of type **Number** and its value is **89;**
VK_Z
This constant property is of type **Number** and its value is **90;**
VK_OPEN_BRACKET
This constant property is of type **Number** and its value is **91;**
VK_BACK_SLASH
This constant property is of type **Number** and its value is **92;**
VK_CLOSE_BRACKET
This constant property is of type **Number** and its value is **93;**
VK_NUMPAD0
This constant property is of type **Number** and its value is **96;**
VK_NUMPAD1
This constant property is of type **Number** and its value is **97;**
VK_NUMPAD2
This constant property is of type **Number** and its value is **98;**
VK_NUMPAD3
This constant property is of type **Number** and its value is **99;**
VK_NUMPAD4
This constant property is of type **Number** and its value is **100;**
VK_NUMPAD5
This constant property is of type **Number** and its value is **101;**
VK_NUMPAD6
This constant property is of type **Number** and its value is **102;**
VK_NUMPAD7
This constant property is of type **Number** and its value is **103;**
VK_NUMPAD8
This constant property is of type **Number** and its value is **104;**
VK_NUMPAD9
This constant property is of type **Number** and its value is **105;**
VK_MULTIPLY
This constant property is of type **Number** and its value is **106;**
VK_ADD
This constant property is of type **Number** and its value is **107;**
VK_SEPARATER
This constant property is of type **Number** and its value is **108;**
VK_SUBTRACT
This constant property is of type **Number** and its value is **109;**
VK_DECIMAL
This constant property is of type **Number** and its value is **110;**
VK_DIVIDE
This constant property is of type **Number** and its value is **111;**
VK_F1
This constant property is of type **Number** and its value is **112;**
VK_F2
This constant property is of type **Number** and its value is **113;**
VK_F3
This constant property is of type **Number** and its value is **114;**

VK_F4
This constant property is of type **Number** and its value is **115;**
VK_F5
This constant property is of type **Number** and its value is **116;**
VK_F6
This constant property is of type **Number** and its value is **117;**
VK_F7
This constant property is of type **Number** and its value is **118;**
VK_F8
This constant property is of type **Number** and its value is **119;**
VK_F9
This constant property is of type **Number** and its value is **120;**
VK_F10
This constant property is of type **Number** and its value is **121;**
VK_F11
This constant property is of type **Number** and its value is **122;**
VK_F12
This constant property is of type **Number** and its value is **123;**
VK_DELETE
This constant property is of type **Number** and its value is **127;**
VK_NUM_LOCK
This constant property is of type **Number** and its value is **144;**
VK_SCROLL_LOCK
This constant property is of type **Number** and its value is **145;**
VK_PRINTSCREEN
This constant property is of type **Number** and its value is **154;**
VK_INSERT
This constant property is of type **Number** and its value is **155;**
VK_HELP
This constant property is of type **Number** and its value is **156;**
VK_META
This constant property is of type **Number** and its value is **157;**
VK_BACK_QUOTE
This constant property is of type **Number** and its value is **192;**
VK_QUOTE
This constant property is of type **Number** and its value is **222;**
VK_COLORED_KEY_0
This constant property is of type **Number** and its value is **403;**
VK_COLORED_KEY_1
This constant property is of type **Number** and its value is **404;**
VK_COLORED_KEY_2
This constant property is of type **Number** and its value is **405;**
VK_COLORED_KEY_3
This constant property is of type **Number** and its value is **406;**
VK_COLORED_KEY_4
This constant property is of type **Number** and its value is **407;**
VK_COLORED_KEY_5
This constant property is of type **Number** and its value is **408;**
VK_POWER
This constant property is of type **Number** and its value is **409;**
VK_DIMMER
This constant property is of type **Number** and its value is **410;**
VK_WINK
This constant property is of type **Number** and its value is **411;**
VK_REWIND
This constant property is of type **Number** and its value is **412;**

VK_STOP
This constant property is of type **Number** and its value is **413;**
VK_EJECT_TOGGLE
This constant property is of type **Number** and its value is **414;**
VK_PLAY
This constant property is of type **Number** and its value is **415;**
VK_RECORD
This constant property is of type **Number** and its value is **416;**
VK_FAST_FWD
This constant property is of type **Number** and its value is **417;**
VK_PLAY_SPEED_UP
This constant property is of type **Number** and its value is **418;**
VK_PLAY_SPEED_DOWN
This constant property is of type **Number** and its value is **419;**
VK_PLAY_SPEED_RESET
This constant property is of type **Number** and its value is **420;**
VK_RECORD_SPEED_NEXT
This constant property is of type **Number** and its value is **421;**
VK_GO_TO_START
This constant property is of type **Number** and its value is **422;**
VK_GO_TO_END
This constant property is of type **Number** and its value is **423;**
VK_TRACK_PREV
This constant property is of type **Number** and its value is **424;**
VK_TRACK_NEXT
This constant property is of type **Number** and its value is **425;**
VK_RANDOM_TOGGLE
This constant property is of type **Number** and its value is **426;**
VK_CHANNEL_UP
This constant property is of type **Number** and its value is **427;**
VK_CHANNEL_DOWN
This constant property is of type **Number** and its value is **428;**
VK_STORE_FAVORITE_0
This constant property is of type **Number** and its value is **429;**
VK_STORE_FAVORITE_1
This constant property is of type **Number** and its value is **430;**
VK_STORE_FAVORITE_2
This constant property is of type **Number** and its value is **431;**
VK_STORE_FAVORITE_3
This constant property is of type **Number** and its value is **432;**
VK_RECALL_FAVORITE_0
This constant property is of type **Number** and its value is **433;**
VK_RECALL_FAVORITE_1
This constant property is of type **Number** and its value is **434;**
VK_RECALL_FAVORITE_2
This constant property is of type **Number** and its value is **435;**
VK_RECALL_FAVORITE_3
This constant property is of type **Number** and its value is **436;**
VK_CLEAR_FAVORITE_0
This constant property is of type **Number** and its value is **437;**
VK_CLEAR_FAVORITE_1
This constant property is of type **Number** and its value is **438;**
VK_CLEAR_FAVORITE_2
This constant property is of type **Number** and its value is **439;**
VK_CLEAR_FAVORITE_3
This constant property is of type **Number** and its value is **440;**

VK_SCAN_CHANNELS_TOGGLE
This constant property is of type **Number** and its value is **441;**
VK_PINP_TOGGLE
This constant property is of type **Number** and its value is **442;**
VK_SPLIT_SCREEN_TOGGLE
This constant property is of type **Number** and its value is **443;**
VK_DISPLAY_SWAP
This constant property is of type **Number** and its value is **444;**
VK_SCREEN_MODE_NEXT
This constant property is of type **Number** and its value is **445;**
VK_VIDEO_MODE_NEXT
This constant property is of type **Number** and its value is **446;**
VK_VOLUME_UP
This constant property is of type **Number** and its value is **447;**
VK_VOLUME_DOWN
This constant property is of type **Number** and its value is **448;**
VK_MUTE
This constant property is of type **Number** and its value is **449;**
VK_SURROUND_MODE_NEXT
This constant property is of type **Number** and its value is **450;**
VK_BALANCE_RIGHT
This constant property is of type **Number** and its value is **451;**
VK_BALANCE_LEFT
This constant property is of type **Number** and its value is **452;**
VK_FADER_FRONT
This constant property is of type **Number** and its value is **453;**
VK_FADER_REAR
This constant property is of type **Number** and its value is **454;**
VK_BASS_BOOST_UP
This constant property is of type **Number** and its value is **455;**
VK_BASS_BOOST_DOWN
This constant property is of type **Number** and its value is **456;**
VK_INFO
This constant property is of type **Number** and its value is **457;**
VK_GUIDE
This constant property is of type **Number** and its value is **458;**
VK_TELETEXT
This constant property is of type **Number** and its value is **459;**
VK_SUBTITLE
This constant property is of type **Number** and its value is **460;**

D.5 HTML Extensions Module Bindings

Object **HTMLAnchorElement**

The **HTMLAnchorElement** object has the following additional properties (in addition to those derived from [DOM2] bindings):

hash
This property is of type **String**.
host
This property is of type **String**.
hostname
This property is of type **String**.
pathname
This property is of type **String**.
port
This property is of type **String**.

protocol
This property is of type **String**.
search
This property is of type **String**.

Object **HTMLDocumentElement**
The **HTMLDocumentElement** object has the following additional methods and properties (in addition to those derived from [DOM2] bindings):
location
This property is of type **String**.
lastModified
This property is of type **String**.
alinkColor
This property is of type **String**.
vlinkColor
This property is of type **String**.
linkColor
This property is of type **String**.
bgColor
This property is of type **String**.
fgColor
This property is of type **String**.
clear()
This method returns a **void**.

Object **HTMLFormElement**
The **HTMLFormElement** object has the following additional properties (in addition to those derived from [DOM2] bindings):
encoding
This property is of type **String**.

Object **HTMLImageElement**
The **HTMLImageElement** object has the following additional methods and properties (in addition to those derived from [DOM2] bindings):
complete
This property is of type **Boolean**.
lowsrc
This property is of type **String**.

Object **HTMLObjectElement**
The **HTMLObjectElement** object has the following additional methods and properties (in addition to those derived from [DOM2] bindings):
complete
This property is of type **Boolean**.
lowsrc
This property is of type **String**.
src
This property is of type **String**.
The **HTMLObjectElement** object has the following additional properties in certain cases described in Section 5.3.1.2.3.7:
backChannel
This property is of type **String**.
contentLevel
This property is of type **String**.
sourceId
This property is of type **String**.
enabled
This property is of type **Boolean**.

releasable
This property is of type **Boolean**.

D.6 Views Extensions Module Bindings

Object **DocumentView**

The **DocumentView** object has the following additional properties (in addition to those derived from [DOM2] bindings):

refWidthMm
This property is of type **Number**.
refHeightMm
This property is of type **Number**.
gfxPosX
This property is of type **Number**.
gfxPosY
This property is of type **Number**.
gfxWidth
This property is of type **Number**.
gfxHeight
This property is of type **Number**.
gfxWidthPx
This property is of type **Number**.
gfxHeightPx
This property is of type **Number**.
refreshOnChange
This property is of type **Boolean**.
sampleBitsR
This property is of type **Number**.
sampleBitsG
This property is of type **Number**.
sampleBitsB
This property is of type **Number**.
sampleBitsA
This property is of type **Number**.
vidPosX
This property is of type **Number**.
vidPosY
This property is of type **Number**.
vidWidth
This property is of type **Number**.
vidHeight
This property is of type **Number**.
vidWidthPx
This property is of type **Number**.
vidHeightPx
This property is of type **Number**.
refresh(id)
This method returns a **boolean**. The **id** parameter is of type **String**.

ANNEX E. UNSUPPORTED HTML FEATURES

The entirety of this section and its subsections is informative.

This annex describes features of [HTML] which are only partially supported or are entirely unsupported by this specification. These features are divided into syntactic and semantic features. Syntactic features pertain to the lexical grammar of [HTML], while semantic features pertain to the vocabulary of [HTML]; i.e., the collection of elements and attributes defined by [HTML] and their implicit or explicit meaning.

> *Note:* Features of [HTML] which are not directly supported by this specification, but are indirectly supported through transcoding are not specified in this annex. See ANNEX F, Transcoding HTML to XDML, for further information on transcoding of [HTML] features.

E.1 Syntax

This section describes unsupported syntactic features of [HTML].

E.1.1 Case Insensitivity

The syntax of [HTML] is derived from [SGML], which specifies that the names of elements and attributes are not sensitive to distinctions in case (i.e., uppercase and lowercase name characters are interpreted as identical for the purpose of name matching). This case insensitivity is not supported by `application/xhtml+xml` content, being derived from [XML] which is case sensitive in all names.

E.1.2 Markup Minimization

The syntax of [HTML] is derived from [SGML], which provides a syntactic feature known as the *(markup) minimization* feature. Two specific types of the minimization feature, OMITTAG and SHORTTAG, are partially supported by [HTML] but not by [XML], from which the markup content defined of this specification is derived.

E.1.2.1 OMITTAG

The OMITTAG feature permits the elision of either the start or end tag of elements in certain contexts. This feature, fully supported by [HTML] according to the specific element declarations of the [HTML] document types, is not supported by this specification.

E.1.2.1.1 Omitted Start Tag

The OMITTAG feature permits omission of a start tag if the element is a contextually required element and any other elements that could occur are contextually optional elements, except if the element type has a required attribute or the content of the instance of the element is empty. See [SGML], Clause 7.3.1.1. This omission is not supported by this specification.

Example: The following shows the omission permissible by [HTML] and the equivalent, more constrained syntax required by [XML] as expressed in `application/xhtml+xml` content. Note that in the text/html example, the start tags of the *html*, *head*, and *body* elements are missing even though the end tags are present.

text/html:

```
<title>Title</title>
</head>
<p>Some Text</p>
</body>
</html>
```

application/xhtml+xml:

```
<html>
<head>
<title>Title</title>
</head>
<body>
<p>Some Text</p>
</body>
</html>
```

E.1.2.1.2 Omitted End Tag

The OMITTAG feature permits omission of an end tag if the element is followed by the end of the document entity, the end tag of another open element, or by an element or SGML character that is not allowed in its content. See [SGML], Clause 7.3.1.2. This omission is not supported by this specification.

Example: The following shows the omission permissible by [HTML] and the equivalent, more constrained syntax required by [XML] as expressed in `application/xhtml+xml` content. Note that in the `text/html` example, the end tags of *html*, *head*, *body*, and *p* elements are missing even though the start tags are present. The appearance of the *p* element's start tag automatically closes the *head* element. The appearance of the end of the document entity automatically closes the *p*, *body*, and *html* elements.

text/html:

```
<html>
<head>
<title>Title</title>
<body>
<p>Some Text
```

application/xhtml+xml:

```
<html>
<head>
<title>Title</title>
</head>
<body>
<p>Some Text</p>
</body>
</html>
```

E.1.2.2 SHORTTAG

The SHORTTAG feature permits the elision of parts of attribute specifications in certain contexts. This feature, partially supported by [HTML] according to the specific attribute declarations of the [HTML] document types, is not supported by this specification.

E.1.2.2.1 Omitted Attribute Name

The SHORTTAG feature permits omission of an attribute name from an attribute specification if the attribute value specification is an undelimited name token that is a member of a group specified in the declared value for that attribute. See [SGML], Clause 7.9.1.2. This omission is not supported by this specification.

Example: The following shows the omission permissible by [HTML] and the equivalent, more constrained syntax required by [XML] as expressed in `application/xhtml+xml` content.

text/html:

```
<object declare>
```

application/xhtml+xml:

```
<object declare="declare">
```

E.1.2.2.2 Omitted Attribute Value Literal Delimiters

The SHORTTAG feature permits omission of delimiters from an attribute value literal if the attribute value contains nothing but name characters. See [SGML], Clause 7.9.3.1. This omission is not supported by this specification.

Example: The following shows the omission permissible by [HTML] and the equivalent, more constrained syntax required by [XML] as expressed in `application/xhtml+xml` content.

text/html:

```
<object data=obj.png>
```

application/xhtml+xml:

```
<object data="obj.png">
```

E.1.3 Reference End

The syntax of [HTML], being derived from [SGML], permits a character reference to omit the reference end (either *refc* or `RE`) if the reference is not followed by a character that could be interpreted as the omitted reference end. See [SGML], Clause 9.4.5. This omission is not supported by this specification.

Example: The following shows the omission permissible by [HTML] and the equivalent, more constrained syntax required by [XML] as expressed in `application/xhtml+xml` content. In the following, the character reference found in the *content* attribute value has omitted the reference close (*refc*) delimiter `';'` since the closing attribute value literal delimiter (*lit*) `'"'` cannot be part of the character reference. In contrast, the delimiter must be present in `application/xhtml+xml` content.

text/html:

```
<meta name="eg" content="&lt">
```

application/xhtml+xml:

```
<meta name="eg" content="&lt;"/>
```

E.2 Semantics

This section describes constraints on partially supported attributes and elements as well as entirely unsupported attributes and elements.

E.2.1 Constrained Attribute Interpretation

The semantics of the following attributes are constrained. A consequence of this lack of support is that the information expressed by these attributes will be lost in certain well-defined cases when performing round-trip transcoding from [HTML] to XDML and back to [HTML].

> *Note:* Notwithstanding the above, it is possible to retain this information through a round-trip mapping by the use of either structured comments or processing instructions. This requires (1) that no subsequent process remove these comments or processing instructions prior to reverse transcoding and (2) that the reverse transcoding process be aware of the precise nature of this mapping process. This specification does not define a standard form for such a mapping.

E.2.1.1 *alt* attribute

The *alt* attribute on the *applet* element is not supported if the *applet* element has non-whitespace `#PCDATA` content or has child elements which are not *param* elements.

E.2.2 Unsupported Attributes

The semantics of the following attributes is not supported. A consequence of this lack of support is that the information expressed by these attributes will be lost when performing round-trip transcoding from [HTML] to XDML and back to [HTML].

> *Note:* Notwithstanding the above, it is possible to retain this information through a round-trip mapping by the use of either structured comments or processing instructions. This requires (1) that no subsequent process remove these comments or processing instructions prior to reverse transcoding and (2) that the reverse transcoding process be aware of the precise nature of this mapping process. This specification does not define a standard form for such a mapping.

E.2.2.1 *cite* attribute

The *cite* attribute on *del* and *ins* elements is not supported.

E.2.2.2 *compact* attribute

The *compact* attribute on *dir*, *dl*, *menu*, *ol*, and *ul* elements is not supported.

E.2.2.3 *datetime* attribute

The *datetime* attribute on *del* and *ins* elements is not supported.

E.2.2.4 *ismap* attribute

The *ismap* attribute on *img* and *input* elements is not supported.

E.2.3 Constrained Element Interpretation

The semantics of the following elements are constrained. A consequence of this lack of support is that the information expressed by these elements will be lost in certain well-defined cases when performing round-trip transcoding from [HTML] to XDML and back to [HTML].

> *Note:* Notwithstanding the above, it is possible to retain this information through a round-trip mapping by the use of either structured comments or processing instructions. This requires (1) that no subsequent process remove these comments or processing instructions prior to reverse transcoding and (2) that the reverse transcoding process be aware of the precise nature of this mapping process. This specification does not define a standard form for such a mapping.

E.2.3.1 *basefont* element

The *basefont* element is not supported in any context other than as a child of the *body* element.

E.2.3.2 *input* element

The *input* element is not supported when the value of the *type* attribute is `file`.

E.2.3.3 *isindex* element

The *isindex* element is not supported as a child of the *head* element.

E.2.3.4 *noframes* element

The *noframes* element is not supported in any context other than as a child of the *frameset* element.

E.2.3.5 *script* element

The *script* element is not supported when the value of the *type* attribute is not `text/ecmascript`.

E.2.3.6 *style* element

The *style* element is not supported when the value of the *type* attribute is not `text/css`.

E.2.4 Unsupported Elements

The semantics of the following elements is not supported. A consequence of this lack of support is that the information expressed by these elements will be lost when performing round-trip transcoding from [HTML] to XDML and back to [HTML].

> *Note:* Notwithstanding the above, it is possible to retain this information through a round-trip mapping by the use of either structured comments or processing instructions. This requires (1) that no subsequent process remove these comments or processing instructions prior to reverse transcoding and (2) that the reverse transcoding process be aware of the precise nature of this mapping process. This specification does not define a standard form for such a mapping.

E.2.4.1 *iframe* element

The functionality of the *iframe* element is not supported. The content of the *iframe* element is retained as alternate content to be presented to the end-user.

ANNEX F. TRANSCODING HTML TO XDML

The entirety of this section and its subsections is informative.

This annex describes a simple process by means of which legacy content based on [HTML] may be transcoded into XDML, the XHTML-based markup language content defined by this specification.

The process described below is one of a number of possible processes which support transcoding of [HTML] into XDML. This process is intentionally made simple in order to support real-time transcoding prior to transmission. In particular, it does not require multiple passes or the generation of a full, in-memory representation of the document. Furthermore, it does not require look ahead by the source parser.

The transcoding process consists of the following steps, which consist of two pre-processing steps (1-2) and one primary mapping step (3):

(1) validate source document;
(2) if document is invalid, apply fix-ups then repeat step (1);
(3) for each comment, processing instruction, document type declaration, start tag, end tag, named character reference, numeric character reference, marked section, and contiguous sequence of #PCDATA content parsed in the valid source document, output a result string to the target document as described in the following subsections; while performing this step, maintain a stack of open elements in order to (a) imply start tags omitted from the source document; (b) imply end tags omitted from the source document; and (c) provide ancestor context in order to effect certain contextual constraints described in ANNEX E.

If the source document is known *a priori* to be valid, then pre-processing steps (1) and (2) can be omitted.

> *Note:* In the following subsections, the transcoding of certain presentation-oriented markup is accomplished by mapping this information to style properties expressed with the inline *style* attribute. A transcoding process may wish to coalesce all such generated *style* attributes into a single stylesheet embedded within the generated, target document, using either an *id* attribute or a *class* attribute to associate an element with its corresponding stylesheet rule.

> *Note:* When generating style properties, a transcoder should take into account the potential for conflicts of cascading order between synthesized styles specified by use of the inline *style* attribute and styles specified using style rules which appear in the source document.

F.1 Syntax

The syntax of HTML is based on [SGML], whereas the syntax of XDML is based on [XHTMLMOD] a reformulation of HTML using [XML]. The syntax of an XDML document instance is a subset of the syntax of an HTML document instance. In order to transcode HTML source documents into XDML target documents, it is necessary to account for certain differences in the syntax permitted by [XML] from that permitted by [SGML]. The following subsections describe how to transcode syntactic elements between these two syntaxes.

F.1.1 Character Encodings

If the character encoding system of the source document is ISO-8859-1, then map to a target document using ISO-8859-1; otherwise, map to a target document using UTF-8, performing character transcoding as required.

F.1.2 Comments

With the exception of comments in *script* and *style* elements, map comments and their content to the null string. See Sections F.2.2.70 and F.2.2.76 for treatment of comments in *script* and *style* elements.

F.1.3 Processing Instructions

Map processing instructions and their content to the null string.

F.1.4 Document Type Declaration

When transcoding a document as opposed to a document fragment, map the document type declaration to a valid XDML document type declaration according to Section 5.1.1.4. In particular, map the public and system identifiers to valid target identifiers and map the internal declaration subset, if present, to a null string. When transcoding a document fragment, do not generate a document type declaration in the target fragment.

In the case that the source document specifies the strict, frameset, or transitional document types of [HTML], the target document uses the XDML document type.

> *Note:* Although [SGML] permits an SGML application, such as [HTML], to specify an internal declaration subset, [HTML] itself does not explicitly support the use of an internal declaration subset. Since very few HTML user agents support the use of an internal declaration subset, it is safe to ignore such usage.

F.1.5 Elements

Map the start tag of elements declared to have empty content to an empty element tag [XML:44]. Map implicit or explicit start and end tags of elements not declared to have empty content to valid XML start and end tags [XML:40 and XML:42].

F.1.6 Character References

Map named character references directly: all named character references defined by [HTML] are also present in XDML. Map numeric character references directly only if it can be ascertained that the source document uses character numbers which adhere to the document character set of [HTML]; otherwise, map numeric character references to a substitution character, e.g., `'?'`.

If a source character reference omits the reference end, then a *refc* delimiter `';'` is to be supplied in the target character reference. See Section E.1.3.

F.1.7 Marked Sections

If an included marked section appears in the source document, then map the marked section's delimiters to the null string, transcoding the content of the marked section. If an ignored marked section appears in the source document, then map the marked section and its contents to the null string. If a `CDATA` marked section appears in the source document, then map the marked section (delimiters and content) to the target document without alteration; that is, do not perform transcoding on the marked section's contents. If an `RCDATA` marked section appears in the source document, then map the marked section (delimiters and content) to a `CDATA` marked section in the target document while transcoding any embedded character references.

> *Note:* An included marked section is a marked section whose effective status keyword is `INCLUDE` or `TEMP`. An ignored marked section is a marked section whose effective status keyword is `IGNORE`. A `CDATA` marked section is a marked section whose effective status keyword is `CDATA`. An `RCDATA` marked section is a marked section whose effective status keyword is `RCDATA`.

F.2 Semantics

The semantics of [HTML] are determined by its vocabulary. This vocabulary comprises a collection of common attributes, which apply to the majority of elements, and a collection of elements. The vocabulary of XDML, the target content type for this transcoding process, is, at present, a subset of the [HTML] vocabulary, being based on [XHTMLMOD], which ultimately derives its vocabulary from [HTML]. Being a subset of the [HTML] vocabulary makes it relatively straightforward to map the semantics of [HTML] to XDML. The instances where this mapping is not trivial derive from the following: (1) the subset nature of XDML, and (2) the use of an XML based markup language (XHTML) rather than an SGML based markup language.

F.2.1 Common Attributes

A majority of elements defined by [HTML] permit the use of a collection of common attributes. Mappings for these attributes are described in the following subsections.

Note: Only the *lang* common attribute requires special treatment during transcoding.

F.2.1.1 *class* attribute

No special treatment is required; an identity mapping applies.

F.2.1.2 *dir* attribute

No special treatment is required; an identity mapping applies.

F.2.1.3 *id* attribute

No special treatment is required; an identity mapping applies.

F.2.1.4 *lang* attribute

Map to the *xml:lang* attribute, the value remaining unchanged.

F.2.1.5 *on{click,dblclick,mouse*,key*}* attributes

No special treatment is required; an identity mapping applies.

F.2.1.6 *style* attribute

No special treatment is required; an identity mapping applies.

F.2.1.7 *title* attribute

No special treatment is required; an identity mapping applies.

F.2.2 Elements

The following subsections specify mappings for each of the ninety-one (91) elements defined by [HTML].

F.2.2.1 *a* element

Except for the *href* and common attributes, no special treatment is required; an identity mapping applies.

If the URI value of the *href* attribute employs the `"http:"` URI scheme, then map the scheme token to `"lid:"` and repackage the referenced resource as a local resource. If the URI value employs the `"javascript:"` URI scheme, then map the scheme token to `"ecmascript:"`. Otherwise, an identity mapping applies for the attribute value.

F.2.2.2 *abbr* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.3 *acronym* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.4 *address* element

With the exception of permissible content and common attributes, no special treatment is required; an identity mapping applies. In regards to permissible content, if a *p* element appears in the content of an *address* element in the source document, then each such *p* element is to be mapped to a *span* element with block display semantics, where the *span* element's content is the content of the *p* element.

Block display semantics on a *span* element are obtained by using an inline *style* attribute with the value `"display: block"`.

If an *align* attribute appears on a *p* element which is mapped to a *span* element according to the above process, then the *align* attribute is to be mapped to the inline *style* attribute on the *span* element according to the procedure specified in Section F.2.2.64.

F.2.2.5 *applet* element

Map the *applet* element to the *object* element, and, with the exception of *align*, *alt*, *code*, *hspace*, *object*, *vspace*, and common attributes, no special treatment is required; an identity mapping applies.

Map the *align* attribute, if present and a valid, non-empty value, to an inline *style* attribute as follows: (1) if an *align* attribute is present and its *value* is one of `left | right`, then map to an inline *style* attribute with the value `"float: <value>"`, where `<value>` is identical to the value of the *align* attribute; or (2) if an *align* attribute is present and its *value* is `bottom | middle | top`, then map to an inline *style* attribute with the value `"vertical-align: <value>"`, where `<value>` is identical to the value of the *align* attribute.

Map the *alt* attribute, if present, to either (1) `#PCDATA` content appearing as a child of the target *object* element or (2) to the *title* attribute. If the source *applet* element has no non-whitespace `#PCDATA` content and no child elements or if all child elements are *param* elements, then the value of the *alt* attribute is to be mapped to `#PCDATA` content appearing as a child of the target *object* element. Otherwise, if the source applet element has non-whitespace `#PCDATA` content or has child elements which are not *param* elements, then map the *alt* attribute to the *title* attribute, if no *title* attribute is present, or to a prefix or suffix of the value of the *title* attribute, if a *title* attribute is present on the source *applet* element.

Map the *code* attribute, if present, to a *classid* attribute, where the value of the target *object* element's *classid* attribute is identical to the value of the source *applet* element's *code* attribute. If the URI value of the *classid* attribute employs the `"http:"` URI scheme, then map the scheme token to `"lid:"` and repackage the referenced resource as a local resource.

Map the *hspace* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value `"margin-left: <value>px; margin-right: <value>px"`, where `<value>` is identical to the value of the *hspace* attribute.

Map the *object* attribute, if present, to a *data* attribute, where the value of the target *object* element's *data* attribute is identical to the value of the source *applet* element's *object* attribute.

Map the *vspace* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value `"margin-top: <value>px; margin-bottom: <value>px"`, where `<value>` is identical to the value of the *vspace* attribute.

If the above mappings would cause the creation of multiple inline *style* attributes, then the values of these multiple attributes are to be collected and expressed in a single inline style attribute.

F.2.2.6 *area* element

Except for the *href* and common attributes, no special treatment is required; an identity mapping applies.

If the URI value of the *href* attribute employs the `"http:"` URI scheme, then map the scheme token to `"lid:"` and repackage the referenced resource as a local resource. If the URI value employs the `"javascript:"` URI scheme, then map the scheme token to `"ecmascript:"`. Otherwise, an identity mapping applies for the attribute value.

F.2.2.7 *b* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.8 *base* element

Map the *base* element to an *xml:base* attribute, as described in [XMLBASE], on all subsequent elements of the *head* element (i.e., the parent element of the base element) and on the document's sole *body* or *frameset* element.

If the URI value of the *href* attribute employs the `"http:"` URI scheme, then map the scheme token to `"lid:"`. Otherwise, an identity mapping applies for the attribute value.

F.2.2.9 *basefont* element

If a *basefont* element appears as a child of a *body* element in the source document, then map the *basefont* element to a *div* element with an inline *style* attribute which expresses (1) inline display semantics; (2) the value of the required *size* attribute; and (3) the values of the optional *color* and *face* attributes, if present. If a *basefont* element appears anywhere other than as a child of the *body* element, then map the *basefont* element to an empty string.

Inline display semantics are obtained by using an inline *style* attribute with the value `"display: inline"`.

Map the *size* attribute to an inline *style* attribute with the value `"font-size: absolute size"`, where `absolute size` is determined from the value of the *size* attribute according to Table 7 Legacy Font Size Attribute Mappings:.

Map the *color* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value `"color: color"`, where `color` is identical to the value of the *color* attribute.

Map the *face* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value `"font-family: family"`, where `family` is identical to the value of the *face* attribute, except that, if the value of the *face* attribute contains any whitespace characters, then *family* is to be further quoted with single-quote (`'`) characters or with an equivalent entity or character reference, i.e., `'` or `'`.

If the above mappings would cause the creation of multiple inline *style* attributes, then the values of these multiple attributes are to be collected and expressed in a single inline style attribute.

If a *basefont* element is mapped to a *div* element as described above, the start tag of the *div* element replaces the *basefont* element (which has empty content, and, therefore, has only a start tag and no end tag). The end tag of the *div* element is then inserted immediately prior to the target mapping of the next sibling basefont element, if any exists; otherwise, it is inserted immediately prior to the end tag of the enclosing *body* element.

If multiple *basefont* elements are mapped and those elements contain the optional *color* or *face* attributes, then the values of these attributes are to be repeatedly mapped on subsequent *basefont* element mappings unless the subsequent element specifies a new value for one of these optional attributes. If a new value for one of these optional attributes is empty or is otherwise invalid, then do not repeatedly map that optional attribute mapping; instead, reset any repeated mapping of that optional attribute (i.e., discontinue repeated mapping for that attribute).

> *Note:* The transcoding process described above may be implemented without requiring either look ahead or a tree representation with full nesting context. Three state variables suffice for implementing this algorithm: (1) a boolean valued variable indicating whether the immediate parent element is a *body* element, (2) a string valued variable containing an empty string or the value of a *color* attribute of the most recently mapped *basefont* element which specified a *color* attribute, and (3) a string valued variable containing an empty string or the value of a *face* attribute of the most recently mapped *basefont* element which specified a *face* attribute.
>
> *Note:* By using a less simplistic transcoding process, it is possible to eliminate the restriction of mapping only those *basefont* elements that appear as children of a *body* element. In particular, by creating a full tree representation of the source document, it can be determined which elements would be affected by the semantics of *basefont* elements independently of where the affected elements or the *basefont* elements appear in the tree. Once the affected element set is determined, then an inline *style* attribute that reflects *basefont* semantics can be instantiated on only those elements without introducing *div* elements as placeholders for inheritable style properties.

Example: The following depicts a source and target fragment after performing the mappings described above. Note that this example shows the repeated use of the optional *color* and *face* attribute mappings as described in the previous paragraph.

Source Fragment:

```
<body>
<p>text1</p>
<basefont size="1">
<p>text2</p>
<basefont size="1" color="red">
<p>text3</p>
<basefont size="3" face="Arial Bold">
<p>text4</p>
<basefont size="4" color="blue" face="">
<p>text5</p>
</body>
```

Target Fragment:

```
<body>
<p>text1</p>
<!-- map 1st basefont -->
<div style="display: inline; font-size: xx-small">
<p>text2</p>
</div>
<!-- map 2nd basefont -->
<div style="display: inline; font-size: xx-small; color: red">
<p>text3</p>
</div>
<!-- map 3rd basefont -->
<div style="display: inline; font-size: medium; color: red; font-family: 'Arial Bold'">
<p>text4</p>
</div>
<!-- map 4th basefont -->
<div style="display: inline; font-size: large; color: blue">
<p>text5</p>
</div>
</body>
```

F.2.2.10 *bdo* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.11 *big* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.12 *blockquote* element

Except for the *cite* and common attributes, no special treatment is required; an identity mapping applies.

If the URI value of the *cite* attribute employs the `"http:"` URI scheme, then map the scheme token to `"lid:"` and repackage the referenced resource as a local resource.

F.2.2.13 *body* element

With the exception of permissible content, *alink*, *background*, *bgcolor*, *link*, *text*, *vlink*, and common attributes, no special treatment is required; an identity mapping applies. In regards to permissible content, if `%Inline.mix;` content or non-whitespace `#PCDATA` appears in the content of a *body* element in the source document, then one or more *div* elements with inline display semantics is to be inserted in the target document, with each *div* element's content being the original `%Inline.mix;` element content or the non-whitespace `#PCDATA` content.

Inline display semantics on a *div* element are obtained by using an inline *style* attribute with the value `"display: inline"`.

If non-whitespace `#PCDATA` is enclosed in a *div* element as described above, then any whitespace `#PCDATA` that appears at the periphery (boundaries) of this non-whitespace `#PCDATA` is also enclosed by the *div* element.

If multiple, contiguous sibling *div* elements would be inserted by following the process described above, then these sibling elements may be collapsed into a single *div* element. In this case, any whitespace `#PCDATA` which appears between children of the *blockquote* element in the source document is also to appear in the content of the single target *div* element. Use of a single target *div* element is preferred over multiple contiguous target *div* elements.

See Section A.1.2 for the definition of `%Inline.mix;` content.

Map the *alink* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value `"color: <value>"`, where `<value>` is identical to the value of the *alink* attribute. This value is to be expressed within a declarations non-terminal enclosed in curly braces within two rules with the following selectors: (1) an `a:link:active` dynamic pseudo-class selector and (2) an `a:visited:active` dynamic pseudo-class selector, with both rules being expressed according to the syntax specified in Section 5.2.1.9.

Map the *background* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value `"background-image: url(<value>)"`, where `<value>` is identical to the value of the *background* attribute. If an *alink*, *link*, or *vlink* attribute is present in the source element, then this value is to be expressed within a declarations non-terminal enclosed in curly braces without a selector according to the syntax specified in Section 5.2.1.9.

Map the *bgcolor* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value `"background-color: <value>"`, where `<value>` is identical to the value of the *bgcolor* attribute. If an *alink*, *link*, or *vlink* attribute is present in the source element, then this value is to be expressed within a declarations non-terminal enclosed in curly braces without a selector according to the syntax specified in Section 5.2.1.9.

Map the *link* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value `"color: <value>"`, where `<value>` is identical to the value of the *link* attribute. This value is to be expressed within a declarations non-terminal enclosed in curly braces with an `a:link` pseudo-class selector according to the syntax specified in Section 5.2.1.9.

Map the *text* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`color: <value>`", where `<value>` is identical to the value of the *text* attribute. If an *alink*, *link*, or *vlink* attribute is present in the source element, then this value is to be expressed within a declarations non-terminal enclosed in curly braces without a selector according to the syntax specified in Section 5.2.1.9.

Map the *vlink* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`color: <value>`", where `<value>` is identical to the value of the *vlink* attribute. This value is to be expressed within a declarations non-terminal enclosed in curly braces with an `a:visited` link pseudo-class selector according to the syntax specified in Section 5.2.1.9.

If the above mappings would cause the creation of multiple inline *style* attributes, then the values of these multiple attributes are to be collected and expressed in a single inline style attribute.

Example: The following depicts a source and two equivalent target fragments after performing the mappings described above. The first target fragment does not merge multiple, contiguous sibling *div* elements, while the second target fragment does perform a merge. This example also shows the use of the more complex form of inline style attribute syntax required to specify inline rule-sets with pseudo-class selectors.

Source Fragment:

```
<body background="bg.png" text="blue" vlink="red">
text1
<b>text2</b>
text3
<p>text4</p>
</blockquote>
```

Target Fragment #1:

```
<body style="{ background-image: url(bg.png); color: blue } a:visited { color: red }">
<div style="display: inline">
text1
</div>
<div style="display: inline">
<b>text2</b>
</div>
<div style="display: inline">
text3
</div>
<p>text4</p>
</body>
```

Target Fragment #2:

```
<body style="{ background-image: url(bg.png); color: blue } a:visited { color: red }">
<div style="display: inline">
text1
<b>text2</b>
text3
</div>
<p>text4</p>
</body>
```

F.2.2.14 *br* element

Map the *clear* attribute to an inline *style* attribute as follows: (1) if a *clear* attribute is present and its *value* is one of `none | left | right`, then map to an inline *style* attribute with the value "`clear: <value>`", where `<value>` is identical to the value of the *clear* attribute; or (2) if a *clear* attribute is present and its value is `all`, then map to an inline *style* attribute with the value "`clear: both`". Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.15 *button* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.16 *caption* element

With the exception of the *align* and common attributes, no special treatment is required; an identity mapping applies.

Map the *align* attribute to an inline *style* attribute as follows: if an *align* attribute is present and its *value* is one of `top | bottom | left | right |`, then map to an inline *style* attribute with the value `"caption-side: <value>"`, where `<value>` is identical to the value of the *align* attribute.

F.2.2.17 *center* element

Map the *center* element to the *div* element with an inline *style* attribute with the value `"text-align: center"`. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.18 *cite* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.19 *code* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.20 *col* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.21 *colgroup* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.22 *dd* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.23 *del* element

Map the *ins* element to the *div* element with block or inline display semantics, and, with the exception of *cite*, *datetime*, and common attributes, no special treatment is required; an identity mapping applies.

If the source *ins* element appears in block display context, then the target *div* element is to specify block display semantics by using an inline *style* attribute with the value `"display: block"`. Otherwise, the target *div* element is to specify inline display semantics by using an inline *style* attribute with the value `"display: inline"`.

Map the *cite* attribute to the null string.

Map the *datetime* attribute to the null string.

F.2.2.24 *dfn* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.25 *dir* element

Map the *dir* element to the *ul* element, and, with the exception of the *compact* and common attributes, no special treatment is required; an identity mapping applies.

Map the *compact* attribute, if present, to the empty string.

F.2.2.26 *div* element

Map the *align* attribute to an inline *style* attribute as follows: if an *align* attribute is present and its *value* is one of `left | center | right | justify`, then map to an inline *style* attribute with the value "`text-align: <value>`", where `<value>` is identical to the value of the *align* attribute. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.27 *dl* element

With the exception of the *compact* and common attributes, no special treatment is required; an identity mapping applies.

Map the *compact* attribute, if present, to the empty string.

F.2.2.28 *dt* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.29 *em* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.30 *fieldset* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.31 *font* element

Map the *font* element to the *span* element with an inline *style* attribute which expresses the values of the *size*, *color*, and *face* attributes, if present. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

If a *size* attribute is present and has an integral value in the range 1 through 7, then map the *size* attribute to an inline *style* attribute with the value "`font-size: <absolute size>`", where `<absolute size>` is determined from the value of the *size* attribute according to Table 7 Legacy Font Size Attribute Mappings.

If a *size* attribute is present and has a signed integral value of -0 or +0, then ignore the *size* attribute; i.e., it is mapped to the empty string.

If a *size* attribute is present and has a signed integral value of -1 or +1, then map the *size* attribute to an inline *style* attribute with the value "`font-size: <relative size>`", where `<relative size>` is either `smaller` or `larger` depending upon the sign.

If a *size* attribute is present and has a signed integral value in the range -3 to -2, then enclose the content of the target *span* element mapped by the source *font* element with either two or one additional *span* element(s) with an inline *style* attribute with the value "`font-size: smaller`".

If a *size* attribute is present and has a signed integral value in the range +3 to +2, then enclose the content of the target *span* element mapped by the source *font* element with either two or one additional *span* element(s) with an inline *style* attribute with the value "`font-size: larger`".

Map the *color* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`color: <value>`", where `<value>` is identical to the value of the *color* attribute.

Map the *face* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`font-family: <value>`", where `<value>` is identical to the value of the *face* attribute, except that, if the value of the *face* attribute contains any whitespace characters, then `<value>` is to be further quoted with single-quote (`'`) characters or with an equivalent entity or character reference, i.e., `'` or `'`.

If the above mappings would cause the creation of multiple inline *style* attributes, then the values of these multiple attributes are to be collected and expressed in a single inline style attribute.

Example: The following depicts a source and target fragment after performing the mappings described above. This example shows the use of a relative font size and the generation of multiple nested *span* elements to map the semantics of this usage.

Source Fragment:

```
<font size="+3" color="red" face="Arial Bold">
text1
</font>
```

Target Fragment:

```
<span style="font-size: larger; color: red; font-family: 'Arial Bold'">
<span style="font-size: larger">
<span style="font-size: larger">
text1
</span>
</span>
</span>
```

F.2.2.32 *form* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.33 *frame* element

Except for the *longdesc*, *src* and common attributes, no special treatment is required; an identity mapping applies.

If the URI value of the *longdesc* attribute employs the "`http:`" URI scheme, then map the scheme token to "`lid:`" and repackage the referenced resource as a local resource. Otherwise, an identity mapping applies for the attribute value.

If the URI value of the *src* attribute employs the "`http:`" URI scheme, then map the scheme token to "`lid:`" and repackage the referenced resource as a local resource. If the URI value employs the "`javascript:`" URI scheme, then map the scheme token to "`ecmascript:`". Otherwise, an identity mapping applies for the attribute value.

F.2.2.34 *frameset* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.35 *h1* element

Map the *align* attribute to an inline *style* attribute as follows: if an *align* attribute is present and its *value* is one of `left | center | right | justify`, then map to an inline *style* attribute with the value "`text-align: <value>`", where `<value>` is identical to the value of the *align* attribute. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.36 *h2* element

Map the *align* attribute to an inline *style* attribute as follows: if an *align* attribute is present and its *value* is one of `left | center | right | justify`, then map to an inline *style* attribute with the value "`text-align: <value>`", where `<value>` is identical to the value of the *align* attribute. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.37 *h3* element

Map the *align* attribute to an inline *style* attribute as follows: if an *align* attribute is present and its *value* is one of `left | center | right | justify`, then map to an inline *style* attribute with the value "`text-align: <value>`", where `<value>` is identical to the value of the *align* attribute. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.38 *h4* element

Map the *align* attribute to an inline *style* attribute as follows: if an *align* attribute is present and its *value* is one of `left | center | right | justify`, then map to an inline *style* attribute with the value "`text-align: <value>`", where `<value>` is identical to the value of the *align* attribute. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.39 *h5* element

Map the *align* attribute to an inline *style* attribute as follows: if an *align* attribute is present and its *value* is one of `left | center | right | justify`, then map to an inline *style* attribute with the value "`text-align: <value>`", where `<value>` is identical to the value of the *align* attribute. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.40 *h6* element

Map the *align* attribute to an inline *style* attribute as follows: if an *align* attribute is present and its *value* is one of `left | center | right | justify`, then map to an inline *style* attribute with the value "`text-align: <value>`", where `<value>` is identical to the value of the *align* attribute. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.41 *head* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.42 *hr* element

With the exception of *align*, *noshade*, *size*, *width*, and common attributes, no special treatment is required; an identity mapping applies.

Map the *align* attribute to an inline *style* attribute as follows: if an *align* attribute is present and its *value* is one of `left | center | right`, then map to an inline *style* attribute with the value "`text-align: <value>`", where `<value>` is identical to the value of the *align* attribute.

Map the *noshade* attribute, if present, to an inline *style* attribute with the value "`color: gray`".

Map the *size* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`height: <value>px`", where `<value>` is identical to the value of the *size* attribute.

Map the *width* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`width: <value>px`", where `<value>` is identical to the value of the *width* attribute.

If the above mappings would cause the creation of multiple inline *style* attributes, then the values of these multiple attributes are to be collected and expressed in a single inline style attribute.

F.2.2.43 *html* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.44 *i* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.45 *iframe* element

Map the *iframe* element to the *div* element, ignoring all attributes other than common attributes, which are mapped as described above. See Section F.2.1

F.2.2.46 *img* element

Map the *img* element to the *object* element, and, with the exception of *align*, *alt*, *border*, *hspace*, *ismap*, *longdesc*, *src*, *vspace*, and common attributes, no special treatment is required; an identity mapping applies.

Map the *align* attribute, if present and a valid, non-empty value, to an inline *style* attribute as follows: (1) if an *align* attribute is present and its *value* is one of `left | right`, then map to an inline *style* attribute with the value "`float: <value>`", where `<value>` is identical to the value of the *align* attribute; or (2) if an *align* attribute is present and its *value* is `bottom | middle | top`, then map to an inline *style* attribute with the value "`vertical-align: <value>`", where `<value>` is identical to the value of the *align* attribute.

Map the *alt* attribute, if present, to `#PCDATA` content appearing as a child of the target *object* element.

Map the *border* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`border: <value>px`", where `<value>` is identical to the value of the *border* attribute.

Map the *hspace* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`margin-left: <value>px; margin-right: <value>px`", where `<value>` is identical to the value of the *hspace* attribute.

Map the *ismap* attribute, if present, to the null string.

Map the *longdesc* attribute, if present, to content appearing as a child of the target *object* element such that the value of the *longdesc* attribute is expressed as the value of the *href* attribute of a generated *a* (anchor) child element with the content of the *a* element being `#PCDATA` that indicates the availability of further information. If an *alt* attribute was also present on the source *img* element and mapped to `#PCDATA` content of the *object* element, then concatenate the mapped data or enclose in separate *p* (paragraph) child elements.

Map the *src* attribute to a *data* attribute, where the value of the target *object* element's *data* attribute is identical to the value of the source *img* element's *src* attribute. In addition, if the URI value of the *src* attribute employs the "`http:`" URI scheme, then map the scheme token to "`lid:`" and repackage the referenced resource as a local resource.

Map the *vspace* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`margin-top: <value>px; margin-bottom: <value>px`", where `<value>` is identical to the value of the *vspace* attribute.

If the above mappings would cause the creation of multiple inline *style* attributes, then the values of these multiple attributes are to be collected and expressed in a single inline style attribute.

Example: The following depicts a source and target fragment after performing the mapping described above. Notice also the mapping of the URI for the long description reference to reflect that the target will be transcoded to XDML.

Source Fragment:

```
<img src="foo.png" alt="An Image" longdesc="foo-desc.html">
```

Target Fragment:

```
<object data="foo.png">
<p>An Image</p>
<p><a href="foo-desc.xhtml">Click here for further information.</a></p>
</object>
```

F.2.2.47 *input* element

With the exception of *align*, *ismap*, *src*, *type*, and common attributes, no special treatment is required; an identity mapping applies.

Map the *align* attribute, if present and a valid, non-empty value, to an inline *style* attribute as follows: (1) if an *align* attribute is present and its *value* is one of `left | right`, then map to an inline *style* attribute with the value "`float: <value>`", where `<value>` is identical to the value of the *align* attribute; or (2) if an *align* attribute is present and its *value* is `bottom | middle | top`, then map to an inline *style* attribute with the value "`vertical-align: <value>`", where `<value>` is identical to the value of the *align* attribute.

Map the *ismap* attribute, if present, to the null string.

If the URI value of the *src* attribute employs the "`http:`" URI scheme, then map the scheme token to "`lid:`" and repackage the referenced resource as a local resource.

If the value of the *type* attribute is `file`, then map this value to `hidden` in the target element's *type* attribute.

F.2.2.48 *ins* element

Map the *ins* element to the *div* element with block or inline display semantics, and, with the exception of *cite*, *datetime*, and common attributes, no special treatment is required; an identity mapping applies.

If the source *ins* element appears in block display context, then the target *div* element is to specify block display semantics by using an inline *style* attribute with the value "`display: block`". Otherwise, the target *div* element is to specify inline display semantics by using an inline *style* attribute with the value "`display: inline`".

Map the *cite* attribute to the null string.

Map the *datetime* attribute to the null string.

F.2.2.49 *isindex* element

Map the *isindex* element to either an *input* element or the null string. If the *isindex* element appears in block context, then map to an *input* element as described below; otherwise, map to the null string.

If the *isindex* element appears in block context, then perform the following steps: (1) map the *isindex* element to an *input* element with a *type* attribute with the value `text`; (2) if no *prompt* attribute is present or if it is present and non-empty, then use an `#IMPLIED` default value for the *prompt* attribute, e.g., `"Enter Search Keyword(s): "`; (3) map the *prompt* attribute, if implied or specified, to an enclosing *p* element containing two children: a `#PCDATA` node consisting of the value of the *prompt* attribute and the target *input* element; (4) If the source *isindex* element is not a child of a *form* element, then enclose the results of the previous step in a *form* element with an *action* attribute with a value equal to the document's base resource identifier.

> *Note:* Consideration should be given to [WAI-UAAG] Checkpoint 9.2 regarding the use of an explicit submit control; in particular, a transcoding process may wish to include within a generated *form* element an additional *input* element with a *type* attribute with the value `submit`.

Example: The following depicts a source and target fragment after performing the mapping described above. The value of the *action* attribute of the form element in the target fragment would be determined from the base resource identifier of the document containing the target fragment.

Source Fragment:

```
<isindex prompt="Enter search phrase: ">
```

Target Fragment:

```
<form action="..."><p>Enter search phrase: <input type="text"/></p></form>
```

F.2.2.50 *kbd* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.51 *label* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.52 *legend* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.53 *li* element

With the exception of the *type*, *value* and common attributes, no special treatment is required; an identity mapping applies.

Map the *type* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value `"list-style-type: <value> "`, where `<value>` is determined from the value of the *type* attribute according to the following table:

Table 17 List Element Type Attribute Mapping

1	decimal
a	lower-latin
A	upper-latin
i	lower-roman
I	upper-roman
disc	disc
square	square
circle	circle

If the value of the *type* attribute is not one of the values shown in the first column of the above table, then map the *type* attribute to the null string.

Map the *value* attribute, if present and a valid, non-empty integral value, to an inline *style* attribute with the value "`counter-reset: item <value - 1>`", where `<value - 1>` is identical to the integral value of the *value* attribute minus one.

F.2.2.54 *link* element

If the *rel* attribute is present on the source *link* element and its value is `stylesheet`, then map the *link* element to an *xml-stylesheet* processing instruction, as described in [XMLSTYLE]; otherwise, map the *link* element to the null string.

If the URI value of the *href* attribute employs the "`http:`" URI scheme, then map the scheme token to "`lid:`" and repackage the referenced resource as a local resource.

F.2.2.55 *map* element

If no *id* attribute is present in the source *map* element, then create a new *id* attribute on the target *map* element with a document unique `ID` value. It is recommended that this value be formed by concatenating the value of the *name* value, a colon `':'`, and a document unique token that adheres to `NMTOKEN` syntax. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

Example: The following depicts a source and target fragment after performing the mapping described above.

Source Fragment:

```
<map name="map1">
<area href="go.html" alt="Click Here!" shape="rect" coords="100,100,200,200">
</map>
```

Target Fragment:

```
<map name="map1" id="map1:id12345">
<area href="go.xhtml" alt="Click Here!" shape="rect" coords="100,100,200,200"/>
</map>
```

F.2.2.56 *menu* element

Map the *menu* element to the *ul* element, and, with the exception of the *compact* and common attributes, no special treatment is required; an identity mapping applies.

Map the *compact* attribute, if present, to the empty string.

F.2.2.57 *meta* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.58 *noframes* element

With the exception of permissible context of use and common attributes, no special treatment is required; an identity mapping applies. In regards to permissible context of use, if a *noframes* element appears in any context other than as a child of a *frameset* element, then each such *noframes* element is to be mapped to a *div* element with block display semantics, where the *div* element's content is the content of the *noframes* element.

Block display semantics on a *div* element are obtained by using an inline *style* attribute with the value "`display: block`".

F.2.2.59 *noscript* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.60 *object* element

With the exception of *align*, *archive*, *border*, *classid*, *codebase*, *data*, *hspace*, *ismap*, *vspace*, and common attributes, no special treatment is required; an identity mapping applies.

Map the *align* attribute, if present and a valid, non-empty value, to an inline *style* attribute as follows: (1) if an *align* attribute is present and its *value* is one of `left | right`, then map to an inline *style* attribute with the value "`float: <value>`", where `<value>` is identical to the value of the *align* attribute; or (2) if an *align* attribute is present and its *value* is `bottom | middle | top`, then map to an inline *style* attribute with the value "`vertical-align: <value>`", where `<value>` is identical to the value of the *align* attribute.

If the URI value of the *archive*, *classid* or *data* attributes employs the "`http:`" URI scheme, then map the scheme token to "`lid:`" and repackage the referenced resource as a local resource. If the URI value of the *codebase* attribute employs the "`http:`" URI scheme, then map the scheme token to "`lid:`".

Map the *border* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`border: <value>px`", where `<value>` is identical to the value of the *border* attribute.

Map the *hspace* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`margin-left: <value>px; margin-right: <value>px`", where `<value>` is identical to the value of the *hspace* attribute.

Map the *ismap* attribute, if present, to the null string.

Map the *vspace* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`margin-top: <value>px; margin-bottom: <value>px`", where `<value>` is identical to the value of the *vspace* attribute.

If the above mappings would cause the creation of multiple inline *style* attributes, then the values of these multiple attributes are to be collected and expressed in a single inline style attribute.

> *Note:* Depending upon the content type of a source object's data or code, it may be necessary to transcode the referenced content as well as the value of the *type* and *codetype* attributes in order to satisfy the limitations of content type support by DASE Declarative Applications.
>
> *Note:* The type attribute of trigger receiver objects contained in source content according to [DDE-1] is mapped to `application/dase-trigger`. See 5.1.1.6.8.2 for further information.

F.2.2.61 *ol* element

With the exception of the *compact*, *start*, *type* and common attributes, no special treatment is required; an identity mapping applies.

Map the *compact* attribute, if present, to the empty string.

Map the *start* attribute, if present and a valid, non-empty integral value, to an inline *style* attribute with the value "`counter-reset: item <value - 1>`", where `<value - 1>` is identical to the integral value of the *start* attribute minus one.

Map the *type* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`list-style-type: <value> `", where `<value>` is determined from the value of the *type* attribute according to the following table:

Table 18 Ordered List Element Type Attribute Mapping

1	decimal
a	lower-latin

A	upper-latin
i	lower-roman
I	upper-roman

If the value of the *type* attribute is not one of the values shown in the first column of the above table, then map the *type* attribute to the null string.

F.2.2.62 *optgroup* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.63 *option* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.64 *p* element

Map the *align* attribute to an inline *style* attribute as follows: if an *align* attribute is present and its *value* is one of `left | center | right | justify`, then map to an inline *style* attribute with the value "`text-align: <value>`", where `<value>` is identical to the value of the *align* attribute. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.65 *param* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.66 *pre* element

With the exception of the *width* and common attributes, no special treatment is required; an identity mapping applies.

Map the *width* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`width: <value>em`", where `<value>` is identical to the value of the *width* attribute.

F.2.2.67 *q* element

Except for the *cite* and common attributes, no special treatment is required; an identity mapping applies.

If the URI value of the *cite* attribute employs the "`http:`" URI scheme, then map the scheme token to "`lid:`" and repackage the referenced resource as a local resource.

F.2.2.68 *s* element

Map the *s* element to the *span* element with an inline *style* attribute with the value "`text-decoration: line-through`". Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.69 *samp* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.70 *script* element

With the exception of element content and the *event*, *for*, *language*, *src* and common attributes, no special treatment is required; an identity mapping applies.

Map the `CDATA` content of the *script* element to `#PCDATA` content as follows: (1) remove markup declaration open (mdo) followed immediately by comment open (co), i.e., "`<!--`", and

markup declaration close (mdc) preceded immediately by comment close, i.e., `"-->"`; and (2) either substitute entity references for each occurrence of `'<'` and `'&'`, e.g., using `<` and `&`, respectively, or enclose the result of step (1) in a `CDATA` marked section.

Map the reserved *event* attribute, if present, to the null string.

Map the reserved *for* attribute, if present, to the null string.

Map the *language* attribute, if present and a valid, non-empty value, to either the *type* attribute or the empty string. If the *language* attribute value specifies a script language that is syntactically and semantically a subset of [ECMASCRIPT] and no *type* attribute is present on the source *script* element, then create a *type* attribute on the target *script* element with the value `text/ecmascript`. If a *type* attribute is present on the source *script* element and its value is `text/ecmascript`, then map the *language* attribute to the empty string. Otherwise, map the entire element, including its content, to the null string.

If the URI value of the *src* attribute employs the `"http:"` URI scheme, then map the scheme token to `"lid:"` and repackage the referenced resource as a local resource. Otherwise, an identity mapping applies for the attribute value.

Example: The following depicts a source and target fragments after performing the mappings described above. Two example targets are shown, one using entity references to map delimiters otherwise recognized in `#PCDATA` content, the other using a `CDATA` marked section to disable `#PCDATA` delimiter recognition.

Source Fragment:

```
<script language="Javascript 1.0">
<!--
x &= y << 1;
-->
</script>
```

Target Fragment #1:

```
<script type="text/ecmascript">
x &amp;= y &lt;&lt; 1;
</script>
```

Target Fragment #2:

```
<script type="text/ecmascript">
<![CDATA[
x &= y << 1;
]]>
</script>
```

F.2.2.71 *select* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.72 *small* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.73 *span* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.74 *strike* element

Map the *strike* element to the *span* element with an inline *style* attribute with the value `"text-decoration: line-through"`. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.75 *strong* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.76 *style* element

With the exception of element content and the *type* and common attributes, no special treatment is required; an identity mapping applies.

Map the `CDATA` content of the *script* element to `#PCDATA` content as follows: (1) remove markup declaration open (mdo) followed immediately by comment open (co), i.e., `"<!--"`, and markup declaration close (mdc) preceded immediately by comment close, i.e., `"-->"`; and (2) either substitute entity references for each occurrence of `'<'` and `'&'`, e.g., using `<` and `&`, respectively, or enclose the result of step (1) in a `CDATA` marked section.

If the value of the *type* attribute is not `text/css`, then map the entire element, including its content, to the null string.

F.2.2.77 *sub* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.78 *sup* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.79 *table* element

With the exception of the *align*, *bgcolor* and common attributes, no special treatment is required; an identity mapping applies.

Map the *align* attribute to an inline *style* attribute as follows: if an *align* attribute is present and its *value* is one of `left | center |`, then map to an inline *style* attribute with the value `"text-align: <value>"`, where `<value>` is identical to the value of the *align* attribute.

Map the *bgcolor* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value `"background-color: <value>"`, where `<value>` is identical to the value of the *bgcolor* attribute.

If the above mappings would cause the creation of multiple inline *style* attributes, then the values of these multiple attributes are to be collected and expressed in a single inline style attribute.

F.2.2.80 *tbody* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.81 *td* element

With the exception of the *bgcolor*, *height*, *nowrap*, *width* and common attributes, no special treatment is required; an identity mapping applies.

Map the *bgcolor* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value `"background-color: <value>"`, where `<value>` is identical to the value of the *bgcolor* attribute.

Map the *height* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value `"height: <value>px"`, where `<value>` is identical to the value of the *height* attribute.

Map the *nowrap* attribute, if present, to an inline *style* attribute with the value `"white-space: nowrap"`.

Map the *width* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`width: <value>px`", where `<value>` is identical to the value of the *width* attribute.

If the above mappings would cause the creation of multiple inline *style* attributes, then the values of these multiple attributes are to be collected and expressed in a single inline style attribute.

F.2.2.82 *textarea* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.83 *tfoot* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.84 *th* element

With the exception of the *bgcolor*, *height*, *nowrap*, *width* and common attributes, no special treatment is required; an identity mapping applies.

Map the *bgcolor* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`background-color: <value>`", where `<value>` is identical to the value of the *bgcolor* attribute.

Map the *height* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`height: <value>px`", where `<value>` is identical to the value of the *height* attribute.

Map the *nowrap* attribute, if present, to an inline *style* attribute with the value "`white-space: nowrap`".

Map the *width* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`width: <value>px`", where `<value>` is identical to the value of the *width* attribute.

If the above mappings would cause the creation of multiple inline *style* attributes, then the values of these multiple attributes are to be collected and expressed in a single inline style attribute.

F.2.2.85 *thead* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.86 *title* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.87 *tr* element

With the exception of the *bgcolor* and common attributes, no special treatment is required; an identity mapping applies.

Map the *bgcolor* attribute, if present and a valid, non-empty value, to an inline *style* attribute with the value "`background-color: <value>`", where `<value>` is identical to the value of the *bgcolor* attribute.

If the above mappings would cause the creation of multiple inline *style* attributes, then the values of these multiple attributes are to be collected and expressed in a single inline style attribute.

F.2.2.88 *tt* element

Except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.89 *u* element

Map the *u* element to the *span* element with an inline *style* attribute with the value `"text-decoration: underline"`. Otherwise, except for common attributes, no special treatment is required; an identity mapping applies.

F.2.2.90 *ul* element

With the exception of the *compact*, *type* and common attributes, no special treatment is required; an identity mapping applies.

Map the *compact* attribute, if present, to the empty string.

Map the *type* attribute to an inline *style* attribute as follows: if a *type* attribute is present and its *value* is one of `disc | circle | square`, then map to an inline *style* attribute with the value `"list-style-type: <value>"`, where `<value>` is identical to the value of the *type* attribute.

F.2.2.91 *var* element

Except for common attributes, no special treatment is required; an identity mapping applies.

ANNEX G. ACCESSIBILITY CONSIDERATIONS

This section is informative.

Wherever possible, authors of a declarative application should follow the guidelines prescribed by *Web Content Accessibility Guidelines 1.0* [WAI-WCAG]. In particular, declarative applications should adhere to all applicable Priority 1 and 2 checkpoints.

> *Note:* This specification makes certain design choices which force or encourage adherence to certain [WAI-WCAG] checkpoints: certain legacy, presentation markup (e.g., the *font* element) is not supported, forcing the use of style sheets to represent presentation information; server-side image maps are not supported, etc.

Wherever possible, implementers of an authoring tool for declarative applications (including tools used to transcode legacy content into the content types defined by this specification) should follow the guidelines prescribed by *Authoring Tool Accessibility Guidelines 1.0* [WAI-ATAG].

Wherever possible, implementers of a declarative application environment should follow the guidelines prescribed by *User Agent Accessibility Guidelines 1.0* [WAI-UAAG].

ANNEX H. EXAMPLES OF OBJECT ELEMENT USAGE

The entirety of this section and its subsections is informative.

This annex presents examples of the use of the *object* element defined by this specification. See Section 5.1.1.6.8.1 for additional information.

H.1 Active Object Content Example

This example depicts an *object* element used to reference a MacroMedia Flash™ 5 Movie which will be decoded and presented by an application specified content decoder implemented as a Java TV Xlet. The base URI of the containing document is `"lid://xyz.com/apps/current/"` for the purpose of this example.

```
<object
  archive="flash5.jar"
  classid="com/acme/decoder/Flash.class"
  codetype="application/javatv-xlet"
  data="movie.swf"
  type="application/octet-stream"
  alt="A Macromedia Flash Movie"
>
  <param name="arg.0" value="play: true"/>
  <param name="arg.1" value="loop: true"/>
  <param name="arg.2" value="quality: high"/>
</object>
```

When the Xlet specified by this *object* element instance is initialized, it will be provided access via its XlexContext to a set of properties derived from the following:

- `"org.atsc.xlet.obj.codebase"`, `"lid://xyz.com/apps/current/"`
- `"org.atsc.xlet.obj.data"`, `"movie.swf"`
- `"org.atsc.xlet.obj.type"`, `"application/octet-stream"`
- `"javax.tv.xlet.args[0]"`, `"play: true"`
- `"javax.tv.xlet.args[1]"`, `"loop: true"`
- `"javax.tv.xlet.args[2]"`, `"quality: high"`

Note: This example presumes that it is possible to construct a decoder for the Flash 5 content type using facilities permitted by [DASE], Section 4.1.3. Limitations of receiver performance may restrict the utility of such a decoder.

H.2 Trigger Content Object Element Example

This example depicts an *object* element which serves as a target for *script* triggers as supported by Section 4.5. This *object* element produces no rendered content except as a possible side effect of decoding and executing script content contained in triggers which are targeted to this *object* element instance.

```
<object id="target1" type="application/dase-trigger"/>
```

Note: In general, the use of a trigger content object element is not required in order to use *script* triggers; rather, the purpose of the trigger content object element is to support certain legacy application content.

CHANGES

This section is informative.

Changes from Candidate Standard to Standard

The following table enumerates the changes between the issuance of the candidate standard edition of this specification and the standard edition.

Table 19 Changes from Candidate Standard

Section	Description
1	Change status to standard.
2.1	Add [XHTMLBASIC] normative reference.
2.1	Add [XHTMLMIME] normative reference.
2.1	Change [DOM2-HTML] to W3C Recommendation status.
2.2	Add [DOM2-ERR] informative reference.
4.7	Specify ecmascript resource identifier scheme.
5.1	Change application/xdml+xml to application/xhtml+xml per [XHTMLMIME].
5.1	Remove "Host Language" from XDML FPI to make consistent with [XHTMLMOD] naming rules.
5.1	Add XHTML Basic to list of supported document types per [XHTMLBASIC].
5.1	Merge XDML primary and frameset document types into single document type.
5.1	Specify full XHTML Family User Agent Conformance.
5.1	Genericize "XDML document" to "XHTML document" to account for generic XHTML UA conformance.
5.1	Improve consistency of recommendation to validate markup content.
5.1	Add recommendation to support additional XHTML host language conformant document types.
5.1	Clarify semantics of name attribute with regard to elements which require name attribute.
5.1	Clarify implied default value of *name* attribute.
5.1	Specify support for base, link, and image modules to facilitate XHTML Basic usage.
5.1	Specify ecmascript resource identifier usage for *a* (anchor) element.
5.1	Clarify active object xlet context property name usage.
5.1	Specify use of external stylesheet reference via link element, stylesheet relationship.
5.2	Add atsc-dynamic-refresh style property.
5.2	Add atsc-nav-{index,left,right,up,down} style properties.
5.2	Clarify usage of phrase *non-standard* with regard to W3C usage.
5.3	Deprecate use of StringExt interface functionality.
5.3	Note clarifications and corrections to [DOM2*] found in [DOM2-ERR].
5.3	Insure consistency of interface support tables 8, 10, 11 and 15.
5.3	Note rationale for permitting Attr::ownerElement to be null for defaulted attribute values.
5.3	Specify org.atsc.dom.events feature string for DOMImplementation::hasFeature.
5.3	Insure consistency of org.atsc.* feature strings.
5.3	Remove HTMLDOMImplementation due to removal from DOM-2 HTML CR.
5.3	Specify use of HTMLImageElement and HTMLOptionsCollection.
5.3	Add HTMLImageElementExt to support legacy properties on HTMLImageElement.
5.3	Clarify cookie attribute-value pair support per [HTTP-STATE].
5.3	Specify use of legacy *expires* cookie attribute-value pair.
5.3	Clarify semantics of HTMLDocument::images property with regards to *img* elements.
5.3	Note rationale for not specifying explicit HTMLDocumentExt::window binding in ECMAScript.
5.3	Add DocumentViewExt::refresh method.
5.3	Add KeyEvent, KeyModifiers, and VirtualKeys interfaces.
5.3	Add key and text event types: org.atsc.key{down,up}, org.atsc.textinput.
5.3	Add mutation event types: org.atsc.resourcechanged.
5.3	Clarify semantics of Navigator::ddeBackChannel.
5.3	Clarify semantics of Navigator::ddeEnabled.
5.3	Clarify semantics of Navigator::ddeSourceId.
5.3	Clarify semantics of Navigator::ddeReleasable.
5.3	Change Navigator::userAgent standard product token from XDML-UA/1.0 to DASE-1/1.0.
A	Remove XDML frameset document type (now merged with standard XDML document type).
A.1	Insure validity of XDML document type against [XHTMLMOD] DTD fragments.
A.1	Add base, link, and image modules and respective content model support.
A.1	Parameterize inline form element class.
A.1	Insure consistency of blockquote content model with XHTML 1.0 Transitional.

Section	Description
A.1	Remove extraneous definition of label content model.
A.1	Add XHTML.dtd.sysid.base parameter entity to qualify XHTMLMOD system identifiers.
A.1	Move inclusion of XHTML Name Identification module to prevent errant pre-declarations.
C	Add Events Extensions Module IDL definition.
C	Add HTMLImageElementExt IDL definition.
C	Add DocumentViewExt::method IDL definition.
D	Add Events Extensions Module IDL ECMAScript binding.
D	Add HTMLImageElementExt IDL ECMAScript binding.
D	Add DocumentViewExt::method IDL ECMAScript binding.
F	Specify transcoding mappings for "http:" and "javascript:" URI schemes in source document.
G	Change annex to informative based on nature of recommendations and use of informative references.