Timed Text Markup Language 1 (TTML1) (Third Edition)

W3C Candidate Recommendation 09 August 2018

This version:
https://www.w3.org/TR/2018/CR-ttml1-20180809/

Latest editor's draft:
https://w3c.github.io/ttml1/index.html

Latest version:
https://www.w3.org/TR/ttml1/

Latest recommendation:
https://www.w3.org/TR/2013/REC-ttml1-20130924/

Previous version:
https://www.w3.org/TR/2018/CR-ttml1-20180424/

Editors:
Glenn Adams, Skynav
Pierre-Anthony Lemieux, MovieLabs

Contributing Authors:
Mike Dolan, Invited Expert
Geoff Freed, WGBH National Center for Accessible Media
Sean Hayes, Microsoft
Erik Hodge, RealNetworks
David Kirby, British Broadcasting Corporation (BBC)
Thierry Michel, W3C
Dave Singer, Apple Computer
Andreas Tai, Invited Expert

See also **translations**.

Abstract

This document specifies Timed Text Markup Language (TTML), Version 1, also known as TTML1, in terms of a vocabulary and semantics thereof.

The Timed Text Markup Language is a content type that represents timed text media for the purpose of interchange among authoring systems. Timed text is textual information that is intrinsically or extrinsically associated with timing information.

It is intended to be used for the purpose of transcoding or exchanging timed text information among legacy distribution content formats presently in use for subtitling and captioning functions.

In addition to being used for interchange among legacy distribution content formats, TTML Content may be used directly as a distribution format, for example, providing a standard content format to reference from a `<track>` element in an HTML5 document, or a `<text>` or `<textstream>` media element in a [SMIL 2.1] document.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

This document was published by the Timed Text Working Group as a Candidate Recommendation. This document is intended to become a W3C Recommendation. Comments regarding this document are welcome. Please send them to public-tt@w3.org (subscribe, archives) with `[ttml1]` at the start of your email's subject. W3C publishes a Candidate Recommendation to indicate that the document is believed to be stable and to encourage implementation by the developer community. This Candidate Recommendation is expected to advance to Proposed Recommendation no earlier than 25 September 2018.

The deadline for comments is 6 September 2018.

This edition of the Timed Text Markup Language (TTML) 1.0 [TTML10] clarifies ambiguities and corrects errors identified in the previous edition, which this edition supersedes. No feature is added or removed in this edition.

For this specification to exit the CR stage, at least 2 independent implementations of every *CR exit criteria test* are required, as documented in the Working Group's implementation report. These *CR exit criteria tests* are intended to cover the ambiguities and errors corrected by this edition, and the implementation report will be limited to these tests. The Working Group does not require that implementations are publicly available but encourages them to be so.

Substantive changes applied since the previous version of this specification are listed at substantive-changes-summary.txt.

For convenience, a diff between this and the previous edition is offered at the W3C HTML Diff service.

The Working Group has not identified features "at risk" for this specification.

Publication as a Candidate Recommendation does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document was produced by a group operating under the W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document is governed by the 1 February 2018 W3C Process Document.

Table of Contents

Appendices

1 Introduction

Unless specified otherwise, this section and its sub-sections are non-normative.

The Timed Text Markup Language (TTML) Version 1, also referred to as TTML1, provides a standardized representation of a particular subset of textual information with which stylistic, layout, and timing semantics are associated by an author or an authoring system for the purpose of interchange and processing.

TTML is expressly designed to meet only a limited set of requirements established by [TTAF1-REQ], and summarized in **I Requirements**. In particular, only those requirements which service the need of performing interchange with existing, legacy distribution systems are satisfied.

In addition to being used for interchange among legacy distribution content formats, TTML Content may be used directly as a distribution format, providing, for example, a standard content format to reference from a `<track>` element in an HTML5 document, or a `<text>` or `<textstream>` media element in a [SMIL 2.1] document. Certain properties of TTML support streamability of content, as described in **L Streaming TTML Content**.

> **Note:**
>
> While TTML is not expressly designed for direct (embedded) integration into an HTML or a SMIL document instance, such integration is not precluded.

> **Note:**
>
> In some contexts of use, it may be appropriate to employ animated content to depict sign language representations of the same content as expressed by a Timed Text *Document Instance*. This use case is not explicitly addressed by TTML mechanisms, but may be addressed by some external multimedia integration technology, such as SMIL.

> **Note:**
>
> In the first edition of this specification, the version number `1.0` was used to refer to this version of TTML. In this second edition, this version number is simplified to `1` as a result of a determination to call the next (future) version `2` instead of `1.1`.

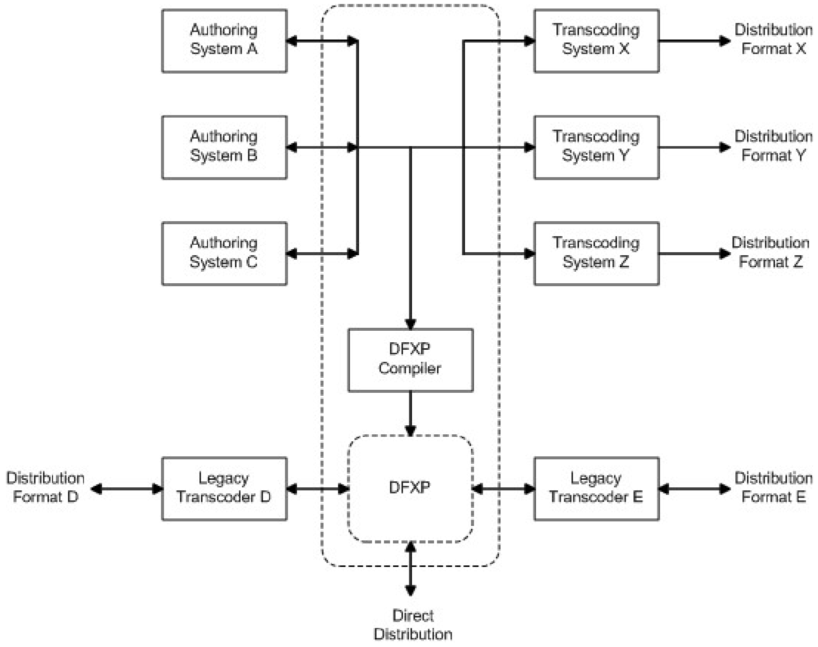
Note:

In previous drafts of this specification, TTML was referred to as DFXP (Distribution Format Exchange Profile). This latter term is retained for historical reasons in certain contexts, such as profile names and designators, and the short name `ttaf1-dfxp` used in older URLs to refer to this specification. Starting in this edition of the first version of TTML, the short name `ttml1` will be used, and subsequent versions of TTML are expected to use short names of `ttml2`, etc., with integer version numbers.

1.1 System Model

Use of TTML is intended to function in a wider context of Timed Text Authoring and Distribution mechanisms that are based upon the system model depicted in **Figure 1 – System Model**, wherein the Timed Text Markup Language serves as a bidirectional interchange format among a heterogeneous collection of authoring systems, and as a unidirectional interchange format to a heterogeneous collection of distribution formats after undergoing transcoding or compilation to the target distribution formats as required, and where one particular distribution format is TTML (labeled as "DFXP" in the figure).

Figure 1 – System Model



1.2 Document Example

A TTML *Document Instance* consists of a `tt` document element that contains a header and a body, where the header specifies document level metadata, styling definitions and layout definitions, and the body specifies text content intermixed with references to style and layout information and inline styling and timing information.

Example Fragment – TTML Document Structure

```
<tt xml:lang="" xmlns="http://www.w3.org/ns/ttml">
  <head>
    <metadata/>
    <styling/>
    <layout/>
  </head>
  <body/>
</tt>
```

Document level metadata may specify a document title, description, and copyright information. In addition, arbitrary metadata drawn from other namespaces may be specified.

Example Fragment – TTML Metadata

```
<metadata xmlns:ttm="http://www.w3.org/ns/ttml#metadata">
  <ttm:title>Timed Text TTML Example</ttm:title>
  <ttm:copyright>The Authors (c) 2006</ttm:copyright>
</metadata>
```

Styling information may be specified in the form of style specification definitions that are referenced by layout and content information, specified inline with content information, or both.

In **Example Fragment – TTML Styling**, four style sets of specifications are defined, with one set serving as a collection of default styles.

Example Fragment – TTML Styling

```
<styling xmlns:tts="http://www.w3.org/ns/ttml#styling">
  <!-- s1 specifies default color, font, and text alignment -->
  <style xml:id="s1"
    tts:color="white"
    tts:fontFamily="proportionalSansSerif"
    tts:fontSize="22px"
    tts:textAlign="center"
  />
  <!-- alternative using yellow text but otherwise the same as style s1 -->
  <style xml:id="s2" style="s1" tts:color="yellow"/>
  <!-- a style based on s1 but justified to the right -->
  <style xml:id="s1Right" style="s1" tts:textAlign="end" />
  <!-- a style based on s2 but justified to the left -->
  <style xml:id="s2Left" style="s2" tts:textAlign="start" />
</styling>
```

Layout information defines one or more regions into which content is intended to be presented. A region definition may reference one or more sets of style specifications in order to permit content flowed in the region to inherit from these styles. In **Example Fragment – TTML Layout**, the region definition makes reference to style specification `s1` augmented by specific inline styles which, together, allow content flowed into the region to inherit from the region's styles (in the case that a style is not already explicitly specified on content or inherited via the content hierarchy.)

Example Fragment – TTML Layout

```
<layout xmlns:tts="http://www.w3.org/ns/ttml#styling">
  <region xml:id="subtitleArea"
    style="s1"
    tts:extent="560px 62px"
    tts:padding="5px 3px"
    tts:backgroundColor="black"
    tts:displayAlign="after"
  />
</layout>
```

The content of a *Document Instance* is expressed in its body, which is organized in terms of block and inline text elements. The hierarchical organization of content elements serves a primary role in determining both spatial and temporal relationships. For example, in **Example Fragment – TTML Body**, each paragraph (`p` element) is flowed into its target region in the specified lexical order; furthermore, the active time interval of each paragraph is timed in accordance to its parent or sibling according to the applicable time containment semantics — in this case, the division parent is interpreted (by default) as a parallel time container.

Example Fragment – TTML Body

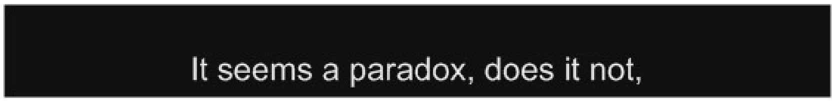
```
<body region="subtitleArea">
  <div>
    <p xml:id="subtitle1" begin="0.76s" end="3.45s">
      It seems a paradox, does it not,
    </p>
    <p xml:id="subtitle2" begin="5.0s" end="10.0s">
      that the image formed on<br/>
      the Retina should be inverted?
    </p>
    <p xml:id="subtitle3" begin="10.0s" end="16.0s" style="s2">
      It is puzzling, why is it<br/>
      we do not see things upside-down?
    </p>
    <p xml:id="subtitle4" begin="17.2s" end="23.0s">
      You have never heard the Theory,<br/>
      then, that the Brain also is inverted?
    </p>
    <p xml:id="subtitle5" begin="23.0s" end="27.0s" style="s2">
      No indeed! What a beautiful fact!
    </p>
    <p xml:id="subtitle6a" begin="28.0s" end="34.6s" style="s2Left">
      But how is it proved?
    </p>
    <p xml:id="subtitle6b" begin="28.0s" end="34.6s" style="s1Right">
      Thus: what we call
    </p>
    <p xml:id="subtitle7" begin="34.6s" end="45.0s" style="s1Right">
      the vertex of the Brain<br/>
      is really its base
    </p>
    <p xml:id="subtitle8" begin="45.0s" end="52.0s" style="s1Right">
      and what we call its base<br/>
      is really its vertex,
    </p>
    <p xml:id="subtitle9a" begin="53.5s" end="58.7s">
      it is simply a question of nomenclature.
    </p>
    <p xml:id="subtitle9b" begin="53.5s" end="58.7s" style="s2">
      How truly delightful!
    </p>
  </div>
</body>
```

The first subtitle **Subtitle 1 – Time Interval [0.76, 3.45)** is presented during the time interval 0.76 to 3.45 seconds. This subtitle inherits its font family, font size, foreground color, and text alignment from the region into which it is presented. Since no region is explicitly specified on the paragraph, the nearest ancestor that specifies a region determines the targeted region. Note also that content is presented at the bottom (after edge) of the containing region due to the `tts:displayAlign="after"` being specified on the region definition.

> **Note:**
>
> The notation "[*X*,*Y*]" denotes a closed interval from *X* to *Y*, including *X* and *Y*; "[*X*,*Y*)" denotes a right half-open interval from *X* to *Y*, including *X* but not including *Y*; "(*X*,*Y*]" denotes a left half-open interval from *X* to *Y*, not including *X* but including *Y*; "(*X*,*Y*)" denotes an open interval from *X* to *Y*, not including *X* or *Y*.

> **Note:**
>
> In this example, the `p` element is used as a presentational element rather than as a semantic element, i.e., as a linguistic paragraph. It is up to an author to determine which TTML elements are used to convey the intended meaning. For instance, this example could be written to use timing on `span` elements in order to preserve the integrity of semantic paragraphs.

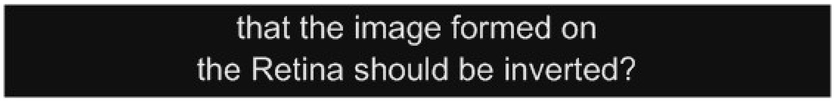
Subtitle 1 – Time Interval [0.76, 3.45)



It seems a paradox, does it not,

The second subtitle continues with the default style, except that it contains two lines of text with an intervening author-specified line break. Note the effects of the use of `tts:textAlign="center"` to specify the paragraph's alignment in the inline progression direction.

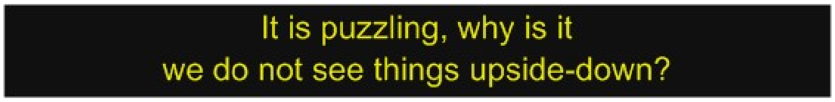
Subtitle 2 – Time Interval [5.0, 10.0)



that the image formed on
the Retina should be inverted?

The third subtitle continues, using a variant style which overrides the default style's foreground color with a different color.

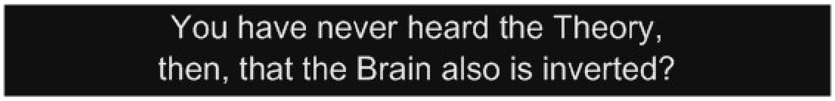
Subtitle 3 – Time Interval [10.0, 16.0)



It is puzzling, why is it
we do not see things upside-down?

The fourth subtitle reverts to the default style.

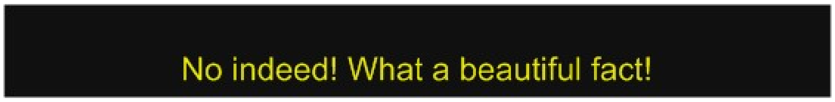
Subtitle 4 – Time Interval [17.2, 23.0)



You have never heard the Theory,
then, that the Brain also is inverted?

The fifth subtitle continues, again using a variant style which overrides the default style's foreground color with a different color.

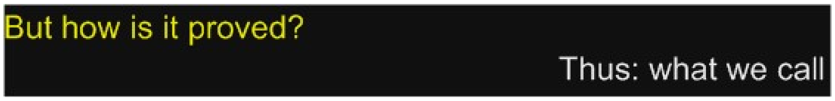
Subtitle 5 – Time Interval [23.0, 27.0)



No indeed! What a beautiful fact!

During the next active time interval, two distinct subtitles are simultaneously active, with the paragraphs expressing each subtitle using different styles that override color and paragraph text alignment of the default style. Note that the flow order is determined by the lexical order of elements as they appear in the content hierarchy.

Subtitles 6a and 6b – Time Interval [28.0, 34.6)

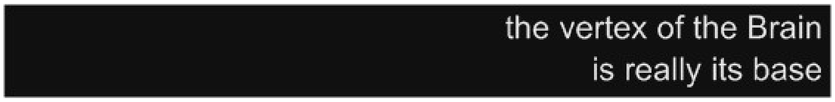


But how is it proved?

Thus: what we call

The next subtitle is specified in a similar manner using a style override to give the paragraph right (end) justification in the inline progression direction.

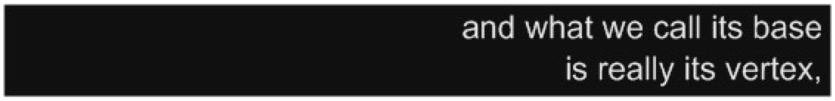
Subtitle 7 – Time Interval [34.6, 45.0)



the vertex of the Brain
is really its base

The eighth subtitle uses the same style override as the previous subtitle in order to maintain the right (end) justification of the paragraph.

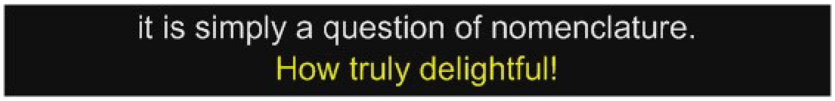
Subtitle 8 – Time Interval [47.3, 49.0)



and what we call its base
is really its vertex,

During the final (ninth) active time interval, two distinct subtitles are again simultaneously active, but with a different style applied to the second paragraph to override the default color. Note that the flow order is determined by the lexical order of elements as they appear in the content hierarchy.

Subtitles 9a and 9b – Time Interval [53.5, 58.7)



it is simply a question of nomenclature.
How truly delightful!

The examples shown above demonstrate the primary types of information that may be authored using TTML: metadata, styling, layout, timing, and content. In typical cases, styling and layout information are separately specified in a *Document Instance*. Content information is expressed in a hierarchical fashion that embodies the organization of both spatial (flow) and timing information. Content makes direct or indirect references to styling and layout information and may specify inline overrides to styling.

2 Definitions

2.1 Acronyms

DFXP	Distribution Format Exchange Profile
TT	Timed Text
TTML	Timed Text Markup Language
TTAF	Timed Text Authoring Format
TTWG	Timed Text Working Group

2.2 Terminology

Abstract Document Instance

An instance of an abstract data set as represented by a *Reduced XML Infoset*.

Abstract Document Type
A set of constraints that defines a class of *XML Information Sets* [XML InfoSet].

Attribute Information Item
Each specified or defaulted attribute of an XML document corresponds with an attribute information item as defined by [XML InfoSet], §2.3.

Character Information Item
Each data character appearing in an XML document corresponds with a character information item as defined by [XML InfoSet], §2.6.

Computed Cell Size
The size (extent) of a cell computed by dividing the width of the *Root Container Region* by the column count, i.e., the number of cells in the horizontal axis, and by dividing the height of the *Root Container Region* by the row count, i.e., the number of cells in the vertical axis, where the column and row counts are determined by the `ttp:cellResolution` parameter attribute.

Content Processor
A processing system capable of importing (receiving) Timed Text Markup Language content for the purpose of transforming, presenting, or otherwise processing the content.

Content Region
A logical region into which rendered content is placed when modeling or performing presentation processing.

Document Instance
A concrete realization of a Timed Text Markup Language document, where the concrete form is specific to the context of reference. For example, a sequence of bytes that represents an XML serialization of a Timed Text document, an internal, parsed representation of such a Timed Text document, etc.

Document Interchange Context
The implied context or environment external to a *Content Processor* in which document interchange occurs, and in which out-of-band protocols or specifications may define certain behavioral defaults, such as an implied profile.

Document Processing Context
The implied context or environment internal to a *Content Processor* in which document processing occurs, and in which out-of-band protocols or specifications may define certain behavioral defaults, such as the establishment or creation of a *Synthetic Document Syncbase*.

Element Information Item
Each element appearing in an XML document corresponds with an element information item as defined by [XML InfoSet], §2.2.

Exchange Profile
A profile of content that serves a set of needs for content interchange.

Extension
A syntactic or semantic expression or capability that is defined and labeled (using a extension designation) in another (public or private) specification.

Feature
A syntactic or semantic expression or capability that is defined and labeled (using a feature designation) in this specification (or a future revision of this specification).

Presentation Processor
A *Content Processor* which purpose is to layout, format, and render, i.e., to present, *Timed Text Markup Language* content by applying the presentation semantics defined in this specification.

Processor
See *Content Processor*.

Profile Definition Document
A document that defines a specific collection of features and extensions for which support is required or optional in a recipient content processor.

Region
A logical construct that models authorial intention regarding desired or potential presentation processing, and which is represented as a rectangular area of a presentation surface, i.e., a *region area*, into which content is composed and rendered during presentation processing.

Reduced XML Infoset
An XML Information Set [XML InfoSet] that satisfies the constraints specify by **A Reduced XML Infoset**.

Related Media Object
A (possibly null) media object associated with or otherwise related to a *Document Instance*. For example, an aggregate audio/video media object for which a *Document Instance* provides caption or subtitle information, and with which that *Document Instance* is associated.

Related Media Object Region
When a non-null *Related Media Object* exists, the region of this media object, expressed in the coordinate system that applies to the *Document Instance* that is associated with the related media object.

Root Container Region
A logical region that establishes a coordinate system into which *Document Instance* content regions are placed and optionally clipped.

Root Temporal Extent
The temporal extent (interval) defined by the temporal beginning and ending of a *Document Instance* in relationship with some external application or presentation context.

SMPTE Time Code
A time code whose format and semantics are established by [SMPTE 12M], which may be embedded into or otherwise associated with media content, such as a broadcast audio/video stream.

Synthetic Document Syncbase
A document level syncbase [SMIL 2.1], § 10.7.1, synthesized or otherwise established by the *Document Processing Context* in accordance with the *Related Media Object* or other processing criteria.

Synthetic SMPTE Document Syncbase
A *Synthetic Document Syncbase* constructed from *SMPTE Time Code* values embedded in or associated with the *Related Media Object* or otherwise determined by the *Document Processing Context*.

Temporally Active
A syntactic or semantic feature, e.g., an element or the presentation of an element, is *Temporally Active* when the current time of selected time base intersects with the active time interval of the feature.

Temporally Active Region
A *Region* that is *Temporally Active*.

Timed Text
Textual information that is intrinsically or extrinsically associated with timing information.

Timed Text Markup Language
A content type that represents timed text media for the purpose of interchange among authoring systems.

Timed Text Authoring System
A content authoring system capable of importing and exporting Timed Text Markup Language content.

Transformation Processor
A *Content Processor* which purpose is to transform or otherwise rewrite *Timed Text Markup Language* content to either *Timed Text Markup Language* or to another (arbitrary) content format. An example of the first is a processor that removes or rewrites TTML features so as to conform to a profile of TTML. An example of the latter is a processor that

translates TTML into a completely different timed text format. Because this specification does not otherwise define a target profile or format for transformation processing, no further transformation semantics are defined by this specification.

Valid Abstract Document Instance

An *Abstract Document Instance* which has been assessed for validity and found to be valid as defined by **4 Document Types**.

2.3 Documentation Conventions

Within normative prose in this specification, the words *may*, *should*, and *must* are defined as follows:

may

Conforming documents and/or TTML processors are permitted to, but need not behave as described.

should

Conforming documents and/or TTML processors are strongly recommended to, but need not behave as described.

must

Conforming documents and/or TTML processors are required to behave as described; otherwise, they are in error.

If normative specification language takes an imperative form, then it is to be treated as if the term **must** applies. Furthermore, if normative language takes a declarative form, and this language is governed by **must**, then it is also to be treated as if the term **must** applies.

> **Note:**
>
> For example, the phrases "treat X as an error" and "consider X as an error" are to be read as mandatory requirements in the context of use. Similarly, if the specification prose is "X must apply", "X applies", or "X is mandatory", and "X" is further defined as "X is Y and Z", then, by transitive closure, this last declarative phrase is to be read as "Y is mandatory" and "Z is mandatory" in the context of use.

All normative syntactic definitions of XML representations and other related terms are depicted with a light yellow-orange background color and labeled as "XML Representation" or "Syntax Representation", such as in the following:

XML Representation – Element Information Item: example

```
<example
  count = integer
  size = (large|medium|small) : medium>
  Content: (all | any*)
</example>
```

In an XML representation, bold-face attribute names (e.g. **count** above) indicate a required attribute information item, and the rest are optional. Where an attribute information item has an enumerated type definition, the values are shown separated by vertical bars, as for `size` above; if there is a default value, it is shown following a colon. Where an attribute information item has a built-in simple type definition defined in [XML Schema Part 2], a hyperlink to its definition therein is given.

In an XML representation, the expression *{any attribute not in default or any TT namespace}* applies only to namespace qualified attributes; unqualified attributes are not permitted unless explicitly defined in this specification.

The allowed content of the information item is shown as a grammar fragment, using the Kleene operators `?`, `*` and `+`. Each element name therein is a hyperlink to its own illustration.

The term linear white-space (LWSP) is to be interpreted as a non-empty sequence of SPACE (U+0020), TAB (U+0009), CARRIAGE RETURN (U+000D), or LINE FEED (U+000A), which corresponds to production [3] `S` as defined by [XML

1.0].

All content of this specification that is not explicitly marked as non-normative is considered to be normative. If a section or appendix header contains the expression "Non-Normative", then the entirety of the section or appendix is considered non-normative.

All paragraphs marked as a **Note** are considered non-normative.

Example code fragments are depicted with a light blue-green background color and labeled as "Example Fragment", such as in the following:

Example Fragment – Sample

```
<tt xml:lang="" xmlns="http://www.w3.org/ns/ttml">
  <head>
    <metadata/>
    <styling/>
    <layout/>
  </head>
  <body/>
</tt>
```

3 Conformance

This section specifies the general conformance requirements for TTML Content and processors.

3.1 Content Conformance

A TTML *Document Instance* conforms to this specification if the following criteria are satisfied:

1. When transporting a *Document Instance* in a *Document Interchange Context* in which a Media Type [Media Types] identifies the content type of the interchanged *Document Instance*, then the specified media type is `application/ttml+xml` in conformance with [XML Media Types] § 7, with which an optional `profile` parameter may appear, the value of which conforms to a profile designator as defined by **5.2 Profiles**.
2. The *Document Instance* is or can be represented as a Reduced XML Infoset as defined by **A Reduced XML Infoset**.
3. The Reduced XML Infoset that corresponds to the *Document Instance* is or can be associated with one of the *Abstract Document Types* defined by **4 Document Types**.
4. The *Reduced XML Infoset* that corresponds to the *Document Instance* is a *Valid Abstract Document Instance* of the associated *Abstract Document Type*.
5. The *Reduced XML Infoset* satisfies all additional mandatory syntactic and semantic constraints defined by this specification. In addition, this Infoset should satisfy the web content accessibility guidelines specified by [WCAG].

3.2 Processor Conformance

3.2.1 Generic Processor Conformance

A TTML *Content Processor* conforms to this specification if the following generic processor criteria are satisfied:

1. The processor provides at least one mechanism for notionally instantiating a Reduced XML Infoset representation of a conformant *Document Instance*.
2. If a process does or can perform validation of a candidate *Document Instance*, then it provides at least one mechanism

to implicitly or explicitly associate the *Reduced XML Infoset* representation of a conformant *Document Instance* with one of the *Abstract Document Types* defined by **4 Document Types**.

3. The processor does not *a priori* reject or abort the processing of a conformant *Document Instance* unless the processor does not support some required (mandatory) feature specified or implied by a TTML profile declared to apply to the *Document Instance*.
4. The processor supports all mandatory processing semantics defined by this specification.

> **Note:**
>
> The phrase *mandatory semantics* refers to all explicit use of the conformance key phrases **must** and **must not** as well as any declarative statement that can be reasonably inferred from such key phrases. For example, these mandatory semantics include support for all features marked as mandatory in **D.2 Feature Support**.

5. If the processor supports some optional processing semantics defined by this specification, then it does so in a manner consistent with the defined semantics.

> **Note:**
>
> The phrase *optional semantics* refers to all explicit use of the conformance key phrases **should**, **should not**, **may**, and **may not**, as well as any declarative statement that can be reasonably inferred from such key phrases. For example, these optional semantics include support for all features marked as optional in **D.2 Feature Support**.

3.2.2 Transformation Processor Conformance

A TTML *Content Processor* is a conformant TTML *Transformation Processor* if the following criteria are satisfied:

1. The processor satisfies all requirements specified by **3.2.1 Generic Processor Conformance**.
2. The processor supports the DFXP Transformation profile as specified by **F.1 DFXP Transformation Profile**.

3.2.3 Presentation Processor Conformance

A TTML *Content Processor* is a conformant TTML *Presentation Processor* if the following criteria are satisfied:

1. The processor satisfies all requirements specified by **3.2.1 Generic Processor Conformance**.
2. The processor supports the DFXP Presentation profile as specified by **F.2 DFXP Presentation Profile**.

3.3 Claims

Any claim of compliance with respect to the conformance of a TTML *Document Instance* or *Content Processor* must make reference to an implementation compliance statement (ICS).

An implementation compliance statement must identify all mandatory and optional features of this specification that are satisfied by the document instance or the content processor implementation. In particular, the statement must identify the utilized or supported TTML vocabulary profile(s) as defined by **5.2 Profiles**, and, if a subset or superset profile is used or supported, then what features are excluded or included in the subset or superset profile.

A *Document Instance* for which a compliance claim is made must specify either (1) a `ttp:profile` attribute on its root `tt` element as defined by **6.2.8 ttp:profile** or (2) a `ttp:profile` element as a child of the `head` element as defined by **6.1.1 ttp:profile**.

4 Document Types

This section defines the following TTML *Abstract Document Types*:

- **4.1 TTML Content**

Each *Abstract Document Type* consists of the following constraints:

- a non-empty collection of element types, where each element type consists of a name, a (possibly empty) collection of attributes, and a content specification
- a non-empty collection of element types that may appear as the document element

An *Abstract Document Instance* may be assessed in terms of validity, and is considered to be a *Valid Abstract Document Instance* if it satisfies the following condition: if after

1. pruning all element information items whose names are not members of the collection of element types defined by the associated *Abstract Document Type*, then
2. pruning character information item children from any remaining element in case that all character children of the element denote XML whitespace characters and the element's type is defined as empty in the associated *Abstract Document Type*, and then
3. pruning all attribute information items having expanded names such that the namespace URI of the expanded names are not listed in **Table 1 – Namespaces**, or, if listed in **Table 1 – Namespaces**, are not members of the collection of attributes defined by the associated *Abstract Document Type* for use with the owning element information item,

then the document element is one of the document element types permitted by the associated *Abstract Document Type*, the descendants of the document element satisfy their respective element type's content specifications, all required attributes are present, and the declared value of each attribute satisfies the type declared by the associated *Abstract Document Type*.

> **Note:**
>
> While a conformant processor may not *a priori* reject a conformant *Document Instance*, a given *Document Instance* may be constrained by the author or authoring tool to satisfy a more restrictive definition of validity.

Note:

As illustrated in the following example, an *Abstract Document Instance* can be a *Valid Abstract Document Instance* even if it includes elements and attributes whose namespace names are listed in **Table 1 – Namespaces** but whose local names are not part of the vocabulary defined by this version of the specification. Specifically, the element `foo` and the attribute `tts:foo` are pruned by above steps (1) and (3), respectively, because they are not members of the associated *Abstract Document Type*, even though their namespace names are listed in **Table 1 – Namespaces**.

Example Fragment – Pruning of elements and attributes that are not members of the associated Abstract Document Type

```
<tt
  xmlns="http://www.w3.org/ns/ttml"
  xmlns:tts="http://www.w3.org/ns/ttml#styling"
  xml:lang="en">
  <body>
    <foo>Foo</foo>
    <div>
      <p
        tts:foo="bar">
          Bar
      </p>
    </div>
  </body>
</tt>
```

4.1 TTML Content

TTML Content is an *Abstract Document Type* of a profile of the Timed Text Markup Language intended to be used for interchange among distribution systems. This document type is defined in terms of the element and attribute vocabulary specified in **5 Vocabulary**.

This specification references two types of schemas that may be used to validate a superset/subset of conformant TTML Content *Document Instances*:

- **B.1 Relax NG Compact (RNC) Schema**
- **B.2 XML Schema Definition (XSD) Schema**

The (root) document element of a TTML Content *Document Instance* must be a `tt` element, as defined by **7.1.1 tt**.

Note:

The schemas referenced by this specification do not validate all syntactic constraints defined by this specification, and, as such, represent a superset of conformant TTML Content. In particular, performing validation with one of the above referenced schemas may result in a *false positive* indication of validity. For example, both the RNC and XSD schemas specify that a `tts:fontFamily` attribute must satisfy the `xs:string` XSD data type; however, this data type is a superset of the values permitted to be used with the `tts:fontFamily` attribute.

In addition, the RNC schema may produce a *false negative* indication of validity when using the `xml:id` attribute with an element in a foreign namespace, thus representing a subset of conformant TTML Content. This is due to a specific limitation in expressing wildcard patterns involving `xsd:ID` typed attributes in Relax NG schemas. Note that this specification defines the formal validity of a *Document Instance* to be based on an *Abstract Document Instance* from which all foreign namespace elements and attributes have been removed. Therefore, the exceptional reporting of this false negative does not impact the formal assessment of *Document Instance* validity.

> **Note:**
>
> The schemas referenced by this specification are intended for use after the pruning steps (1)-(3) specified by **4 Document Types** have been applied.

5 Vocabulary

This section defines the namespaces, profiles, and vocabulary (as an element and attribute catalog) of the Timed Text Markup Language (TTML) as follows:

- **5.1 Namespaces**
- **5.2 Profiles**
- **5.3 Catalog**

5.1 Namespaces

The Timed Text Markup Language (TTML) employs a number of XML Namespaces [XML Namespaces 1.0] for elements and certain global attributes. The following table specifies this set of namespaces and indicates the default prefix used within this specification and the normative URI that denotes each namespace.

> **Note:**
>
> In a specific *Document Instance*, it is not required that the default prefixes shown below are used. Any prefix or namespace binding that satisfies the constraints of XML Namespaces [XML Namespaces 1.0] may be used that is associated with the specified namespace URI.

Table 1 – Namespaces

Name	Prefix	Value
TT	`tt:`	`http://www.w3.org/ns/ttml`
TT Parameter	`ttp:`	`http://www.w3.org/ns/ttml#parameter`
TT Style	`tts:`	`http://www.w3.org/ns/ttml#styling`
TT Metadata	`ttm:`	`http://www.w3.org/ns/ttml#metadata`
TT Profile	*none*	`http://www.w3.org/ns/ttml/profile/`
TT Feature	*none*	`http://www.w3.org/ns/ttml/feature/`
TT Extension	*none*	`http://www.w3.org/ns/ttml/extension/`

> **Note:**
>
> If a reference to an element type is used in this specification and the name of the element type is not namespace qualified, then the TT Namespace applies.
>
> For certain namespaces defined above, the default prefix is specified as *none* if no XML vocabulary is defined in the namespace by this specification (nor expected to be defined in a future version of this specification). In such cases, the use of the namespace URI is for purposes other than defining XML vocabulary, e.g., for designating profiles, features, extensions and for dereferencing standard profile definitions.

All TTML Namespaces are *mutable* [NSState]; all undefined names in these namespaces are reserved for future standardization by the W3C.

5.2 Profiles

The Timed Text Markup Language (TTML) employs a number of standard, predefined profiles of its vocabulary and associated semantics. The following table specifies this set of profiles, indicating a normative name and designator for each predefined profile, and where each of these profiles is formally elaborated in **F Profiles** or in another TTWG specification.

Table 2 – Profiles

Name	Designator
DFXP Transformation	`http://www.w3.org/ns/ttml/profile/dfxp-transformation`
DFXP Presentation	`http://www.w3.org/ns/ttml/profile/dfxp-presentation`
DFXP Full	`http://www.w3.org/ns/ttml/profile/dfxp-full`
SDP US	`http://www.w3.org/ns/ttml/profile/sdp-us`

A profile designator must adhere to the `xsd:anyURI` data type defined by [XML Schema Part 2], §3.2.17. If the profile designator is expressed as a relative URI, then it must be absolutized by using the TT Profile Namespace value as the base URI.

> **Note:**
>
> For example, if a profile designator is expressed as `dfxp-presentation`, then the absolutized profile designator would be `http://www.w3.org/ns/ttml/profile/dfxp-presentation`.

All profile designators which have the TT Profile Namespace as a prefix but are otherwise not listed in **Table 2 – Profiles** are reserved for future standardization, and must not appear in a conformant *Document Instance*. Nothwithstanding this constraint, a profile designator is not restricted to the set of designators enumerated in **Table 2 – Profiles**, but may be any URI that feasibly dereferences a TTML *Profile Definition Document* provided it does not use the TT Profile Namespace as a prefix.

The profile of TTML that must be supported by a TTML *Content Processor* in order to process a *Document Instance* is determined either (1) by specifying a `ttp:profile` attribute on the root `tt` element, as defined by **6.2.8 ttp:profile**, or (2) by including one or more `ttp:profile` elements in the `head` element, in accordance with **6.1.1 ttp:profile**.

If a `ttp:profile` element appears as a descendant of the `tt` element, then the `ttp:profile` attribute should not be specified on the `tt` element. If both a `ttp:profile` element and a `ttp:profile` attribute are present (in a given *Document Instance*), then the `ttp:profile` attribute must be ignored for the purpose of determining the declared profile requirements.

If more than one `ttp:profile` element appears in a *Document Instance*, then all specified profiles apply simultaneously. In such a case, if some feature or some extension is specified by one profile to be `used` (mandatory and enabled) and by another profile to be `required` (mandatory) or `optional` (voluntary), then that feature or extension must be considered to be `used` (mandatory and enabled); if some feature or some extension is specified by one profile to be merely `required` (mandatory) and by another profile to be `optional` (voluntary), then that feature or extension must be considered to be `required` (mandatory).

If neither `ttp:profile` attribute nor `ttp:profile` element is present in a *Document Instance*, and if the *Document Interchange Context* does not make an implicit or explicit reference to a pre-defined profile or does not specify a *Profile Definition Document* or another equivalent set of feature designations, then the DFXP Transformation profile applies.

> **Note:**
>
> It is not a requirement on a conformant *Document Instance* that a profile be internally defined by use of a `ttp:profile` element or internally referenced by a `ttp:profile` attribute. More specifically, it is permitted that the *Document Interchange Context* determines the applicable profile through private agreement, out-of-band protocol, or common use (between sender and receiver) of a profile defined by an external specification.

> **Note:**
>
> It is intended that the `ttp:profile` attribute be used when the author wishes to reference one of the standard, predefined profiles of TTML Content, and does not wish to modify (by supersetting or subsetting) that profile. This attribute may also be used by an author to indicate the use of a non-standard profile, in which case the specified profile designator expresses a URI that denotes an externally defined *Profile Definition Document*. However, it is not required that a conformant TTML Content Processor be able to dereference such an externally specified profile definition.
>
> In contrast, it is intended that the `ttp:profile` element be used when the author wishes to make use of a modified predefined profile or wishes to include in the *Document Instance* a non-standard profile definition not based upon one of the predefined profiles.
>
> A predefined profile is supersetted by specifying some feature or extension to be `required` (mandatory) that was either not specified in the underlying, baseline profile or was specified as `optional` (voluntary) in the baseline profile. A predefined profile is subsetted by specifying some feature or extension to be `optional` (voluntary) that was specified as `required` (mandatory) in the underlying, baseline profile.
>
> When a baseline profile is modified by subsetting, the resulting, derived profile is referred to as a *subtractive* profile; when modified by supersetting, the result is referred to as an *additive* profile. It is also possible to define a derived profile that is simultaneously subtractive and additive.

If a *Document Instance* makes use of a feature defined by **D.1 Feature Designations** and if the intended use of the document requires the recognition and processing of that feature, then the document must include a *required feature* or a *used feature* specification in one of its declared or referenced profiles. If a *Document Instance* makes use of an extension designatable by **E.1 Extension Designations** and if the intended use of the document requires the recognition and processing of that extension, then the document must include a *required extension* or a *used extension* specification in one of its declared or referenced profiles.

> **Note:**
>
> A *required feature* or *used feature* specification is expressed directly (or indirectly by referring to a profile) by means of a `ttp:feature` element where the value of its `value` attribute is `required` or `use`, respectively. A *required extension* or *used extension* specification is expressed directly (or indirectly by referring to a profile) by means of a `ttp:extension` element where the value of its `value` attribute is `required` or `use`, respectively.

An example of an author defined additive, derived profile of the DFXP Presentation profile is shown below in **Example Fragment – DFXP Additive Profile**.

Example Fragment – DFXP Additive Profile

```
<tt xml:lang="" xmlns="http://www.w3.org/ns/ttml">
 <head>
   <profile use="dfxp-presentation" xmlns="http://www.w3.org/ns/ttml#parameter">
     <features xml:base="http://www.w3.org/ns/ttml/feature/">
       <feature value="required">#fontStyle-italic</feature>
     </features>
   </profile>
 </head>
 <body/>
</tt>
```

> **Note:**
>
> In the above example, the baseline profile is declared to be the DFXP Presentation profile, which is then additively modified by making the `#fontStyle-italic` feature required (rather than optional as it is defined in **F.2 DFXP Presentation Profile**). Note also the resetting of the default XMLNS binding on the `profile` element to the TT Parameter Namespace.

5.3 Catalog

The vocabulary of the Timed Text Markup Language (TTML) is defined in the following major catalogs (divisions of vocabulary):

- **5.3.1 Core Catalog**
- **5.3.2 Extension Catalog**

The core catalog defines the baseline, core vocabulary of TTML, and, in particular, the vocabulary of TTML Content. The extension catalog serves as a placeholder for extensions to the core vocabulary defined by TTML.

5.3.1 Core Catalog

The core vocabulary catalog is intended to satisfy the needs of TTML while providing a baseline vocabulary for future profiles. This vocabulary is divided into distinct categories, specified in detail in the following sections:

- **6 Parameters**
- **7 Content**
- **8 Styling**
- **9 Layout**
- **10 Timing**
- **11 Animation**
- **12 Metadata**

The core element vocabulary specified for use with a *Document Instance* is enumerated in **Table 3 – Element Vocabulary**.

Table 3 – Element Vocabulary

Module	Elements
Animation	set
Content	body, div, p, span, br
Document	tt
Head	head
Layout	layout, region
Metadata	metadata
Metadata Items	ttm:actor, ttm:agent, ttm:copyright, ttm:desc, ttm:name, ttm:title
Parameter Items	ttp:profile, ttp:features, ttp:feature, ttp:extensions, ttp:extension
Styling	styling, style

Element vocabulary groups that are used in defining content models for TTML element types are enumerated in **Table 4 – Element Vocabulary Groups**.

Table 4 – Element Vocabulary Groups

Group	Elements
Animation.class	set
Block.class	div | p
Inline.class	span | br | #PCDATA
Metadata.class	metadata | ttm:agent | ttm:copyright | ttm:desc | ttm:title
Parameters.class	ttp:profile

The attribute vocabulary specified for use with the core vocabulary catalog is enumerated in **Table 5 – Attribute Vocabulary**.

Table 5 – Attribute Vocabulary

Module	Attributes
Core Attributes	xml:id, xml:lang, xml:space
Layout	region
Metadata Attributes	ttm:agent, ttm:role
Parameter Attributes	ttp:cellResolution, ttp:clockMode, ttp:dropMode, ttp:frameRate, ttp:frameRateMultiplier, ttp:markerMode, ttp:pixelAspectRatio, ttp:profile, ttp:subFrameRate, ttp:tickRate, ttp:timeBase
Styling	style
Styling Attributes	tts:backgroundColor, tts:color, tts:direction, tts:display, tts:displayAlign, tts:extent, tts:fontFamily, tts:fontSize, tts:fontStyle, tts:fontWeight, tts:lineHeight, tts:opacity, tts:origin, tts:overflow, tts:padding, tts:showBackground, tts:textAlign, tts:textDecoration, tts:textOutline, tts:unicodeBidi, tts:visibility, tts:wrapOption, tts:writingMode, tts:zIndex
Timing Attributes	begin, dur, end, timeContainer

> **Note:**
>
> Only those attributes defined as either (1) global, i.e., namespace qualified, or (2) shared element-specific, i.e., not namespace qualified but shared across multiple element types, are listed in **Table 5 – Attribute Vocabulary** above.

> **Note:**
>
> All vocabulary defined by TTML consistently makes use of the so-called *lowerCamelCase* naming convention. In some cases, this results in the change of a name when the name was based upon another specification that used a different naming convention.

5.3.2 Extension Catalog

The extension vocabulary catalog is intended for use by future profiles of TTML, and is not further defined by this version of this specification.

In addition to standardized extension vocabulary, a conforming *Document Instance* may contain arbitrary namespace qualified elements that reside in any namespace other than those namespaces defined for use with this specification. Furthermore, a conforming *Document Instance* may contain arbitrary namespace qualified attributes on TTML defined vocabulary where such attributes reside in any namespace other than those defined for use with this specification.

6 Parameters

This section specifies the *parameters* matter of the core vocabulary catalog, where parameters are to be understood as information that is either (1) essential or (2) of significant importance for the purpose of interpreting the semantics of other types of information expressed by core vocabulary items or for establishing a *Document Processing Context* by means of which TTML Content can be related to an external environment.

6.1 Parameter Element Vocabulary

The following elements, all defined in the TT Parameter Namespace, specify parametric information that applies to a *Document Instance* or *Content Processor*:

- **6.1.1 ttp:profile**
- **6.1.2 ttp:features**
- **6.1.3 ttp:feature**
- **6.1.4 ttp:extensions**
- **6.1.5 ttp:extension**

6.1.1 ttp:profile

The `ttp:profile` element is used to specify a collection of used (mandatory and enabled), required (mandatory), and optional (voluntary) features and extensions that must or may be supported by a *Content Processor* in order to process a *Document Instance* that makes (or may make) use of such features and extensions.

> **Note:**
>
> The difference between a *feature* and an *extension* is where it is defined and how it is labeled: if defined in this specification (or a future revision thereof) and labeled with a feature designation in **D Features**, then it is considered to be a feature; if defined in another specification and labeled there with an extension designation, then it is considered to be an extension. In general, features are expected to be defined by the W3C standards process, while extensions are expected to be defined by third parties.

This specification defines two distinct uses of the `ttp:profile` element:

- as a child of the `head` element within a TTML *Document Instance*;
- as the root element of a TTML *Profile Definition Document* instance;

When a `ttp:profile` element appears within a TTML *Document Instance*, its purpose is to express authorial intentions about which features and extensions must or may be supported by a recipient content processor. In addition, the element indirectly expresses information about the set of features or extensions that are (or may expected to be) used by the *Document Instance*.

When a `ttp:profile` element is used by a TTML *Profile Definition Document* instance, it serves to publish a machine readable specification of a specific TTML profile, of which this specification defines three such *Profile Definition Documents* in **F Profiles**.

The `ttp:profile` element accepts as its children zero or more elements in the `Metadata.class` element group, followed by zero or more `ttp:features` elements, followed by zero or more `ttp:extensions` elements.

XML Representation – Element Information Item: ttp:profile

```
<ttp:profile
  use = string
  xml:id = ID
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*, ttp:features*, ttp:extensions*
</ttp:profile>
```

If specified, the `use` attribute must adhere to the `xsd:anyURI` data type defined by [XML Schema Part 2], §3.2.17, and, furthermore, must denote a profile designator in accordance with **5.2 Profiles**. In this case, the profile designator must refer to (1) a standard, predefined *Profile Definition Document* as defined by **F Profiles**, or (2) a feasibly dereferenceable resource representing a valid *Profile Definition Document* instance. In either case, the referenced profile serves as the baseline profile of the specifying `ttp:profile` element.

If the `use` attribute is not specified, then the baseline profile of the `ttp:profile` element must be considered to be the empty (null) profile, i.e., a profile definition containing no feature or extension specifications.

The collection of features and extensions of a profile are determined according to the following ordered rules:

1. initialize the features and extensions of the profile to the empty set;
2. if a `use` attribute is present, then augment the profile with the set of features and extensions specified by the referenced baseline profile;
3. for each `ttp:feature` and `ttp:extension` element descendant of the `ttp:profile` element, using a post-order traversal, merge the specified feature or extension with the features and extensions of the profile, where merging a feature or extension entails replacing an existing feature or extension specification, if it already exists, or adding a new feature or extension specification, if it does not yet exist in the profile;

A conformant TTML processor is not required to be able to dereference a *Profile Definition Document* that is not one of the standard, predefined profiles defined by **F Profiles**. Furthermore, a conformant TTML processor may make use of a built-in, static form of each standard, predefined profile so as not to require dereferencing a network resource.

If a TTML processor is unable to dereference a non-standard *Profile Definition Document*, then it must not further process the document without the presence of an explicit override from an end-user or some implementation specific parameter traceable to an end-user or to a user or system configuration setting. If a TTML processor aborts processing of a *Document Instance* due to the inability to reference a non-standard *Profile Definition Document*, then some end-user notification should be given unless the end-user or system has disabled such a notification, or if the processor does not permit or entail the intervention of an end-user.

The `ttp:profile` element is illustrated by the following example.

Example Fragment – ttp:profile

```
<ttp:profile use="dfxp-presentation">
  <ttp:features xml:base="http://www.w3.org/ns/ttml/feature/">
    <ttp:feature>#text-outline</ttp:feature>
  </ttp:features>
</ttp:profile>
```

> **Note:**
>
> In the above example, the DFXP presentation profile is used as the baseline profile. This baseline profile is then supersetted (thus creating an additive derived profile) by requiring support for `#text-outline` feature.

6.1.2 ttp:features

The `ttp:features` element is a container element used to group infomation about feature support requirements.

The `ttp:features` element accepts as its children zero or more elements in the `Metadata.class` element group, followed by zero or more `ttp:feature` elements.

XML Representation – Element Information Item: ttp:features

```
<ttp:features
  xml:base = string : TT Feature Namespace
  xml:id = ID
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*, ttp:feature*
</ttp:features>
```

If specified, the `xml:base` attribute must (1) adhere to the `xsd:anyURI` data type defined by [XML Schema Part 2], §3.2.17, (2) express an absolute URI that adheres to [XML Base] and, (3) express a feature namespace as defined by **D.1 Feature Designations**. If not specified, the `xml:base` attribute's default value applies, which is the TT Feature Namespace.

The `xml:base` attribute is used to permit the abbreviation of feature designation URIs expressed by child `ttp:feature` elements.

6.1.3 ttp:feature

The `ttp:feature` element is used to specify infomation about support requirements for a particular feature.

The children of the `ttp:feature` element must express a non-empty sequence of character information items that adheres to the `xsd:anyURI` data type defined by [XML Schema Part 2], §3.2.17.

XML Representation – Element Information Item: ttp:feature

```
<ttp:feature
  value = (optional|required|use) : required
  xml:id = ID
  {any attribute not in default or any TT namespace}>
  Content: #PCDATA
</ttp:feature>
```

If the URI expressed by the content of the `ttp:feature` element is a relative URI, then, when combined with the feature namespace value expressed by the `xml:base` attribute of the nearest ancestor `ttp:features` element, it must express an absolute URI. In either case (original absolute URI or resulting absolutized URI), the URI expressed by the `ttp:feature` element must further adhere to the syntax of a feature designation as defined by **D.1 Feature Designations**, and, furthermore, the specific designation that appears in this URI, i.e., the portion of the feature designation that starts with the fragment identifier separator '#', must be defined by this specification or some published version thereof (that has achieved REC status).

If the URI expressed by the content of the `ttp:feature` element is a relative URI, then an `xml:base` attribute should be specified on the nearest ancestor `ttp:features` element.

The `value` attribute specifies whether a conforming TTML processor must or may implement the designated feature in order to process the document. If the value of the `value` attribute is `optional`, then the processor need not implement or otherwise support the feature in order to process the document; if the value is `required`, then the processor must implement or otherwise support the feature, irrespective of whether the feature is enabled or disabled, in order to process the document; if the value is `use`, then the processor must both (1) implement or otherwise support the feature and (2) have enabled (activated) use of the feature.

> **Note:**
>
> The default value of the `value` attribute is `required`, as indicated in the above element information item definition. Therefore, if a `value` attribute is not specified on a `ttp:feature` element, it is equivalent to specifying that support for the feature is required.

If the value of the `value` attribute is `required` or `use` and the TTML processor implementation does not support the feature, or if the `value` attribute is `use` and the TTML processor implementation supports but has disabled that feature, then it must not further process the document without the presence of an explicit override from an end-user or some implementation specific parameter traceable to an end-user or to a user or system configuration setting. If a TTML processor aborts processing of a *Document Instance* due to the specification of a required, but unsupported feature by this element, then some end-user notification should be given unless the end-user or system has disabled such a notification, or if the processor does not permit or entail the intervention of an end-user.

If the value of the `value` attribute is `optional`, and if the TTML processor implementation does not support the feature, then it may further process the document even in the case that some use of the feature is present in the document. In the case of actual use of a feature designated as optional, the default semantics associated with that feature apply; that is, the processor may behave as if the feature were not actually used or referenced by the document. Notwithstanding the above, the syntactic presence or reference to an optional feature by a document must not be considered to be a violation of document validity or a barrier to further processing if the syntactic expression is well-formed and valid.

If some defined (i.e., standardized) or otherwise well known feature is not specified by a `ttp:feature` element in a given profile, then it must be interpreted as if the feature were specified with the `value` attribute equal to `optional`.

> **Note:**
>
> In particular, if some feature is not present in a profile definition, then it is not to be interpreted as meaning the use of that feature (in a *Document Instance*) is disallowed or otherwise prohibited.

The `ttp:feature` element is illustrated by the following example.

Example Fragment – ttp:feature

```
<ttp:profile use="http://www.w3.org/ns/ttml/profile/dfxp-presentation">
  <ttp:features xml:base="http://www.w3.org/ns/ttml/feature/">
    <ttp:feature value="required">#fontStyle-italic</ttp:feature>
    <ttp:feature value="required">#textDecoration-under</ttp:feature>
  </ttp:features>
</ttp:profile>
```

> **Note:**
>
> In the above example, the DFXP presentation profile is used as the baseline profile. This baseline profile is then modified by two `ttp:feature` elements in order to superset the baseline profile (since neither `#fontStyle-italic` nor `#textDecoration-under` are required by the DFXP presentation profile).
>
> The effect of this example is to express authorial intentions that italic font style and text underlining must be supported.

6.1.4 ttp:extensions

The `ttp:extensions` element is a container element used to group infomation about extension support requirements.

The `ttp:extensions` element accepts as its children zero or more elements in the `Metadata.class` element group, followed by zero or more `ttp:extension` elements.

XML Representation – Element Information Item: ttp:extensions

```
<ttp:extensions
  xml:base = string : TT Extension Namespace
  xml:id = ID
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*, ttp:extension*
</ttp:extensions>
```

If specified, the `xml:base` attribute must (1) adhere to the `xsd:anyURI` data type defined by [XML Schema Part 2], §3.2.17, (2) express an absolute URI that adheres to [XML Base] and, (3) express an extension namespace as defined by **E.1 Extension Designations**. If not specified, the `xml:base` attribute's default value applies, which is the TT Extension Namespace.

The `xml:base` attribute is used to permit the abbreviation of feature designation URIs expressed by child `ttp:extension` elements.

6.1.5 ttp:extension

The `ttp:extension` element is used to specify infomation about support requirements for a particular extension.

The children of the `ttp:extension` element must express a non-empty sequence of character information items that adheres to the `xsd:anyURI` data type defined by [XML Schema Part 2], §3.2.17.

XML Representation – Element Information Item: ttp:extension

```
<ttp:extension
  value = (optional|required|use) : required
  xml:id = ID
  {any attribute not in default or any TT namespace}>
  Content: #PCDATA
</ttp:extension>
```

If the URI expressed by the content of the `ttp:extension` element is a relative URI, then, when combined with the extension namespace value expressed by the `xml:base` attribute of the nearest ancestor `ttp:extensions` element, it must express an absolute URI. In either case (original absolute URI or resulting absolutized URI), the URI expressed by the `ttp:extension` element must further adhere to the syntax of an extension designation as defined by **E.1 Extension Designations**.

If the URI expressed by the content of the `ttp:extension` element is a relative URI, then an `xml:base` attribute should be specified on the nearest ancestor `ttp:extensions` element.

The `value` attribute specifies whether a conforming TTML processor must or may implement the designated extension in order to process the document. If the value of the `value` attribute is `optional`, then the processor need not implement or otherwise support the extension in order to process the document; if the value is `required`, then the processor must implement or otherwise support the extension in order to process the document; if the value is `use`, then the processor must both (1) implement or otherwise support the extension and (2) enable (activate) use of the extension.

> **Note:**
>
> The default value of the `value` attribute is `required`, as indicated in the above element information item definition. Therefore, if a `value` attribute is not specified on a `ttp:extension` element, it is equivalent to specifying that support for the extension is required.

If the value of the `value` attribute is `required` or `use` and the TTML processor implementation does not support the extension, or if the `value` attribute is `use` and the TTML processor implementation supports but has disabled that extension, then it must not further process the document without the presence of an explicit override from an end-user or some implementation specific parameter traceable to an end-user or to a user or system configuration setting. If a TTML processor aborts processing of a *Document Instance* due to the specification of a required, but unsupported extension by this element, then some end-user notification should be given unless the end-user or system has disabled such a notification, or if the processor does not permit or entail the intervention of an end-user.

If the value of the `value` attribute is `optional`, and if the TTML processor implementation does not support the extension, then it may further process the document even in the case that some use of the extension is present in the document. In the case of actual use of an extension designated as optional, the default semantics associated with that extension apply; that is, the processor may behave as if the extension were not actually used or referenced by the document. Notwithstanding the above, the syntactic presence or reference to an optional extension by a document must not be considered to be a violation of document validity or a barrier to further processing if the syntactic expression is well-formed and valid.

If some well known extension is not specified by a `ttp:extension` element in a given profile, then it must be interpreted as if the extension were specified with the `value` attribute equal to `optional`.

> **Note:**
>
> In particular, if some extension is not present in a profile definition, then it is not to be interpreted as meaning the use of that extension (in a *Document Instance*) is disallowed or otherwise prohibited.

The `ttp:extension` element is illustrated by the following example.

Example Fragment – ttp:extension

```
<ttp:profile use="http://www.w3.org/ns/ttml/profile/dfxp-transformation">
  <ttp:extensions xml:base="http://example.org/ttml/extension/">
    <ttp:extension value="use">#prefilter-by-language</ttp:extension>
  </ttp:extensions>
</ttp:profile>
```

> **Note:**
>
> In the above example, the DFXP transformation profile is used as the baseline profile. This baseline profile is then supersetted by specifying that support and use is required for a private extension defined in a third party namespace.
>
> The effect of this example is to express authorial intentions that a recipient processor must support the DFXP transformation profile and must also support and enable an extension defined by a third party.

6.2 Parameter Attribute Vocabulary

The following attributes are defined in the TT Parameter Namespace.

- **6.2.1 ttp:cellResolution**

- **6.2.2 ttp:clockMode**
- **6.2.3 ttp:dropMode**
- **6.2.4 ttp:frameRate**
- **6.2.5 ttp:frameRateMultiplier**
- **6.2.6 ttp:markerMode**
- **6.2.7 ttp:pixelAspectRatio**
- **6.2.8 ttp:profile**
- **6.2.9 ttp:subFrameRate**
- **6.2.10 ttp:tickRate**
- **6.2.11 ttp:timeBase**

Unless explicitly stated otherwise, linear white-space (LWSP) must appear between adjacent non-terminal components of a TT Parameter value unless some other delimiter is permitted and used.

6.2.1 ttp:cellResolution

The `ttp:cellResolution` attribute may be used by an author to express the number of horizontal and vertical cells into which the *Root Container Region* area is divided for the purpose of expressing presentation semantics in terms of a uniform grid.

If specified, the value of this attribute must adhere to the following syntax:

Syntax Representation – ttp:cellResolution

```
ttp:cellResolution
  : columns rows                            // columns != 0; rows != 0

columns | rows
  : <digit>+
```

If not specified, the number of columns and rows must be considered to be 32 and 15, respectively. If specified, then columns or rows must not be zero (0).

> **Note:**
>
> The choice of values 32 and 15 are based on this being the maximum number of columns and rows defined by [CEA-608-E].

A `ttp:cellResolution` attribute is considered to be significant only when specified on the `tt` element.

> **Note:**
>
> The use of a uniform grid is employed only for the purpose of measuring lengths and expressing coordinates. In particular, it is not assumed that the presentation of text or the alignment of individual glyph areas is coordinated with this grid. Such alignment is possible, but requires the use of a monospaced font and a font size whose EM square exactly matches the cell size.

Except where indicated otherwise, when a <length> expressed in cells denotes a dimension parallel to the inline or block progression dimension, the cell's dimension in the inline or block progression dimension applies, respectively.

> **Note:**
>
> For example, if padding (on all four edges) is specified as 0.1c, the cell resolution is 20 by 10, and the extent of the *Root Container Region* is 640 by 480, then, assuming top to bottom, left to right writing mode, the start and end padding will be (640 / 20) * 0.1 pixels and the before and after padding will be (480 / 10) * 0.1 pixels.

6.2.2 ttp:clockMode

The `ttp:clockMode` attribute is used to specify the interpretation of time expressions as real-time time coordinates when operating with time base of `clock` as defined by **6.2.11 ttp:timeBase**.

> **Note:**
>
> See **10.3 Time Value Expressions** for the specification of time expression syntax and semantics.

If specified, the value of this attribute must adhere to the following syntax:

Syntax Representation – ttp:clockMode

```
ttp:clockMode
  : "local"
  | "gps"
  | "utc"
```

If the time base, defined by **6.2.11 ttp:timeBase**, is designated as `clock`, then this parameter applies as follows: if the parameter's value is `local`, then time expressions are interpreted as local wall-clock time coordinates; if `utc`, then time expressions are interpreted as UTC time coordinates [UTC]; if `gps`, then time expressions are interpreted as GPS time coordinates [GPS].

> **Note:**
>
> The primary difference between GPS time and UTC time is that GPS time is not adjusted for leap seconds, while UTC time is adjusted as follows: UTC = TAI (*Temp Atomique International*) + *leap seconds accumulated since 1972*. TAI is maintained by the *Bureau International des Poids et Mesures* (BIPM) in Sevres, France. The GPS system time is steered to a Master Clock (MC) at the US Naval Observatory which is kept within a close but unspecified tolerance of TAI.

If not specified, the value of this parameter must be considered to be `utc`.

A `ttp:clockMode` attribute is considered to be significant only when specified on the `tt` element.

6.2.3 ttp:dropMode

The `ttp:dropMode` attribute is used to specify constraints on the interpretation and use of frame counts that correspond with [SMPTE 12M] time coordinates when operating with time base of `smpte` as defined by **6.2.11 ttp:timeBase**.

If specified, the value of this attribute must adhere to the following syntax:

Syntax Representation – ttp:dropMode

```
ttp:dropMode
  : "dropNTSC"
  | "dropPAL"
  | "nonDrop"
```

If the time base, defined by 6.2.11 ttp:timeBase, is designated as `smpte`, then this parameter applies as follows: if the parameter's value is `nonDrop`, then, within any given second of a time expression, frames count from 0 to $N-1$, where N is the value specified by the `ttp:frameRate` parameter, but while ignoring any value specified by the `ttp:frameRateMultiplier` parameter.

> **Note:**
>
> When operating in `nonDrop` mode, a second of a time expression may or may not be equal to a second of real time during normal (1x speed) forward playback. If the `ttp:frameRateMultiplier` parameter is specified and is not equal to 1:1, then a second of a time expression will either be shorter or longer than a second of elapsed play in real time.

> **Note:**
>
> See 10.3 Time Value Expressions regarding the absence of defined semantics for both the *offset-time* form and the *fraction* component of time expressions when the `smpte` time base applies.

If this parameter's value is `dropNTSC`, then, within any given second of a time expression except the second `00`, frames count from 0 to $N-1$, where N is the value specified by the `ttp:frameRate` parameter, but while ignoring any value specified by the `ttp:frameRateMultiplier` parameter. If the second of a time expression is `00` and the minute of the time expression is not `00`, `10`, `20`, `30`, `40`, or `50`, then frame codes `00` and `01` are dropped during that second; otherwise, these frame codes are not dropped.

> **Note:**
>
> For example, when operating in `dropNTSC` mode with `ttp:frameRate` of `30`, a discontinuity in frame count occurs between consecutive frames as shown in the following sequence of time expressions: `01:08:59:28`, `01:08:59:29`, `01:09:00:02`, `01:09:00:03`.

If this parameter's value is `dropPAL`, then, within any given second of a time expression except the second `00`, frames count from 0 to $N-1$, where N is the value specified by the `ttp:frameRate` parameter, but while ignoring any value specified by the `ttp:frameRateMultiplier` parameter. If the second of a time expression is `00` and the minute of the time expression is even but not `00`, `20`, or `40`, then frame codes `00` through `03` are dropped during that second; otherwise, these frame codes are not dropped.

> **Note:**
>
> For example, when operating in `dropPAL` mode with `ttp:frameRate` of `30`, a discontinuity in frame count occurs between consecutive frames as shown in the following sequence of time expressions: `01:09:59:28`, `01:09:59:29`, `01:10:00:04`, `01:10:00:05`.

> **Note:**
>
> The `dropPAL` mode is also known as the *M/PAL* or *PAL (M)* drop-frame code, which uses PAL modulation with the NTSC frame rate of ~29.97 frames/second. The M/PAL system is used primarily in Brazil.

If not specified, then `nonDrop` must be assumed to apply.

A `ttp:dropMode` attribute is considered to be significant only when specified on the `tt` element.

6.2.4 ttp:frameRate

The `ttp:frameRate` attribute is used to specify the frame rate of a related media object or the intrinsic frame rate of a *Document Instance* in case it is intended to function as an independent media object.

If specified, the value of this attribute must adhere to the following syntax:

Syntax Representation – ttp:frameRate

```
ttp:frameRate
  : <digit>+                              // value > 0
```

The frame rate that applies to a *Document Instance* is used to interpret time expressions that are expressed in *frames* as defined by **10.3.1 <timeExpression>**.

If the `media` time base applies and the effective frame rate is integral, then a frame is interpreted as a division of a second of media time, such that if the frame rate is specified as *F*, then a second of media time is divided into *F* intervals of equal duration, where each interval is labeled as frame *f*, with $f \in [0 \ldots F-1]$.

> **Note:**
>
> See **N.2 Media Time Base** for further details on the interpretation of time expressions for the `media` time base.

If not specified, the frame rate must be considered to be equal to some application defined frame rate, or if no application defined frame rate applies, then thirty (30) frames per second. If specified, then the frame rate must be greater than zero (0).

A `ttp:frameRate` attribute is considered to be significant only when specified on the `tt` element.

6.2.5 ttp:frameRateMultiplier

The `ttp:frameRateMultiplier` attribute is used to specify a multiplier to be applied to the frame rate specified by a `ttp:frameRate` attribute in order to compute the effective frame rate.

If specified, the value of this attribute must adhere to the following syntax:

Syntax Representation – ttp:frameRateMultiplier

```
ttp:frameRateMultiplier
  : numerator denominator                 // numerator != 0; denominator != 0

numerator | denominator
  : <digit>+
```

A frame rate multiplier is used when the desired frame rate cannot be expressed as an integral number of frames per second.

If not specified, the frame rate multiplier must be considered to be equal to some application defined frame rate multiplier, or if no application defined frame rate multiplier applies, then one (1:1). Both numerator and denominator must be non-zero.

A `ttp:frameRateMultiplier` attribute is considered to be significant only when specified on the `tt` element.

> **Note:**
>
> The frame rate multiplier used for synchronizing with NTSC [SMPTE 170M] formatted video objects at 30 frames per second is nominally 1000:1001. The nominal frame rate of NTSC video is defined as the chrominance sub-carrier frequency of 3,579,545.45…Hz (= 5.0MHz × 63/88) times the ratio 2/455 divided by the number of horizontal lines per frame of 525, which yields a frame rate of 29.970029970029… (= 30 × 1000/1001) frames per second. Other frame rate multipliers apply to different regions of usage and video format standards.

> **Note:**
>
> Except in the case of PAL/M, the frame rate multiplier used for synchronizing with PAL formatted video objects at 25 frames per second is nominally 1:1.

6.2.6 ttp:markerMode

The `ttp:markerMode` attribute is used to specify constraints on the interpretation and use of time expressions that correspond with [SMPTE 12M] time coordinates when operating with time base of `smpte` as defined by **6.2.11 ttp:timeBase**.

If specified, the value of this attribute must adhere to the following syntax:

Syntax Representation – ttp:markerMode

```
ttp:markerMode
  : "continuous"
  | "discontinuous"
```

If the time base, defined by **6.2.11 ttp:timeBase**, is designated as `smpte`, then this parameter applies as follows: if the parameter's value is `continuous`, then [SMPTE 12M] time coordinates may be assumed to be linear and either monotonically increasing or decreasing; however, if `discontinuous`, then any assumption must not be made regarding linearity or monotonicity of time coordinates.

If not specified, the value of this parameter must be considered to be `discontinuous`.

> **Note:**
>
> The default value for this parameter was originally specified (in TTML 1.0 First Edition) as `continuous`; however, further evaluation of the state of the industry indicates this choice was incorrect, and that the most common default is `discontinuous`.

> **Note:**
>
> Due to lack of industry consensus on the utility and interpretation of the `continuous` marker mode, authors are advised to avoid its use. Furthermore, the `ttp:markerMode` is being considered for deprecation in the next revision of this specification.

A `ttp:markerMode` attribute is considered to be significant only when specified on the `tt` element.

If a value of `continuous` applies, then time expressions may be converted to real time coordinates by taking into account the computed frame rate and drop mode as expressed by the `ttp:dropMode` parameter. In this case, the *Content Processor* must create and maintain a *Synthetic SMPTE Document Syncbase* within which these time expressions are interpreted as further described in **10.4 Time Intervals**.

> **Note:**
>
> When operating with `smpte` time base and `continuous` marker mode, there is an implied time coordinate space, the *Synthetic SMPTE Document Syncbase*, defined by the monotonically increasing (or decreasing) [SMPTE 12M] time coordinates, while taking into account the computed frame rate and drop mode. All time expressions are interpreted in relationship to this time coordinate space based upon *SMPTE Time Code* synchronization events (markers), where the *Document Processing Context* emits these events with implied constraints regarding time coordinate monoticity and resynchronization in the presence of dropped frames.
>
> Use of `continuous` marker mode with the `smpte` time base is different from using the `media` time base because (1) the semantics of the `ttp:dropMode` parameter apply to the former, but not the latter, and (2) [SMPTE 12M] time coordinates may be applied monotonically to media which has been subjected to dilation in time, constriction in time, or reversal in time.

If a value of `discontinuous` applies, then time expressions must not be converted to real time coordinates, arithmetical operators (addition, multiplication) are not defined on time expressions, and, consequently, any (well-formed) expression of a duration must be considered to be invalid.

> **Note:**
>
> When operating with `smpte` time base and `discontinuous` marker mode, there is no effective time coordinate space; rather, all time expressions are interpreted as labeled synchronization events (markers), where the *Document Processing Context* emits these events, which, when they correspond with time expressions that denote the same label, cause a temporal interval to begin or end accordingly.
>
> An additional side-effect of operating in `discontinuous` mode is that time expressions of children have no necessary relationship with time expressions of their temporal container; that is, temporal containers and children of these containers are temporally activated and inactivated independently based on the occurrence of a labeled synchronization (marker) event.

> **Note:**
>
> The notion of marker discontinuity as captured by this parameter is logically independent from the method used to count frames as expressed by the `ttp:dropMode` parameter. In particular, even if the `ttp:dropMode` parameter is specified as `dropNTSC` or `dropPAL`, the marker mode may be specified as `continuous`, even in the presence of frame count discontinuities induced by the frame counting method, unless there were some other non-linearity or discontinuity in marker labeling, for example, two consecutive frames labeled as `10:00:00:00` and `10:00:01:00`.

6.2.7 ttp:pixelAspectRatio

The `ttp:pixelAspectRatio` attribute may be used by a content author to express the aspect ratio of non-square pixels in the production of content that makes use of pixel coordinates.

If specified, the value of this attribute must adhere to the following syntax:

Syntax Representation – ttp:pixelAspectRatio

```
ttp:pixelAspectRatio
  : width height                          // width != 0; height != 0

width | height
```

```
  : <digit>+
```

If not specified, then square pixels (i.e., aspect ratio 1:1) must be assumed to apply. If specified, then both width and height must be non-zero.

A `ttp:pixelAspectRatio` attribute is considered to be significant only when specified on the `tt` element.

> **Note:**
>
> This parameter may be used by a content transcoder or translator in order to convert pixel measurements between different pixel aspect ratios while still maintaining authorial layout intentions.

6.2.8 ttp:profile

The `ttp:profile` attribute may be used by a content author to express the profile of the Timed Text Markup Language (TTML) used in a *Document Instance*.

If specified, the value of this attribute must adhere to the `xsd:anyURI` data type defined by [XML Schema Part 2], §3.2.17, and, further, must specify a profile designator in accordance with **5.2 Profiles**.

A `ttp:profile` attribute is considered to be significant only when specified on the `tt` element.

6.2.9 ttp:subFrameRate

The `ttp:subFrameRate` attribute is used to specify the sub-frame rate of a related media object or the intrinsic sub-frame rate of a *Document Instance* in case it is intended to function as an independent media object.

If specified, the value of this attribute must adhere to the following syntax:

Syntax Representation – ttp:subFrameRate

```
ttp:subFrameRate
  : <digit>+                                 // value > 0
```

The sub-frame rate that applies to a *Document Instance* is used to interpret time expressions that are expressed in *sub-frames* as defined by **10.3.1 <timeExpression>**.

If the `media` time base applies and the effective frame rate is integral, a sub-frame is interpreted as a division of a frame of media time, such that if the sub-frame rate is specified as *S*, then a frame of media time is divided into *S* intervals of equal duration, where each interval is labeled as sub-frame *s*, with $s \in [0 \ldots S-1]$.

> **Note:**
>
> See **N.2 Media Time Base** for further details on the interpretation of time expressions for the `media` time base.

If not specified, the sub-frame rate must be considered to be equal to some application defined sub-frame rate, or if no application defined sub-frame rate applies, then one (1). If specified, then the sub-frame rate must be greater than zero (0).

A `ttp:subFrameRate` attribute is considered to be significant only when specified on the `tt` element.

> **Note:**
>
> The sub-frame is sometimes referred to as a *field* in the context of synchronization with an interlaced video media object.

6.2.10 ttp:tickRate

The `ttp:tickRate` attribute is used to specify the tick rate of a related media object or the intrinsic tick rate of content of a *Document Instance* in case it is intended to function as an independent media object.

If specified, the value of this attribute must adhere to the following syntax:

Syntax Representation – ttp:tickRate

```
ttp:tickRate
  : <digit>+                              // value > 0
```

The tick rate that applies to a *Document Instance* is used to interpret time expressions that are expressed in *ticks* by using the `t` metric as defined by **10.3.1 <timeExpression>**.

If the `media` time base applies, a tick is interpreted as an integral division of a second of media time, such that if the tick rate is specified as T, then a second of media time is divided into T intervals of equal duration, where each interval is labeled as tick t, with $t \in [0 \ldots T-1]$.

> **Note:**
>
> See **N.2 Media Time Base** for further details on the interpretation of time expressions for the `media` time base.

If not specified, then if a frame rate is specified, the tick rate must be considered to be the effective frame rate multiplied by the sub-frame rate (i.e., ticks are interpreted as sub-frames); or, if no frame rate is specified, the tick rate must be considered to be one (1) tick per second of media time. If specified, then the tick rate must not be zero (0).

> **Note:**
>
> There is no predefined relationship between ticks and frames or sub-frames. Ticks are an arbitrary division of seconds that permit use of fixed point arithmetic rather than fractional (and potentially inexact) expressions of seconds.

A `ttp:tickRate` attribute is considered to be significant only when specified on the `tt` element.

6.2.11 ttp:timeBase

The `ttp:timeBase` attribute is used to specify the temporal coordinate system by means of which time expressions are interpreted in a *Document Instance*.

If specified, the value of this attribute must adhere to the following syntax:

Syntax Representation – ttp:timeBase

```
ttp:timeBase
  : "media"
  | "smpte"
  | "clock"
```

If the time base is designated as `media`, then a time expression denotes a coordinate in some media object's time line, where the media object may be an external media object with which the content of a *Document Instance* is to be synchronized, or it may be the content of a *Document Instance* itself in a case where the timed text content is intended to establish an independent time line.

> **Note:**
>
> When using a media time base, if that time base is paused or scaled positively or negatively, i.e., the media play rate is not unity, then it is expected that the presentation of associated Timed Text content will be similarly paused, accelerated, or decelerated, respectively. The means for controlling an external media time base is outside the scope of this specification.

If the time base is designated as `smpte`, then a time expression denotes a [SMPTE 12M] time coordinate with which the content of a *Document Instance* is to be synchronized. In this case, the value of the `ttp:markerMode` and `ttp:dropMode` parameters apply, as defined by **6.2.6 ttp:markerMode** and **6.2.3 ttp:dropMode**, respectively.

> **Note:**
>
> When the time base is designated as `smpte`, every time expression denotes a media marker value akin to that defined by [SMIL 2.1], § 10.4.1, except instead of using an opaque marker name, a structured [SMPTE 12M] time coordinate serves as the marker name.

If the time base is designated as `clock`, then the time expression denotes a coordinate in some real-world time line as established by some real-time clock, such as the local wall-clock time or UTC (Coordinated Universal Time) or GPS (Global Positioning System) time lines.

If not specified, the default time base must be considered to be `media`.

A `ttp:timeBase` attribute is considered to be significant only when specified on the `tt` element.

When operating with either `media` or `smpte` time bases, a diachronic presentation of a *Document Instance* may be subject to transformations of the controlling time line, such as temporal reversal, dilation (expansion), or constriction (compression); however, when operating with the `clock` time base, no transformations are permitted, and diachronic presentation proceeds on a linear, monotonically increasing time line based on the passage of real time.

> **Note:**
>
> Due to there being only one time base parameter that applies to a given *Document Instance*, the interpretation of time expressions is uniform throughout the *Document Instance*.

> **Note:**
>
> See **N Time Expression Semantics** for further details on the interpretation of time expressions according to the designated time base.

7 Content

This section specifies the *content* matter of the core vocabulary catalog.

7.1 Content Element Vocabulary

The following elements specify the structure and principal content aspects of a *Document Instance*:

- **7.1.1 tt**
- **7.1.2 head**
- **7.1.3 body**
- **7.1.4 div**
- **7.1.5 p**
- **7.1.6 span**
- **7.1.7 br**

7.1.1 tt

The `tt` element serves as the root document element of a *Document Instance*.

The `tt` element accepts as its children zero or one `head` element followed by zero or one `body` element.

XML Representation – Element Information Item: tt

```
<tt
  tts:extent = string
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve) : default
  {any attribute in TT Parameter namespace}
  {any attribute not in default or any TT namespace}>
  Content: head?, body?
</tt>
```

The *Root Temporal Extent*, i.e., the time interval over which a *Document Instance* is active, has an implicit duration that is equal to the implicit duration of the `body` element of the document, if the `body` element is present, or zero, if the `body` element is absent.

If the `tts:extent` attribute is specified on the `tt` element, then it must adhere to **8.2.7 tts:extent**, in which case it specifies the spatial extent of the *Root Container Region* in which content regions are located and presented. If no `tts:extent` attribute is specified, then the spatial extent of the *Root Container Region* is considered to be determined by the *Document Processing Context*. The origin of the *Root Container Region* is determined by the *Document Processing Context*.

> **Note:**
>
> In the absence of other requirements, and if a *Related Media Object* exists, then it is recommended that the *Document Processing Context* determine that:
>
> - if no `tts:extent` is specified on the root `tt` element, the extent of the *Root Container Region* be established as equal to the extent of the *Related Media Object Region*; and
> - the origin of the *Root Container Region* be established so that this region is centered in the *Related Media Object Region*.

> **Note:**
>
> If an author desires to signal the (storage or image) aspect ratio of the *Root Container Region* without specifying its resolution, then this may be accomplished by using metadata specified in an external namespace, such as `m708:aspectRatio` as defined in [SMPTE 2052-11], §5.4.4. This would permit, for example, the interchange of information that reflects the the semantics of [CEA-708-E] , §4.5 “Caption Service Metadata”, “ASPECT RATIO”.

An `xml:lang` attribute must be specified on the `tt` element. If the attribute value is empty, it signifies that there is no default language that applies to the text contained within the *Document Instance*.

If no `xml:space` attribute is specified upon the `tt` element, then it must be considered as if the attribute had been specified with a value of `default`.

7.1.2 head

The `head` element is a container element used to group header matter, including metadata, profile, styling, and layout information.

The `head` element accepts as its children zero or more elements in the `Metadata.class` element group, followed by zero or more elements in the `Parameters.class` element group, followed by zero or one `styling` element, followed by zero or one `layout` element.

Any metadata specified by children in the `Metadata.class` element group applies semantically to the *Document Instance* as a whole, and not just the `head` element.

Any parameters specified by children in the `Parameters.class` element group applies semantically to the *Document Instance* as a whole, and not just the `head` element.

A `styling` child element is used to specify style constructs that are referenced from other style constructs, by layout constructs, and by Content elements.

A `layout` child element is used to specify layout constructs that are referenced by Content elements.

XML Representation – Element Information Item: head

```
<head
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*, Parameters.class*, styling?, layout?
</head>
```

To the extent that time semantics apply to the content of the `head` element, the implied time interval of this element is defined to be coterminous with the *Root Temporal Extent*.

7.1.3 body

The `body` element functions as a logical container and a temporal structuring element for a sequence of textual content units represented as logical divisions.

The `body` element accepts as its children zero or more elements in the `Metadata.class` element group, followed by zero or more elements in the `Animation.class` element group, followed by zero or more `div` elements.

Any metadata specified by children in the `Metadata.class` element group applies semantically to the `body` element and its descendants as a whole.

Any animation elements specified by children in the `Animation.class` element group apply semantically to the `body` element.

XML Representation – Element Information Item: body

```
<body
  begin = <timeExpression>
  dur = <timeExpression>
  end = <timeExpression>
  region = IDREF
  style = IDREFS
  timeContainer = (par|seq)
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute in TT Metadata namespace}
  {any attribute in TT Style namespace}
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*, Animation.class*, div*
</body>
```

An author may specify a temporal interval for a `body` element using the `begin`, `dur`, and `end` attributes. If the begin point of this interval remains unspecified, then the begin point is interpreted as the beginning point of the *Root Temporal Extent*. Similarly, if the end point of this interval remains unspecified, then the end point is interpreted as the ending point of the *Root Temporal Extent*.

> **Note:**
>
> A *Document Instance* referenced from a SMIL presentation is expected to follow the same timing rules as apply to other SMIL media objects.

If relative begin or end times are specified on the `body` element, then these times are resolved by reference to the beginning and ending time of the *Root Temporal Extent*.

If the *Root Temporal Extent* is shorter than the computed duration of the `body` element, then the active time interval of a `body` element is truncated to the active end point of the *Root Temporal Extent*.

An author may associate a set of style properties with a `body` element by means of either the `style` attribute or inline style attributes or a combination thereof.

> **Note:**
>
> Style properties that are associated with a body element in a *Document Instance* are available for style inheritance by descendant Content elements such as div, p, span and br.

If no timeContainer attribute is specified on a body element, then it must be interpreted as having *parallel* time containment semantics.

7.1.4 div

The div element functions as a logical container and a temporal structuring element for a sequence of textual content units represented as logical sub-divisions or paragraphs.

> **Note:**
>
> When rendered on a continuous (non-paged) visual presentation medium, a div element is expected to generate one or more block areas that contain zero or more child block areas generated by the div element's descendant p elements.
>
> If some block area generated by a div element does not contain any child areas, then it is not expected to be presented.

The div element accepts as its children zero or more elements in the Metadata.class element group, followed by zero or more elements in the Animation.class element group, followed by zero or more div or p elements.

Any metadata specified by children in the Metadata.class element group applies semantically to the div element and its descendants as a whole.

Any animation elements specified by children in the Animation.class element group apply semantically to the div element.

XML Representation – Element Information Item: div

```
<div
  begin = <timeExpression>
  dur = <timeExpression>
  end = <timeExpression>
  region = IDREF
  style = IDREFS
  timeContainer = (par|seq)
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute in TT Metadata namespace}
  {any attribute in TT Style namespace}
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*, Animation.class*, Block.class*
</div>
```

An author may associate a set of style properties with a div element by means of either the style attribute or inline style attributes or a combination thereof.

> **Note:**
>
> Style properties that are associated with a `div` element in a *Document Instance* are available for style inheritance by descendant Content elements such as `div`, `p`, `span`, and `br`.

If no `timeContainer` attribute is specified on a `div` element, then it must be interpreted as having *parallel* time containment semantics.

7.1.5 p

A `p` element represents a logical paragraph, serving as a transition between block level and inline level formatting semantics.

The `p` element accepts as its children zero or more elements in the `Metadata.class` element group, followed by zero or more elements in the `Animation.class` element group, followed by zero or more intermixed `span` elements, `br` elements, or text nodes.

Any metadata specified by children in the `Metadata.class` element group applies semantically to the `p` element and its descendants as a whole.

Any animation elements specified by children in the `Animation.class` element group apply semantically to the `p` element.

XML Representation – Element Information Item: p

```
<p
  begin = <timeExpression>
  dur = <timeExpression>
  end = <timeExpression>
  region = IDREF
  style = IDREFS
  timeContainer = (par|seq)
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute in TT Metadata namespace}
  {any attribute in TT Style namespace}
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*, Animation.class*, Inline.class*
</p>
```

An author may associate a set of style properties with a `p` element by means of either the `style` attribute or inline style attributes or a combination thereof.

> **Note:**
>
> Style properties that are associated with a `p` element in a *Document Instance* are available for style inheritance by descendant Content elements such as `span` and `br`.

If no `timeContainer` attribute is specified on a `p` element, then it must be interpreted as having *parallel* time containment semantics.

If a sequence of children of a `p` element consists solely of character information items, then that sequence must be considered to be an *anonymous span* for the purpose of applying style properties that apply to `span` elements.

> **Note:**
>
> The presentation semantics of TTML effectively implies that a p element constitutes a line break. In particular, it is associated with a block-stacking constraint both before the first generated line area and after the last generated line area. See **9.3.4 Synchronic Flow Processing** for further details.

7.1.6 span

The span element functions as a logical container and a temporal structuring element for a sequence of textual content units having inline level formatting semantics.

When presented on a visual medium, a span element is intended to generate a sequence of inline areas, each containing one or more glyph areas.

The span element accepts as its children zero or more elements in the Metadata.class element group, followed by zero or more elements in the Animation.class element group, followed by zero or more intermixed span elements, br elements, or text nodes.

Any metadata specified by children in the Metadata.class element group applies semantically to the span element and its descendants as a whole.

Any animation elements specified by children in the Animation.class element group apply semantically to the span element.

XML Representation – Element Information Item: span

```
<span
  begin = <timeExpression>
  dur = <timeExpression>
  end = <timeExpression>
  region = IDREF
  style = IDREFS
  timeContainer = (par|seq)
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute in TT Metadata namespace}
  {any attribute in TT Style namespace}
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*, Animation.class*, Inline.class*
</span>
```

An author may associate a set of style properties with a span element by means of either the style attribute or inline style attributes or a combination thereof.

> **Note:**
>
> Style properties that are associated with a span element in a *Document Instance* are available for style inheritance by descendant Content elements such as span and br.

If no timeContainer attribute is specified on a span element, then it must be interpreted as having *parallel* time containment semantics.

7.1.7 br

The br element denotes an explicit line break.

Any metadata specified by children in the Metadata.class element group applies semantically to the br element and its descendants as a whole.

Any animation elements specified by children in the Animation.class element group apply semantically to the br element.

> **Note:**
>
> While permitted, the use of Animation.class children is not recommended since no style property applies to the br element. Instead, animating a br element can be achieved by wrapping it in an animated span element.

XML Representation – Element Information Item: br

```
<br
  style = IDREFS
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute in TT Metadata namespace}
  {any attribute in TT Style namespace}
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*, Animation.class*
</br>
```

When presented on a visual medium, the presence of a br element must be interpreted as a forced line break.

> **Note:**
>
> The visual presentation of a br element is intended to produce the same effect as the control character CR (U+000D) followed by the control code LF (U+000A) when presented on a teletype device. Therefore, two br elements in sequence will produce a different effect than a single br element.

7.2 Content Attribute Vocabulary

This section defines the following common attributes used with many or all element types in the core vocabulary catalog:

- **7.2.1 xml:id**
- **7.2.2 xml:lang**
- **7.2.3 xml:space**

7.2.1 xml:id

The xml:id attribute is used as defined by [XML ID].

The xml:id attribute may be used with any element in the core vocabulary catalog.

7.2.2 xml:lang

The xml:lang attribute is used as defined by [XML 1.0], §2.12, *Language Identification*.

The `xml:lang` attribute must be specified on the `tt` element and may be specified by an instance of any other element type in the core vocabulary catalog except parameter vocabulary.

7.2.3 xml:space

The `xml:space` attribute is used as defined by [XML 1.0], §2.10, *White Space Handling*.

The `xml:space` attribute may be used with any element in the core vocabulary catalog except parameter vocabulary.

The semantics of the value `default` are fixed to mean that when performing presentation processing of a *Document Instance* as described by **9.3.4 Synchronic Flow Processing**, processing must occur as if the following properties were specified on the affected elements of an equivalent intermediate XSL-FO document:

- `suppress-at-line-break="auto"`
- `linefeed-treatment="treat-as-space"`
- `white-space-collapse="true"`
- `white-space-treatment="ignore-if-surrounding-linefeed"`

Similarly, the semantics of the value `preserve` are fixed to mean that when performing presentation processing, processing must occur as if the following properties were specified on the affected elements of an equivalent intermediate XSL-FO document:

- `suppress-at-line-break="retain"`
- `linefeed-treatment="preserve"`
- `white-space-collapse="false"`
- `white-space-treatment="preserve"`

When performing other types of processing intended to eventually result in a visual presentation by means other than those described in this specification, the semantics of space collapsing and preservation as described above should be respected. For other types of processing, the treatment of the `xml:space` attribute is processor dependent, but should respect the semantics described above if possible.

> **Note:**
>
> The semantics of the above four cited XSL-FO properties are defined by by [XSL 1.1], § 7.17.3, 7.16.7, 7.16.12, and 7.16.8, respectively.

> **Note:**
>
> No presentation semantics are specified for the TAB (U+0009) character. Furthermore, the TAB (U+0009) character can generate a glyph area. As a result, the use of the TAB (U+0009) character in `#PCDATA` content within `p` and `span` elements is not recommended.

8 Styling

This section specifies the *styling* matter of the core vocabulary catalog, where styling is to be understood as a separable layer of information that applies to content and that denotes authorial intentions about the presentation of that content.

Styling attributes are included in TTML to enable authorial intent of presentation to be included within a self-contained document. This section describes the semantics of style presentation in terms of a standard processing model. TTML

Processors are not required to present *Document Instances* in any particular way; but an implementation of this model by a TTML Presentation Processor that provides externally observable results that are consistent with this model is likely to lead to a user experience that closely resembles the experience intended by the documents' authors.

> **Note:**
>
> The semantics of TTML style presentation are described in terms of the layout and formatting model defined in [XSL 1.1]. The effects of the attributes in this section are intended to be compatible with this model; however, presentation agents may use any technology to satisfy the authorial intent of the document. For example, a [CSS2] processor may be used to implement features it has in common with this model.

No normative use of an `<?xml-stylesheet ... ?>` processing instruction is defined by this specification.

8.1 Styling Element Vocabulary

The following elements specify the structure and principal styling aspects of a *Document Instance*:

- **8.1.1 styling**
- **8.1.2 style**

8.1.1 styling

The `styling` element is a container element used to group styling matter, including metadata that applies to styling matter.

The `styling` element accepts as its children zero or more elements in the `Metadata.class` element group, followed by zero or more `style` elements.

XML Representation – Element Information Item: styling

```
<styling
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*, style*
</styling>
```

To the extent that time semantics apply to the content of the `styling` element, the implied time interval of this element is defined to be coterminous with the *Root Temporal Extent*.

8.1.2 style

The `style` element is used to define a set of style specifications expressed as a specified style set in accordance with **8.4.4.2 Specified Style Set Processing**.

XML Representation – Element Information Item: style

```
<style
  style = IDREFS
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute in TT Style namespace}
```

```
    {any attribute not in default or any TT namespace}>
   Content: EMPTY
  </style>
```

If a `style` element appears as a descendant of a `region` element, then the `style` element must be ignored for the purpose of computing referential styles as defined by **8.4.1.2 Referential Styling** and **8.4.1.3 Chained Referential Styling**.

> **Note:**
>
> That is to say, when referential styling is used by an element to refer to a `style` element, then the referenced `style` element must appear as a descendant of the `styling` element, and not in any other context.

8.2 Styling Attribute Vocabulary

This section defines the **8.2.1 style** attribute used with both style definition elements as well as Content elements.

In addition, this section specifies the following attributes in the TT Style Namespace for use with style definition elements, certain layout elements, and Content elements that support inline style specifications:

- **8.2.2 tts:backgroundColor**
- **8.2.3 tts:color**
- **8.2.4 tts:direction**
- **8.2.5 tts:display**
- **8.2.6 tts:displayAlign**
- **8.2.7 tts:extent**
- **8.2.8 tts:fontFamily**
- **8.2.9 tts:fontSize**
- **8.2.10 tts:fontStyle**
- **8.2.11 tts:fontWeight**
- **8.2.12 tts:lineHeight**
- **8.2.13 tts:opacity**
- **8.2.14 tts:origin**
- **8.2.15 tts:overflow**
- **8.2.16 tts:padding**
- **8.2.17 tts:showBackground**
- **8.2.18 tts:textAlign**
- **8.2.19 tts:textDecoration**
- **8.2.20 tts:textOutline**
- **8.2.21 tts:unicodeBidi**
- **8.2.22 tts:visibility**
- **8.2.23 tts:wrapOption**
- **8.2.24 tts:writingMode**
- **8.2.25 tts:zIndex**

Unless explicitly stated otherwise, linear white-space (LWSP) must appear between adjacent non-terminal components of a value of a TT Style or TT Style Extension Property value unless some other delimiter is permitted and used.

> **Note:**
>
> This specification makes use of *lowerCamelCased* local names for style attributes that are based upon like-named properties defined by [XSL 1.1]. This convention is likewise extended to token values of such properties.

> **Note:**
>
> A style property may be expressed as a specified attribute on any Content element type independently of whether the property applies to that element type. This capability permits the expression of an inheritable style property on ancestor elements to which the property does not apply.

> **Note:**
>
> Due to the general syntax of this specification (and the schemas it references) with respect to how style attributes are specified, particularly for the purpose of supporting inheritance, it is possible for an author to inadvertently specify a non-inheritable style attribute on an element that applies neither to that element or any of its descendants while still remaining conformant from a content validity perspective. Content authors may wish to make use of TTML content verification tools that detect and warn about such usage.

8.2.1 style

The `style` attribute is used by referential style association to reference one or more `style` elements each of which define a style (property) set.

The `style` attribute may be specified by an instance of the following element types:

- `body`
- `br`
- `div`
- `p`
- `region`
- `span`
- `style`

If specified, the value of a `style` attribute must adhere to the `IDREFS` data type defined by [XML Schema Part 2], § 3.3.10, and, furthermore, each IDREF must reference a `style` element which has a `styling` element as an ancestor.

If the same IDREF, ID_1, appears more than one time in the value of a `style` attribute, then there should be an intervening IDREF, ID_2, where ID_2 is not equal to ID_1.

> **Note:**
>
> This constraint is intended to discourage the use of redundant referential styling while still allowing the same style to be referenced multiple times in order to potentially override prior referenced styles, e.g., when an intervening, distinct style is referenced in the IDREFS list.

> **Note:**
>
> See the specific element type definitions that permit use of the `style` attribute, as well as **8.4.1.2 Referential Styling** and **8.4.1.3 Chained Referential Styling**, for further information on its semantics.

8.2.2 tts:backgroundColor

The `tts:backgroundColor` attribute is used to specify a style property that defines the background color of a region or an area generated by content flowed into a region.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	<color>
Initial:	`transparent`
Applies to:	`body`, `div`, `p`, `region`, `span`
Inherited:	no
Percentages:	N/A
Animatable:	discrete

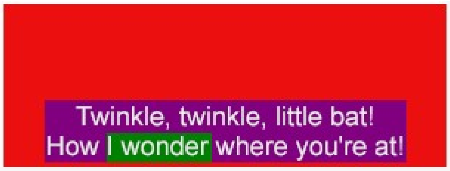
For the purpose of determining applicability of this style property, each character child of a `p` element is considered to be enclosed in an anonymous span.

The `tts:backgroundColor` style is illustrated by the following example.

Example Fragment – Background Color

```
<region xml:id="r1">
  <style tts:extent="306px 114px"/>
  <style tts:backgroundColor="red"/>
  <style tts:color="white"/>
  <style tts:displayAlign="after"/>
  <style tts:padding="3px 40px"/>
</region>
...
<p region="r1" tts:backgroundColor="purple" tts:textAlign="center">
  Twinkle, twinkle, little bat!<br/>
  How <span tts:backgroundColor="green">I wonder</span> where you're at!
</p>
```

Example Rendition – Background Color



Twinkle, twinkle, little bat!
How I wonder where you're at!

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.8.2.

8.2.3 tts:color

The `tts:color` attribute is used to specify a style property that defines the foreground color of marks associated with an area generated by content flowed into a region.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	<color>
Initial:	see prose
Applies to:	`span`
Inherited:	yes
Percentages:	N/A
Animatable:	discrete

For the purpose of determining applicability of this style property, each character child of a `p` element is considered to be enclosed in an anonymous span.

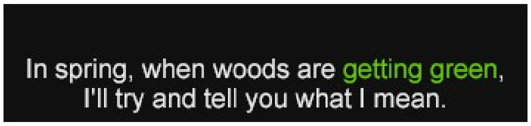
The initial value of the `tts:color` property is considered to be implementation dependent. In the absence of end-user preference information, a conformant presentation processor should use an initial value that is highly contrastive to the background color of the *Root Container Region*.

The `tts:color` style is illustrated by the following example.

Example Fragment – Color

```
<region xml:id="r1">
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="after"/>
  <style tts:textAlign="center"/>
</region>
...
<p region="r1">
  In spring, when woods are <span tts:color="green">getting green</span>,<br/>
  I'll try and tell you what I mean.
</p>
```

Example Rendition – Color



In spring, when woods are getting green,
I'll try and tell you what I mean.

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.18.1.

8.2.4 tts:direction

The `tts:direction` attribute is used to specify a style property that, depending on the context of use, determines (1) the bidirectional paragraph level, or (2) the directionality of a bidirectional embedding or override.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`ltr` | `rtl`

Initial:	`ltr`
Applies to:	`p`, `span`
Inherited:	yes
Percentages:	N/A
Animatable:	discrete

For the purpose of determining applicability of this style property, each character child of a `p` element is considered to be enclosed in an anonymous span.

When applied to a `p` element, the computed value of this property explicitly establishes the *paragraph level* as specified by [UAX9], § 4.3, Higher Level Protocol **HL1**.

When applied to a `span` element (or anonymous span), the computed value of this property, in combination with the computed value of the `tts:unicodeBidi` style property, determines the directionality of a bidirectional embedding or override as specified by [UAX9], § 4.3, Higher Level Protocol **HL3**.

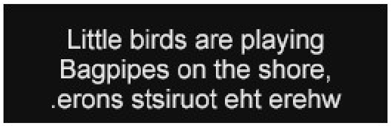
If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `ltr`.

The `tts:direction` style is illustrated by the following example.

Example Fragment – Direction

```
<region xml:id="r1">
  <style tts:extent="265px 84px"/>
  <style tts:padding="5px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="after"/>
  <style tts:textAlign="center"/>
</region>
...
<p region="r1">
  Little birds are playing<br/>
  Bagpipes on the shore,<br/>
  <span tts:unicodeBidi="bidiOverride" tts:direction="rtl">where the tourists snore.</span>
</p>
```

Example Rendition – Direction



Little birds are playing
Bagpipes on the shore,
.erons stsiruot eht erehw

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.29.1.

8.2.5 tts:display

The `tts:display` attribute is used to specify a style property that defines whether an element is a candidate for layout and composition in a region.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	auto | none
Initial:	auto
Applies to:	body, div, p, region, span
Inherited:	no
Percentages:	N/A
Animatable:	discrete

For the purpose of determining applicability of this style property, each character child of a p element is considered to be enclosed in an anonymous span.

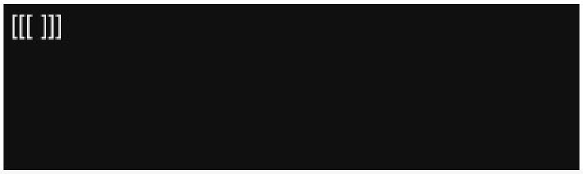
If the value of this attribute is auto, then the affected element is a candidate for region layout and presentation; however, if the value is none, then the affected element and its descendants must be considered ineligible for region layout and presentation.

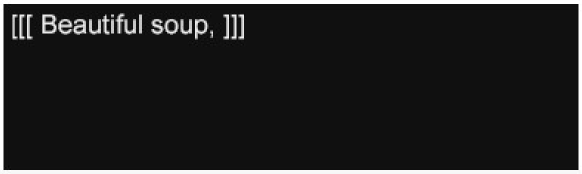
The tts:display style is illustrated by the following example.

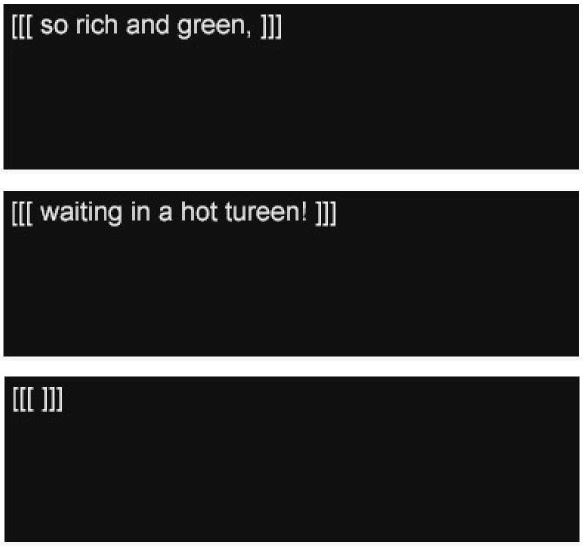
Example Fragment – Display

```
<region xml:id="r1">
  <style tts:extent="369px 119px"
            tts:backgroundColor="black"
            tts:color="white"
            tts:displayAlign="before"
            tts:textAlign="start"/>
</region>
...
<div region="r1">
  <p dur="5s">
    [[[
    <span tts:display="none">
      <set begin="1s" dur="1s" tts:display="auto"/>
      Beautiful soup,
    </span>
    <span tts:display="none">
      <set begin="2s" dur="1s" tts:display="auto"/>
      so rich and green,
    </span>
    <span tts:display="none">
      <set begin="3s" dur="1s" tts:display="auto"/>
      waiting in a hot tureen!
    </span>
    ]]]
  </p>
</div>
```

Example Rendition – Display


[[[ ]]]


[[[Beautiful soup,]]]



[[[so rich and green,]]]

[[[waiting in a hot tureen!]]]

[[[ ]]]

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [CSS2], § 9.2.4.

8.2.6 tts:displayAlign

The `tts:displayAlign` attribute is used to specify a style property that defines the alignment of block areas in the block progression direction.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`before` | `center` | `after`
Initial:	`before`
Applies to:	`region`
Inherited:	no
Percentages:	N/A
Animatable:	discrete

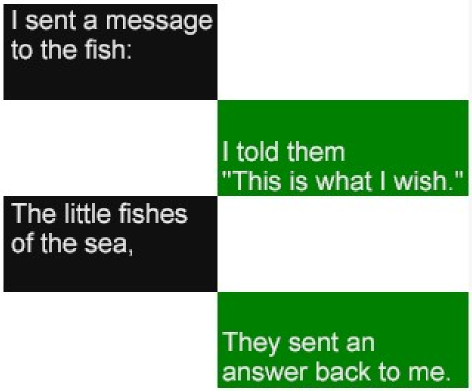
If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `before`.

The `tts:displayAlign` style is illustrated by the following example.

Example Fragment – Display Align

```
<region xml:id="r1">
  <style tts:extent="128px 66px" tts:origin="0px 0px"
       tts:backgroundColor="black" tts:color="white"/>
  <style tts:displayAlign="before"/>
  <style tts:textAlign="start"/>
</region>
<region xml:id="r2">
  <style tts:extent="192px 66px" tts:origin="128px 66px"/>
       tts:backgroundColor="green" tts:color="white"/>
  <style tts:displayAlign="after"/>
  <style tts:textAlign="start"/>
</region>
<region xml:id="r3">
  <style tts:extent="128px 66px"/> style tts:origin="0px 132px"
       tts:backgroundColor="black" tts:color="white"/>
  <style tts:displayAlign="before"/>
  <style tts:textAlign="start"/>
</region>
<region xml:id="r4">
  <style tts:extent="192px 66px" tts:origin="128px 198px"/>
       tts:backgroundColor="green" tts:color="white"/>
  <style tts:displayAlign="after"/>
  <style tts:textAlign="start"/>
</region>
...
<div>
  <p region="r1">I sent a message to the fish:</p>
  <p region="r2">I told them<br/> "This is what I wish."</p>
  <p region="r3">The little fishes of the sea,</p>
  <p region="r4">They sent an<br/> answer back to me.</p>
</div>
```

Example Rendition – Display Align



> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.14.4.

8.2.7 tts:extent

The `tts:extent` attribute is used to specify the *width* and *height* of a region area (which may be the *Root Container*

Region).

> **Note:**
>
> If padding is applied to a region, then the region's extent includes that padding, i.e., padding is applied as an inset, and, therefore, is interior to the region's extent.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`auto` | <length> <length>
Initial:	`auto`
Applies to:	`tt`, `region`
Inherited:	no
Percentages:	relative to width and height of *Root Container Region*
Animatable:	discrete

If the value of this attribute consists of two <length> specifications, then they must be interpreted as *width* and *height*, where the first specification is the *width*, and the second specification is the *height*.

The <length> value(s) used to express extent must be non-negative.

If the value of this attribute is `auto`, then the computed value of the style property must be considered to be the same as the extent of the *Root Container Region*.

The extent of the *Root Container Region* is determined either by a `tts:extent` specified on the `tt` element, if present, or as described by **7.1.1 tt** if not present. If `tts:extent` is specified on the `tt` element, then the width and height must be expressed in terms of two <length> specifications, and these specifications must be expressed as non-percentage, definite lengths using pixel units.

If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the closest supported value.

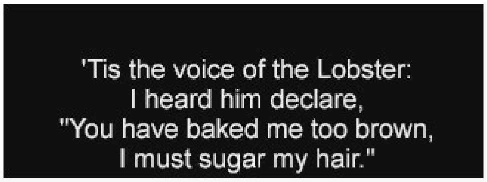
> **Note:**
>
> In this context, the phrase *closest supported value* means the value for which the Euclidean distance between the computed extent and the supported extent is minimized. If there are multiple closest supported values equally distant from the computed value, then the value most distant from [0,0], i.e., of greatest extent, is used.
>
> This rule for resolving *closest supported value* for the `tts:extent` attribute makes use of the nearest larger rather than nearest smaller supported distance. The rationale for this difference in treatment is that use of a larger extent ensures that the affected content will be contained in the region area without causing region overflow, while use of a smaller extent makes region overflow more likely.

The `tts:extent` style is illustrated by the following example.

Example Fragment – Extent

```
<region xml:id="r1">
  <style tts:extent="330px 122px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="after"/>
  <style tts:textAlign="center"/>
</region>
...
<p region="r1">
  'Tis the voice of the Lobster:<br/>
  I heard him declare,<br/>
  "You have baked me too brown,<br/>
  I must sugar my hair."
</p>
```

Example Rendition – Extent



'Tis the voice of the Lobster:
I heard him declare,
"You have baked me too brown,
I must sugar my hair."

8.2.8 tts:fontFamily

The `tts:fontFamily` attribute is used to specify a style property that defines the font family from which glyphs are selected for glyph areas generated by content flowed into a region.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	(<familyName> | <genericFamilyName>) ("," (<familyName> | <genericFamilyName>))*
Initial:	`default`
Applies to:	`span`
Inherited:	yes
Percentages:	N/A
Animatable:	discrete

> **Note:**
>
> While permitted, the use of linear whitespace (LWSP) around the comma delimiters in the value of this attribute is not recommended. The rationale for this recommendation is that some existing processors may not be tolerant of this usage.

> **Note:**
>
> The initial value, `default`, is a generic font family name, and is further described in **8.3.6 <genericFamilyName>** below.

For the purpose of determining applicability of this style property, each character child of a `p` element is considered to be enclosed in an anonymous span.

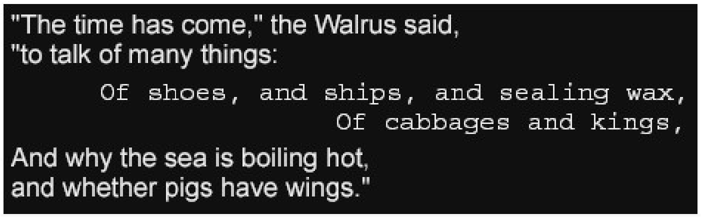
If a computed value of the property associated with this attribute is not supported, then a presentation processor must attempt to map the computed font family to a supported font family that has similar typographic characteristics, or, in the absence of such a mapping, it must use the value `default`.

The `tts:fontFamily` style is illustrated by the following example.

Example Fragment – Font Family

```
<region xml:id="r1">
  <style tts:extent="474px 146px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="center"/>
  <style tts:textAlign="start"/>
  <style tts:fontFamily="proportionalSansSerif"/>
</region>
...
<div region="r1">
  <p>
    "The time has come," the Walrus said,<br/>
    "to talk of many things:
  </p>
  <p tts:textAlign="end" tts:fontFamily="monospaceSerif">
    Of shoes, and ships, and sealing wax,<br/>
    Of cabbages and kings,
  </p>
  <p>
    And why the sea is boiling hot,<br/>
    and whether pigs have wings."
  </p>
</div>
```

Example Rendition – Font Family



"The time has come," the Walrus said,
"to talk of many things:

Of shoes, and ships, and sealing wax,
Of cabbages and kings,

And why the sea is boiling hot,
and whether pigs have wings."

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.9.2.

8.2.9 tts:fontSize

The `tts:fontSize` attribute is used to specify a style property that defines the font size for glyphs that are selected for glyph areas generated by content flowed into a region.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	<length> <length>?

Initial:	`1c`
Applies to:	`span`
Inherited:	yes
Percentages:	if not `region` element, then relative to the parent element's computed font size; otherwise, relative to the *Computed Cell Size*.
Animatable:	discrete

The computed value of the property associated with this attribute consists of a 2-tuple the entries of which denote the computed values of the width and height components of the font size respectively.

If a single <length> value is specified, then this length applies equally to horizontal and vertical size of a glyph's EM square; if two <length> values are specified, then the first expresses the horizontal size and the second expresses vertical size.

> **Note:**
>
> A glyph's EM square is conventionally defined as the EM square of the font that contains the glyph. That is, glyphs do not have an EM square that is distinct from their font's EM square.

> **Note:**
>
> Anamorphic font scaling, i.e., fonts scaled to independent (and distinct) horizontal and vertical sizes, is expected to be used in a number of contexts, such as when an author desires to synthesize a narrow or wide font face, when an author desires to employ font sizes based on non-square cell units, etc.

If horizontal and vertical sizes are expressed independently, then the units of the <length> values must be the same.

Relative <length> values that appear in this attribute, i.e., values expressed in percentage (`%`), cell (`c`), or EM (`em`) units, are resolved in relation to a referenced 2-dimensional size value consisting of two components, a width component and a height component.

When a single relative <length> value is specified, then this <length> is resolved in terms of the height component of the referenced value; when two relative <length> values are specified, the first <length> is resolved in terms of the width component of the referenced value and the second <length> is resolved in terms of the height component of the referenced value.

If the relative unit is cell (`c`), then the referenced value is the *Computed Cell Size*.

> **Note:**
>
> For example, consider a paragraph (`p`) element `P`. If the *Computed Cell Size* is (24px,36px), and if `tts:fontSize="1c"` is specified on `P`, then the computed value of `tts:fontSize` resolves to (36px,36px), while if `tts:fontSize="1c 1c"` is specified on `P`, then the computed value resolves to (24px,36px).

If the relative unit is EM (`em`), then the referenced value is determined as if percentage units were used, where `1em` equals `100%`.

> **Note:**
>
> For example, consider a span element `S`, a child of a paragraph (`p`) element `P`. If the computed value of `tts:fontSize` on `P` is (18px,24px), and if `tts:fontSize="1em"` is specified on `S`, then this is equivalent to specifying `100%`, which resolves to (24px,24px). However, if `tts:fontSize="1em 1em"` is specified on `S`, then this is equivalent to specifying `100% 100%` which resolves to (18px,24px).

The <length> value(s) used to express font size must be non-negative.

For the purpose of determining applicability of this style property, each character child of a `p` element is considered to be enclosed in an anonymous span.

If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the closest supported value.

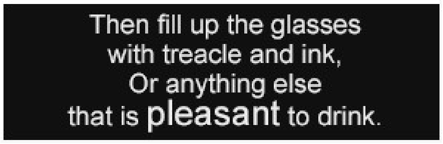
> **Note:**
>
> In this context, the phrase *closest supported value* means the value for which the Euclidean distance between the computed font size and the supported font size is minimized. If there are multiple closest supported values equally distant from the computed value, then the value most distant from 0 (single length specification) or [0,0] (two length specifications) is used, i.e., the largest font size, is used.

The `tts:fontSize` style is illustrated by the following example.

Example Fragment – Font Size

```
<region xml:id="r1">
  <style tts:extent="299px 97px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="center"/>
  <style tts:textAlign="center"/>
  <style tts:fontFamily="proportionalSansSerif"/>
  <style tts:fontSize="18px"/>
</region>
...
<p region="r1">
  Then fill up the glasses<br/>
  with treacle and ink,<br/>
  Or anything else<br/>
  that is <span tts:fontSize="24px">pleasant</span> to drink.
</p>
```

Example Rendition – Font Size



Then fill up the glasses
with treacle and ink,
Or anything else
that is pleasant to drink.

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.9.4. The addition of a second length component to permit specifying font width and height independently is an extension introduced by TTML.

8.2.10 tts:fontStyle

The `tts:fontStyle` attribute is used to specify a style property that defines the font style to apply to glyphs that are selected for glyph areas generated by content flowed into a region, where the mapping from font style value to specific font face or style parameterization is not determined by this specification.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`normal` | `italic` | `oblique`
Initial:	`normal`
Applies to:	`span`
Inherited:	yes
Percentages:	N/A
Animatable:	discrete

For the purpose of determining applicability of this style property, each character child of a `p` element is considered to be enclosed in an anonymous span.

Use of the value `oblique` denotes a shear transformation (at an unspecified angle) in the inline progression dimension.

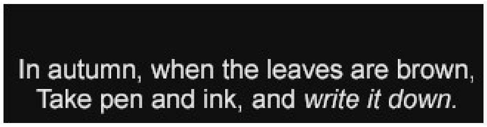
If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `normal`.

The `tts:fontStyle` style is illustrated by the following example.

Example Fragment – Font Style

```
<region xml:id="r1">
  <style tts:extent="331px 84px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="after"/>
  <style tts:textAlign="center"/>
  <style tts:fontFamily="proportionalSansSerif"/>
</region>
...
<p region="r1">
  In autumn, when the leaves are brown,<br/>
  Take pen and ink, and <span tts:fontStyle="italic">write it down.</span>
</p>
```

Example Rendition – Font Style



In autumn, when the leaves are brown,
Take pen and ink, and *write it down.*

Note:

The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.9.7.

8.2.11 tts:fontWeight

The `tts:fontWeight` attribute is used to specify a style property that defines the font weight to apply to glyphs that are selected for glyph areas generated by content flowed into a region, where the mapping from font weight value to specific font face or weight parameterization is not determined by this specification.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`normal` | `bold`
Initial:	`normal`
Applies to:	`span`
Inherited:	yes
Percentages:	N/A
Animatable:	discrete

For the purpose of determining applicability of this style property, each character child of a `p` element is considered to be enclosed in an anonymous span.

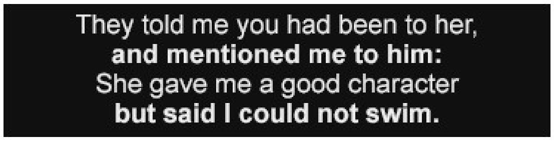
If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `normal`.

The `tts:fontWeight` style is illustrated by the following example.

Example Fragment – Font Weight

```
<region xml:id="r1">
  <style tts:extent="376px 95px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="center"/>
  <style tts:textAlign="center"/>
  <style tts:fontFamily="proportionalSansSerif"/>
</region>
...
<p region="r1">
  They told me you had been to her,<br/>
  <span tts:fontWeight="bold">and mentioned me to him:</span><br/>
  She gave me a good character<br/>
  <span tts:fontWeight="bold">but said I could not swim.</span>
</p>
```

Example Rendition – Font Weight



They told me you had been to her,
and mentioned me to him:
She gave me a good character
but said I could not swim.

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.9.9.

8.2.12 tts:lineHeight

The `tts:lineHeight` attribute is used to specify a style property that defines the inter-baseline separation between line areas generated by content flowed into a region.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`normal` | <length>
Initial:	`normal`
Applies to:	`p`

Inherited:	yes. See also special inheritance semantics.
Percentages:	relative to this element's font size
Animatable:	discrete

If the value of this attribute is `normal`, then the used value of this style property should be determined as follows:

1. Let *P* be the `p` element to which this style property applies.
2. Let *FF* be the computed value of the `tts:fontFamily` style property of *P*.
3. Let *FS* be the computed value of the `tts:fontSize` style property of *P*.
4. Let *F0* be the first font obtained when sequentially mapping each font family in *FF* to a set of available fonts, where this set of available fonts is constrained as needed to satisfy the computed values of the `tts:fontStyle` and `tts:fontWeight` style properties of *P*.
5. If *F0* is associated with font metrics that specify altitude *A*, descent *D*, and line gap *G*, then set *LH* to the sum of scaled(*A*), scaled(*D*), and scaled(*G*), where scaled(*X*) denotes font metric *X* scaled according to font size *FS*.
6. Otherwise, *LH* is considered to be implementation dependent; however, in the absence of implementation specific requirements, *LH* is recommended to be set to 125% of *FS*.
7. Set the used value of this style property to *LH*.

> **Note:**
>
> If a content author wishes to avoid the possibility of different interpretations of `normal`, for example, due to differences in the set of available fonts, then it is recommended that a <length> value expression be used to explicitly specify line height value.
>
> It is the intention of this specification that the above algorithm be compatible with [XSL 1.1] and [CSS2].

If specified as a <length>, then the length must be non-negative.

If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the closest supported value.

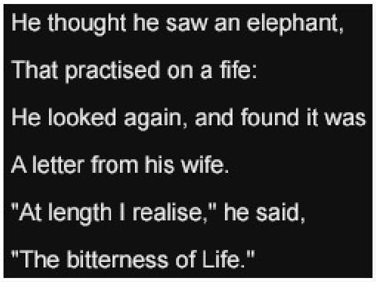
> **Note:**
>
> In this context, the phrase *closest supported value* means the value for which the Euclidean distance between the computed line height and the supported line height is minimized. If there are multiple closest supported values equally distant from the computed value, then the value most distant from 0, i.e., the largest line height, is used.

The `tts:lineHeight` style is illustrated by the following example.

Example Fragment – Line Height

```
<region xml:id="r1">
  <style tts:extent="255px 190px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="center"/>
  <style tts:textAlign="start"/>
  <style tts:fontFamily="proportionalSansSerif"/>
  <style tts:fontSize="16px"/>
  <style tts:lineHeight="32px"/>
</region>
...
<p region="r1">
  He thought he saw an elephant,<br/>
  That practised on a fife:<br/>
  He looked again, and found it was<br/>
  A letter from his wife.<br/>
  "At length I realise," he said,<br/>
  "The bitterness of Life.
"</p>
```

Example Rendition – Line Height



He thought he saw an elephant,

That practised on a fife:

He looked again, and found it was

A letter from his wife.

"At length I realise," he said,

"The bitterness of Life."

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.16.4. Furthermore, it is the intention of this specification that the allocation rectangle of a line be consistent with the **per-inline-height-rectangle** as defined by [XSL 1.1], § 4.5, i.e., that a CSS-style line box stacking strategy be used.

8.2.12.1 Special Inheritance Semantics

When the value of the attribute is `normal` and when applying inheritance semantics, the value `normal`, not the computed value, is inherited. However, when this computed value is to be used to perform paragraph layout, it is further processed to obtain a used value as described above.

8.2.13 tts:opacity

The `tts:opacity` attribute is used to specify a style property that defines the opacity (or conversely, the transparency) of marks associated with a region.

When presented onto a visual medium, the opacity of the region is applied uniformly and on a linear scale to all marks produced by content targeted to the region.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

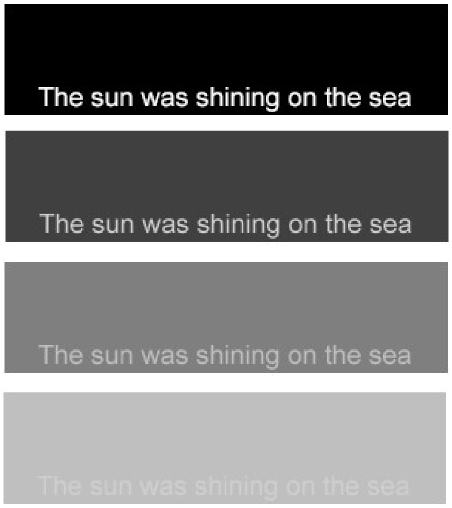
Values:	<alpha>
Initial:	`1.0`
Applies to:	`region`
Inherited:	no
Percentages:	N/A
Animatable:	discrete

The `tts:opacity` style is illustrated by the following example.

Example Fragment – Opacity

```
<region xml:id="r1" dur="5s">
  <set begin="0s" dur="1s" tts:opacity="1.00"/>
  <set begin="1s" dur="1s" tts:opacity="0.75"/>
  <set begin="2s" dur="1s" tts:opacity="0.50"/>
  <set begin="3s" dur="1s" tts:opacity="0.25"/>
  <set begin="4s" dur="1s" tts:opacity="0.00"/>
  <style tts:extent="304px 77px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="after"/>
  <style tts:textAlign="center"/>
</region>
...
<p region="r1">
  The sun was shining on the sea
</p>
```

Example Rendition – Opacity


The sun was shining on the sea
The sun was shining on the sea
The sun was shining on the sea
The sun was shining on the sea

Note:

The semantics of the style property represented by this attribute are based upon that defined by [CSS3 Color], § 3.2.

8.2.14 tts:origin

The `tts:origin` attribute is used to specify the *x* and *y* coordinates of the origin of a region area with respect to the origin of the *Root Container Region*.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`auto` | <length> <length>
Initial:	`auto`
Applies to:	`region`
Inherited:	no
Percentages:	relative to width and height of *Root Container Region*
Animatable:	discrete

If the value of this attribute consists of two <length> specifications, then they must be interpreted as *x* and *y* coordinates, where the first specification is the *x* coordinate, and the second specification is the *y* coordinate.

If the value of this attribute is `auto`, then the computed value of the style property must be considered to be the same as the origin of the *Root Container Region*.

If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the closest supported value.

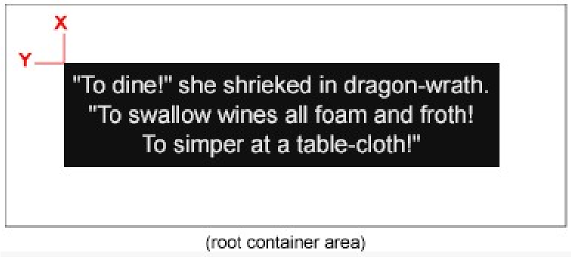
> **Note:**
>
> In this context, the phrase *closest supported value* means the value for which the Euclidean distance between the computed origin and the supported origin is minimized. If there are multiple closest supported values equally distant from the computed value, then the value least distant from [0,0], i.e., closest to the coordinate space origin, is used.

The `tts:origin` style is illustrated by the following example.

Example Fragment – Origin

```
<region xml:id="r1">
  <style tts:origin="40px 40px"/>
  <style tts:extent="308px 92px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="center"/>
  <style tts:textAlign="center"/>
</region>
...
<p region="r1">
  "To dine!" she shrieked in dragon-wrath.<br/>
  "To swallow wines all foam and froth!<br/>
   To simper at a table-cloth!"
</p>
```

Example Rendition – Origin


X
Y
"To dine!" she shrieked in dragon-wrath.
"To swallow wines all foam and froth!
To simper at a table-cloth!"
(root container area)

8.2.15 tts:overflow

The `tts:overflow` attribute is used to specify a style property that defines whether a region area is clipped or not if the descendant areas of the region overflow its extent.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`visible` | `hidden`
Initial:	`hidden`
Applies to:	`region`
Inherited:	no
Percentages:	N/A
Animatable:	discrete

If the value of this attribute is `visible`, then content should not be clipped outside of the affected region, and region composition and layout must be performed as if the region's width and height were unconstrained, but with a well-defined origin. If the value is `hidden`, then content should be clipped outside of the affected region.

> **Note:**
>
> This attribute has no impact on presentation processing when no overflow condition applies. An overflow condition applies when the bounding box of some descendant line area extends outside of the containing region's nominal content area extent in either or both 1) the inline and 2) the block progression dimensions, where the nominal content area extent in both dimensions is determined by the computed values of the `tts:extent` and `tts:padding` style properties of the containing region. Overflow in the inline progression dimension can occur only if `tts:wrapOption` is `noWrap`. Furthermore, when an overflow condition applies, it is not intended that the effective extent of the region be modified for the purpose of presentation processing. For example, the area painted with the region's background color is not extended in either dimension to enclose the overflowed content.
>
> Note that, in particular, the normative text in the previous paragraph "region composition and layout must be performed as if the region's width and height were unconstrained" refers to the process of determining the effective extent and origin of descendant line areas produced in either (or both) of the two overflow contexts described here, and is not intended to imply that the region extent is altered for the purpose of determining the region's padding area insets or the extent of its background color. More specifically, the normative language above is not intended to override or contradict the semantics of [XSL 1.1], § 7.21.2, or of [CSS2], § 11.1.1, on which the former is based.

> **Note:**
>
> Unless a manual line break element `br` is used by the content author, a paragraph of a given region will generate no more than one line area in that region if the value of the `tts:overflow` style that applies to the region is `visible` and if the applicable `tts:wrapOption` style is `noWrap`.

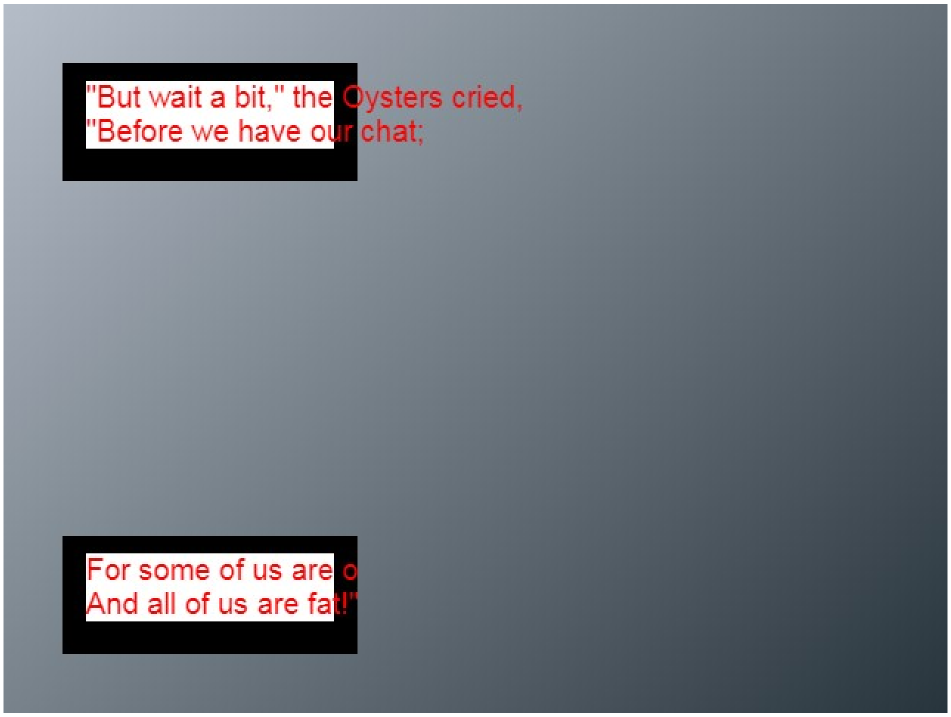
If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `hidden`.

The `tts:overflow` style is illustrated by the following example.

Example Fragment – Overflow

```
<?xml version="1.0"?>
<tt xmlns="http://www.w3.org/ns/ttml"
    xmlns:tts="http://www.w3.org/ns/ttml#styling"
    xml:lang="en"
    tts:extent="320px 240px">
  <head>
    <styling>
      <style xml:id="s1" tts:backgroundColor="black"
             tts:padding="6px"/>
      <style xml:id="s2" tts:color="red" tts:backgroundColor="white"
             tts:wrapOption="noWrap" tts:fontFamily="proportionalSansSerif"
             tts:fontSize="10px"/>
    </styling>
    <layout>
      <region xml:id="r1" style="s1" tts:extent="100px 40px"
              tts:origin="20px 20px" tts:overflow="visible"/>
      <region xml:id="r2" style="s1" tts:extent="100px 40px"
              tts:origin="20px 180px" tts:overflow="hidden"/>
    </layout>
  </head>
  <body>
    <div>
      <p region="r1" style="s2">
        "But wait a bit," the Oysters cried,<br/>
        "Before we have our chat;
      </p>
      <p region="r2" style="s2">
        For some of us are out of breath,<br/>
        And all of us are fat!"
      </p>
    </div>
  </body>
</tt>
```

Example Rendition – Overflow



"But wait a bit," the Oysters cried,
"Before we have our chat;

For some of us are o
And all of us are fat!"

> **Note:**
>
> In the above example, the `tts:noWrap` is set to `noWrap` to prevent automatic line wrapping (breaking); if this were not specified, then overflow would occur in the block progression direction as opposed to the inline progression direction.

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.21.2.

8.2.16 tts:padding

The `tts:padding` attribute is used to specify padding (or inset) space on all sides of a region area.

> **Note:**
>
> Padding is applied as an inset to a region area, which is to say, the content rectangle of a region area is reduced by the presence of padding applied to the region.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	<length> | <length> <length> | <length> <length> <length> | <length> <length> <length> <length>
Initial:	`0px`
Applies to:	`region`
Inherited:	no
Percentages:	relative to width and height of region

Animatable:	discrete

If the value of this attribute consists of one <length> specification, then that length applies to all edges of the affected areas. If the value consists of two <length> specifications, then the first applies to the before and after edges, and the second applies to the start and end edges. If three <length> specifications are provided, then the first applies to the before edge, the second applies to the start and end edges, and the third applies to the after edge. If four <length> specifications are provided, then they apply to before, end, after, and start edges, respectively.

> **Note:**
>
> Percentage values are relative to the dimension of the region to which they apply. For example, if the before and after edges correspond to the top and bottom edges of the region, as is the case for Latin text where `tts:writingMode` is equal to `"lrtb"`, then percentage values that apply to either of the two edges are relative to the height of the region. Conversely, if the before and after edges correspond to the right and left edges of the region, as is the case for Japanese text where `tts:writingMode` is equal to `"tbrl"`, then percentage values that apply to either of the two edges are relative to the width of the region.

The <length> value(s) used to express padding must be non-negative.

If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the closest supported value.

> **Note:**
>
> In this context, the phrase *closest supported value* means the value for which the one-dimensional Euclidean distance between the computed padding and the supported padding is minimized on a per-edge basis. If there are multiple closest supported values equally distant from the computed value for a given edge, then the value least distant from 0, i.e., the least padding, is used.

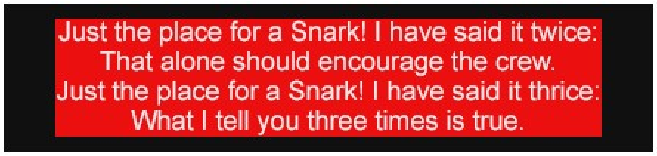
The `tts:padding` style is illustrated by the following example.

Example Fragment – Padding

```
<region xml:id="r1">
  <style tts:extent="446px 104px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="after"/>
  <style tts:textAlign="center"/>
  <style tts:padding="10px 40px"/>
</region>
...
<p region="r1" tts:backgroundColor="red">
  Just the place for a Snark! I have said it twice:<br/>
  That alone should encourage the crew.<br/>
  Just the place for a Snark! I have said it thrice:<br/>
  What I tell you three times is true.
</p>
```

When rendering an area to which padding applies, the background color that applies to the area is rendered into the padded portion of the area.

Example Rendition – Padding



Just the place for a Snark! I have said it twice:
That alone should encourage the crew.
Just the place for a Snark! I have said it thrice:
What I tell you three times is true.

Note:

The above example depicts how padding is applied as an inset to a region area. In particular, `10px` of padding is applied to the before (top) and after (bottom) edges, and `40px` of padding is applied at the start (left) and end (right) edges. Subtracting these from the extent of the region area results in the region's content rectangle having `366px` width and `84px` height. The black background color of the region appears in the region's padding rectangle while the red background color of the paragraph appears in the region's content rectangle.

Note:

The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.31.15, except that individual shorthand values map to writing mode relative padding values as defined by [XSL 1.1], § 7.8.31, 7.8.32, 7.8.33, and 7.8.34.

8.2.17 tts:showBackground

The `tts:showBackground` attribute is used to specify constraints on when the background color of a region is intended to be presented.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`always` | `whenActive`
Initial:	`always`
Applies to:	`region`
Inherited:	no
Percentages:	N/A
Animatable:	discrete

If the value of this attribute is `always`, then the background color of a region is always rendered when performing presentation processing on a visual medium; if the value is `whenActive`, then the background color of a region is rendered only when some content is flowed into the region.

A region satisfies the `whenActive` case if (1) it is a *Temporally Active Region* and (2) content is selected into the region, where that content is also *Temporally Active*.

If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `always`.

The `tts:showBackground` style is illustrated by the following example.

Example Fragment – Show Background

```
<region xml:id="r1">
  <style tts:origin="0px 0px"/>
  <style tts:extent="265px 100px"/>
  <style tts:backgroundColor="black"/>
  <style tts:showBackground="always"/>
  <style tts:color="white"/>
  <style tts:displayAlign="before"/>
  <style tts:textAlign="start"/>
</region>
<region xml:id="r2">
  <style tts:origin="205px 60px"/>
  <style tts:extent="290px 100px"/>
  <style tts:backgroundColor="red"/>
  <style tts:color="white"/>
  <style tts:displayAlign="before"/>
  <style tts:textAlign="end"/>
  <style tts:showBackground="whenActive"/>
</region>
```

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [SMIL 2.1], § 5.3.3.

8.2.18 tts:textAlign

The `tts:textAlign` attribute is used to specify a style property that defines how inline areas are aligned within a containing block area in the inline progression direction.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`left` | `center` | `right` | `start` | `end`
Initial:	`start`
Applies to:	`p`
Inherited:	yes
Percentages:	N/A
Animatable:	discrete

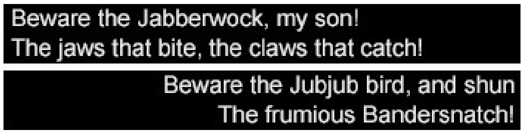
If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `start`.

The `tts:textAlign` style is illustrated by the following example.

Example Fragment – Text Align

```
<region xml:id="r1">
  <style tts:extent="355px 43px"/>
  <style tts:origin="0px 0px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:textAlign="start"/>
</region>
<region xml:id="r2">
  <style tts:extent="355px 43px"/>
  <style tts:origin="0px 47px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:textAlign="end"/>
</region>
...
<p region="r1">
  Beware the Jabberwock, my son!<br/>
  The jaws that bite, the claws that catch!
</p>
<p region="r2">
  Beware the Jubjub bird, and shun<br/>
  The frumious Bandersnatch!
</p>
```

Example Rendition – Text Align



Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!

Beware the Jubjub bird, and shun
The frumious Bandersnatch!

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.16.9.

8.2.19 tts:textDecoration

The `tts:textDecoration` attribute is used to specify a style property that defines a text decoration effect to apply to glyph areas or other inline areas that are generated by content flowed into a region.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`none` | [[`underline` | `noUnderline`] || [`lineThrough` | `noLineThrough`] || [`overline` | `noOverline`]]
Initial:	`none`
Applies to:	`span`
Inherited:	see special semantics
Percentages:	N/A
Animatable:	discrete

For the purpose of determining applicability of this style property, each character child of a `p` element is considered to be enclosed in an anonymous span.

If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `none`.

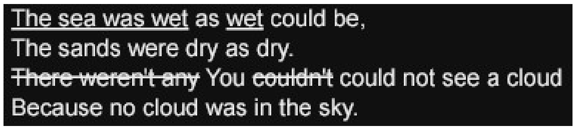
> **Note:**
>
> The syntax used above in defining the value of this property is based on the value component syntax defined in [CSS2], § 1.4.2.1. In essence, one or more of the values separated by `||` may appear in the property value in any order, such as `"noUnderline overline lineThrough"`.

The `tts:textDecoration` style is illustrated by the following example.

Example Fragment – Text Decoration

```
<region xml:id="r1">
  <style tts:extent="385px 82px"/>
  <style tts:origin="0px 0px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:padding="5px 2px"/>
  <style tts:textDecoration="underline"/>
</region>
...
<p region="r1">
  The sea was wet<span tts:textDecoration="noUnderline"> as </span>wet
  <span tts:textDecoration="noUnderline">
    could be,<br/>
    The sand was dry as dry.<br/>
    <span tts:textDecoration="lineThrough">There weren't any</span>
    You <span tts:textDecoration="lineThrough">couldn't</span>
    could not see a cloud<br/>
    Because no cloud was in the sky.
  </span>
</p>
```

Example Rendition – Text Decoration



The sea was wet as wet could be,
The sands were dry as dry.
~~There weren't any~~ You ~~couldn't~~ could not see a cloud
Because no cloud was in the sky.

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.17.4.

8.2.19.1 Special Semantics

The computed value of the property associated with this attribute consists of a 3-tuple, each of which denotes, respectively, whether an underline, line through, or overline decoration is to be applied to the affected text. Furthermore, when applying inheritance semantics, each of these sub-values is considered to be inherited separately from the others.

> **Note:**
>
> For example, if the computed value of `tts:textDecoration` that applies to a `div` (division) element is the tuple `(noUnderline, lineThrough, overline)` and a `p` (paragraph) element child of that `div` specifies `tts:textDecoration="noLineThrough"`, then the aggregate computed value that applies to the `p` element is `(noUnderline, noLineThrough, overline)`, which value is then inherited by children of the `p` element.

8.2.20 tts:textOutline

The `tts:textOutline` attribute is used to specify a style property that defines a text outline effect to apply to glyphs that are selected for glyph areas generated by content flowed into a region.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`none` | <color>? <length> <length>?
Initial:	`none`
Applies to:	`span`
Inherited:	yes
Percentages:	relative to this element's font size
Animatable:	discrete

For the purpose of determining applicability of this style property, each character child of a `p` element is considered to be enclosed in an anonymous span.

The value of this attribute consists of an optional <color> term followed by one or two <length> terms. If a *color* term is present, then it denotes the outline color; if no *color* term is present, the computed value of the `tts:color` applies. The first *length* term denotes the outline thickness and the second length term, if present, indicates the blur radius.

When a relative <length> value is specified for outline thickness or blur radius, then it is resolved in terms of the height component of the referenced font size or *Computed Cell Size*.

The <length> value(s) used to express thickness and blur radius must be non-negative.

If no blur radius is specified, i.e., only one <length> term is present, then the computed value of `0px` applies.

> **Note:**
>
> When a <length> expressed in cells is used in a `tts:textOutline` value, the cell's height applies. For example, if text outline thickness is specified as 0.1c, the cell resolution is 20 by 10, and the extent of the *Root Container Region* is 640 by 480, then the outline thickness will be a nominal 480 / 10 * 0.1 pixels, i.e., 4.8px, without taking into account rasterization effects.

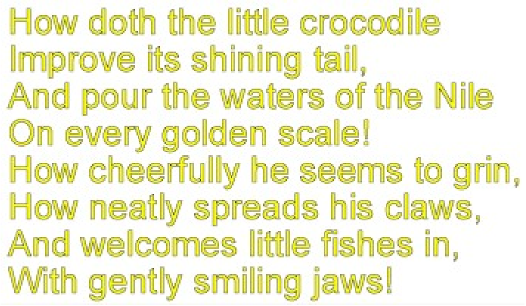
If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `none`.

The `tts:textOutline` style is illustrated by the following example.

Example Fragment – Text Outline

```
<region xml:id="r1">
  <style tts:backgroundColor="transparent"/>
  <style tts:color="yellow"/>
  <style tts:textOutline="black 2px 0px"/>
  <style tts:fontFamily="proportionalSansSerif"/>
  <style tts:fontSize="24px"/>
</region>
...
<p>
  How doth the little crocodile<br/>
  Improve its shining tail,<br/>
  And pour the waters of the Nile<br/>
  On every golden scale!<br/>
  How cheerfully he seems to grin,<br/>
  How neatly spreads his claws,<br/>
  And welcomes little fishes in,<br/>
  With gently smiling jaws!
</p>
```

Example Rendition – Text Outline



How doth the little crocodile
Improve its shining tail,
And pour the waters of the Nile
On every golden scale!
How cheerfully he seems to grin,
How neatly spreads his claws,
And welcomes little fishes in,
With gently smiling jaws!

8.2.21 tts:unicodeBidi

The `tts:unicodeBidi` attribute is used to specify a style property that defines a directional embedding or override according to the Unicode bidirectional algorithm.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`normal` | `embed` | `bidiOverride`
Initial:	`normal`
Applies to:	`p`, `span`
Inherited:	no
Percentages:	N/A
Animatable:	discrete

For the purpose of determining applicability of this style property, each character child of a `p` element is considered to be enclosed in an anonymous span.

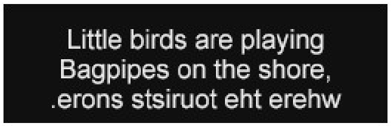
If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `normal`.

The `tts:unicodeBidi` style is illustrated by the following example.

Example Fragment – Unicode Bidirectionality

```
<region xml:id="r1">
  <style tts:extent="265px 84px"/>
  <style tts:padding="5px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="after"/>
  <style tts:textAlign="center"/>
</region>
...
<p region="r1">
  Little birds are playing<br/>
  Bagpipes on the shore,<br/>
  <span tts:unicodeBidi="bidiOverride" tts:direction="rtl">where the tourists snore.</span>
</p>
```

Example Rendition – Unicode Bidirectionality



Little birds are playing
Bagpipes on the shore,
.erons stsiruot eht erehw

Note:

The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.29.6.

8.2.22 tts:visibility

The `tts:visibility` attribute is used to specify a style property that defines whether generated areas are visible or not when rendered on a visual presentation medium.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`visible` | `hidden`
Initial:	`visible`
Applies to:	`body`, `div`, `p`, `region`, `span`
Inherited:	yes
Percentages:	N/A
Animatable:	discrete

For the purpose of determining applicability of this style property, each character child of a `p` element is considered to be enclosed in an anonymous span.

The `tts:visibility` style has no affect on content layout or composition, but merely determines whether composed content is visible or not.

If the computed value of the property associated with this attribute is `visible`, then areas generated by this element are rendered visible when presented on a visual medium. If the computed value is `hidden`, then they are not rendered visible, i.e., they are hidden; notwithstanding the foregoing, a descendant element of a hidden element is rendered visible if the computed value of this property is `visible` on the descendant.

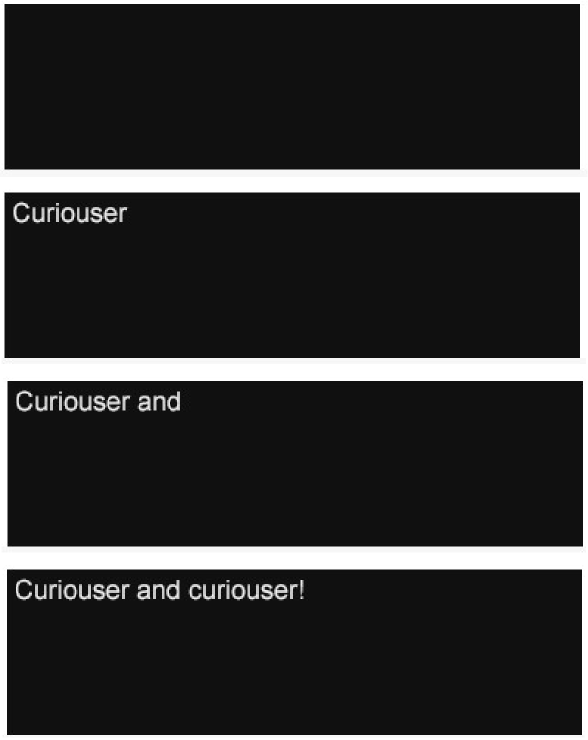
If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `visible`.

The `tts:visibility` style is illustrated by the following example.

Example Fragment – Visibility

```
<region xml:id="r1">
  <style tts:extent="398px 121px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style/>
</region>
...
<p region="r1" dur="4s">
  <span tts:visibility="hidden">
    <set begin="1s" tts:visibility="visible"/>
    Curiouser
  </span>
  <span tts:visibility="hidden">
    <set begin="2s" tts:visibility="visible"/>
    and
  </span>
  <span tts:visibility="hidden">
    <set begin="3s" tts:visibility="visible"/>
    curiouser!
  </span>
</p>
```

Example Rendition – Visibility



> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.30.17.

8.2.23 tts:wrapOption

The `tts:wrapOption` attribute is used to specify a style property that defines whether or not automatic line wrapping (breaking) applies within the context of the affected element.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`wrap` | `noWrap`
Initial:	`wrap`
Applies to:	`span`
Inherited:	yes
Percentages:	N/A
Animatable:	discrete

For the purpose of determining applicability of this style property, each character child of a `p` element is considered to be enclosed in an anonymous span.

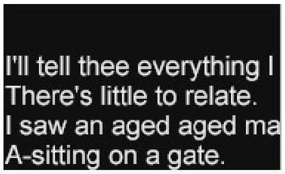
If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `wrap`.

The `tts:wrapOption` style is illustrated by the following example.

Example Fragment – Wrap Option

```
<region xml:id="r1">
  <style tts:extent="192px 117px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:displayAlign="after"/>
  <style tts:overflow="hidden"/>
  <style tts:wrapOption="noWrap"/>
</region>
...
<p>
  I'll tell thee everything I can:<br/>
  There's little to relate.<br/>
  I saw an aged aged man,<br/>
  A-sitting on a gate.
</p>
```

Example Rendition – Wrap Option



I'll tell thee everything I
There's little to relate.
I saw an aged aged ma
A-sitting on a gate.

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.16.13.

8.2.24 tts:writingMode

The `tts:writingMode` attribute is used to specify a style property that defines the block and inline progression directions to be used for the purpose of stacking block and inline areas within a region area.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`lrtb` | `rltb` | `tbrl` | `tblr` | `lr` | `rl` | `tb`
Initial:	`lrtb`
Applies to:	`region`
Inherited:	no
Percentages:	N/A
Animatable:	discrete

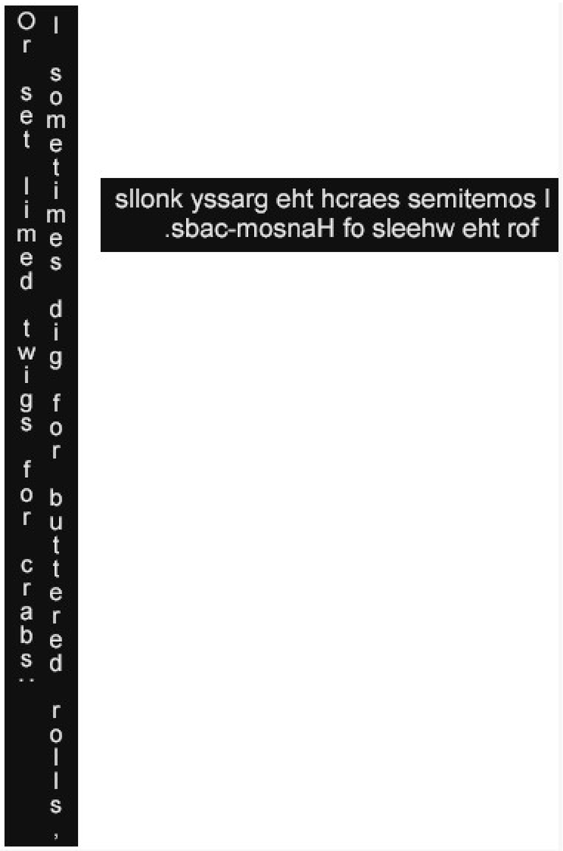
If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the value `lrtb`.

The `tts:writingMode` style is illustrated by the following example.

Example Fragment – Writing Mode

```
<region xml:id="r1">
  <style tts:extent="50px 570px"/>
  <style tts:origin="0px 0px"/>
  <style tts:padding="10px 3px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:writingMode="tbrl"/>
</region>
<region xml:id="r2">
  <style tts:extent="310px 50px"/>
  <style tts:origin="70px 120px"/>
  <style tts:padding="10px 3px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:writingMode="rltb"/>
</region>
...
<p region="r1">
  I sometimes dig for buttered rolls,<br/>
  Or set limed twigs for crabs:
</p>
<p region="r2" tts:direction="rtl" tts:unicodeBidi="bidiOverride">
  I sometimes search the grassy knolls for the wheels of Hansom-cabs.
</p>
```

Example Rendition – Writing Mode



I sometimes dig for buttered rolls,

Or set limed twigs for crabs:

sllonk yssarg eht hcraes semitemos I
.sbac-mosnaH fo sleehw eht rof

> **Note:**
>
> In the second paragraph in the above example that targets region `r2`, the `tts:unicodeBidi` and `tts:direction` properties are set to `bidiOverride` and `rtl`, respectively, in order to override the normally left-to-right directionality of characters in the Latin script.

> **Note:**
>
> The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.29.7.

8.2.25 tts:zIndex

The `tts:zIndex` attribute is used to specify a style property that defines the front-to-back ordering of region areas in the case that they overlap.

This attribute may be specified by any element type that permits use of attributes in the TT Style Namespace; however, this attribute applies as a style property only to those element types indicated in the following table.

Values:	`auto` | <integer>
Initial:	`auto`
Applies to:	`region`
Inherited:	no
Percentages:	N/A
Animatable:	discrete

If two areas are associated with the same Z-index value, then, if those areas overlap in space, the area(s) generated by lexically subsequent elements must be rendered over area(s) generated by lexically prior elements, where lexical order is defined as the postorder traversal of a *Document Instance*.

The `tt` element is considered to establish the root of the stacking context, as defined by [XSL 1.1], § 7.30.18.

If a computed value of the property associated with this attribute is not supported, then a presentation processor must use the closest supported value.

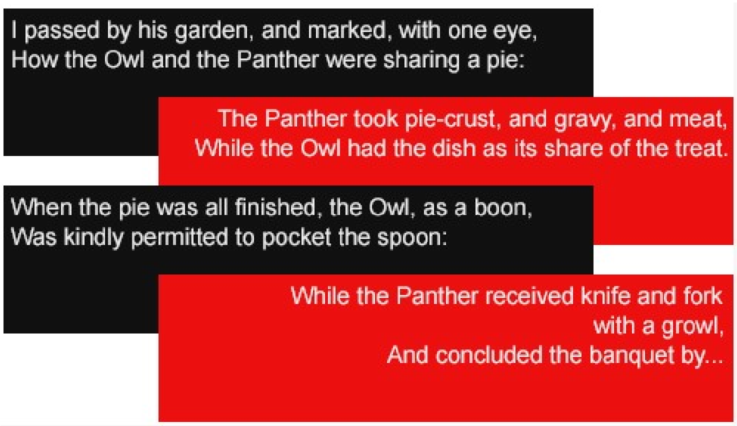
> **Note:**
>
> In this context, the phrase *closest supported value* means the value for which the Euclidean distance between the computed z-index and the supported z-index is minimized. If there are multiple closest supported values equally distant from the computed value, then the value least distant from 0, i.e., closest to the base stack level of the current stacking context, is used.

The `tts:zIndex` style is illustrated by the following example.

Example Fragment – Z Index

```
<region xml:id="r1">
  <style tts:origin="0px 0px"/>
  <style tts:extent="400px 100px"/>
  <style tts:padding="5px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:zIndex="0"/>
</region>
<region xml:id="r2">
  <style tts:origin="100px 60px"/>
  <style tts:extent="400px 100px"/>
  <style tts:padding="5px"/>
  <style tts:backgroundColor="red"/>
  <style tts:color="white"/>
  <style tts:textAlign="end"/>
  <style tts:zIndex="1"/>
</region>
<region xml:id="r3">
  <style tts:origin="0px 120px"/>
  <style tts:extent="400px 100px"/>
  <style tts:padding="5px"/>
  <style tts:backgroundColor="black"/>
  <style tts:color="white"/>
  <style tts:zIndex="2"/>
</region>
<region xml:id="r4">
  <style tts:origin="100px 180px"/>
  <style tts:extent="400px 100px"/>
  <style tts:padding="5px"/>
  <style tts:backgroundColor="red"/>
  <style tts:color="white"/>
  <style tts:textAlign="end"/>
  <style tts:zIndex="3"/>
</region>
...
<p region="r1">
  I passed by his garden, and marked, with one eye,<br/>
  How the Owl and the Panther were sharing a pie.
</p>
<p region="r2">
  The Panther took pie-crust, and gravy, and meat,<br/>
  While the Owl had the dish as its share of the treat.
</p>
<p region="r3">
  When the pie was all finished, the Owl, as a boon,<br/>
  Was kindly permitted to pocket the spoon:
</p>
<p region="r4">
  While the Panther received knife and fork<br/>
  with a growl,<br/>
  And concluded the banquet by...
</p>
```

Example Rendition – Z Index



I passed by his garden, and marked, with one eye,
How the Owl and the Panther were sharing a pie:

The Panther took pie-crust, and gravy, and meat,
While the Owl had the dish as its share of the treat.

When the pie was all finished, the Owl, as a boon,
Was kindly permitted to pocket the spoon:

While the Panther received knife and fork
with a growl,
And concluded the banquet by...

Note:

The semantics of the style property represented by this attribute are based upon that defined by [XSL 1.1], § 7.30.18.

8.3 Style Value Expressions

Style property values include the use of the following expressions:

- **8.3.1 <alpha>**
- **8.3.2 <color>**
- **8.3.3 <digit>**
- **8.3.5 <familyName>**
- **8.3.6 <genericFamilyName>**
- **8.3.7 <hexDigit>**
- **8.3.8 <integer>**
- **8.3.9 <length>**
- **8.3.10 <namedColor>**

In the syntax representations defined in this section, no linear whitespace (LWSP) is implied or permitted between tokens unless explicitly specified.

8.3.1 <alpha>

An <alpha> expression is used to express an opacity value, where 0 means fully transparent and 1 means fully opaque.

Syntax Representation – <alpha>

```
<alpha>
  : float
```

In the above syntax representation, the syntactic element *float* must adhere to the lexical representation defined by [XML Schema Part 2] § 3.2.4.1. If the value represented is less than 0.0, then it must be interpreted as equal to 0.0; similarly, if the value represented is greater than 1.0, then it must be interpreted as 1.0. The value NaN must be interpreted as 0.0.

A specified value for <alpha> should not be NaN, less than 0, or greater than 1.

If a presentation processor does not support a specific, valid opacity value, then it must interpret it as being equal to the closest supported value.

> **Note:**
>
> In this context, the phrase *closest supported value* means the value for which the Euclidean distance between the computed opacity and the supported opacity is minimized. If there are multiple closest supported values equally distant from the computed value, then the value most distant from 0, i.e., the greatest opacity, is used.

8.3.2 <color>

A <color> expression is used to specify a named color, exact RGB color triple, or exact RGBA color tuple, where the alpha component, if expressed, is maximum (255) at 100% opacity and minimum (0) at 0% opacity, and where the applicable color space is defined by [SRGB].

Syntax Representation – <color>

```
<color>
  : "#" rrggbb
  | "#" rrggbbaa
  | "rgb(" r-value "," g-value "," b-value ")"
  | "rgba(" r-value "," g-value "," b-value "," a-value ")"
  | <namedColor>

rrggbb
  :  <hexDigit>{6}

rrggbbaa
  :  <hexDigit>{8}

r-value | g-value | b-value | a-value
  : component-value

component-value
  : non-negative-integer                    // valid range: [0,255]

non-negative-integer
  : <digit>+
```

> **Note:**
>
> While permitted, the use of linear whitespace (LWSP) around the comma delimiters in `<color>` expressions is not recommended. The rationale for this recommendation is that some existing processors may not be tolerant of this usage.

When expressing RGB component values, these values are considered to **not** be premultiplied by alpha.

For the purpose of performing presentation processing such that non-opaque or non-transparent alpha or opacity values apply, then the semantics of compositing functions are defined with respect to the use of the [SRGB] color space for both inputs and outputs of the composition function.

> **Note:**
>
> The use of [SRGB] for the stated semantics of composition is not meant to prevent an actual processor from using some other color space either for internal or external purposes. For example, a presentation processor may ultimately convert the SRGB values used here to the YUV color space for rendition on a television device.

If a presentation processor does not support a specific, valid color or alpha value, then it must interpret it as being equal to the closest supported value.

> **Note:**
>
> In this context, the phrase *closest supported value* means the value for which the Euclidean distance between the computed color or alpha and the supported color or alpha in the RGB color space is minimized. If there are multiple closest supported values equally distant from the computed value, then the value least distant from opaque black `rgba(0,0,0,255)`, i.e., the closest to opaque black, is used.

8.3.3 <digit>

A <digit> is used to express integers and other types of numbers or tokens.

Syntax Representation – <digit>

```
<digit>
  : "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

8.3.4 <duration>

This section is non-normative.

> **Note:**
>
> The information from this section has been removed due to lack of any normative use within this specification. This section is retained in its empty form in order to prevent section renumbering.

8.3.5 <familyName>

A <familyName> expression specifies a font family name.

Syntax Representation – <familyName>

```
<familyName>
  : unquoted-string
  | quoted-string

unquoted-string
  : identifier ( lwsp identifier )*

lwsp
  : ( ' ' | '\t' | '\n' | '\r' )+

identifier
```

```
  : [-]? identifier-start identifier-following*

identifier-start
  : [_a-zA-Z]
  | non-ascii-or-c1
  | escape

identifier-following
  : [_a-zA-Z0-9-]
  | non-ascii-or-c1
  | escape

non-ascii-or-c1
  : [^\0-\237]

escape
  : '\\' char

quoted-string
  : double-quoted-string
  | single-quoted-string

double-quoted-string
  : '"' ( [^"\\] | escape )+ '"'

single-quoted-string
  : "'" ( [^'\\] | escape )+ "'"
```

In addition to adhering to the syntax rules specified above, the following semantic rules apply:

- the semantic value of a <familyName> expression is the semantic value of its `unquoted-string` or `quoted-string` non-terminal, according to whichever applies;
- the semantic value of an `unquoted-string` non-terminal is a pair <*quoted*, *content*>, where *quoted* is a boolean `false`, and where *content* is the result of appending the value of each `identifier` non-terminal, in lexical order, where the value of each identifier is preceded by a single SPACE (U+0020) character if it is not the first identifier;
- the semantic value of a `quoted-string` non-terminal is a pair <*quoted*, *content*>, where *quoted* is a boolean `true`, and where *content* is the unquoted content of the quoted string, i.e., the sequence of characters between the delimiting quotes.
- the semantic value of an `escape` non-terminal is the value of the escaped `char`;
- a <familyName> that takes the form of an `unquoted-string` that contains an `identifier` that starts with two `-` HYPHEN-MINUS (U+002D) characters must be considered to be invalid;
- a <familyName> that takes the form of an `unquoted-string` that contains a single `identifier` that matches (by case sensitive comparison) a <genericFamilyName> must be interpreted as that <genericFamilyName>;
- a <familyName> that takes the form of a `quoted-string` whose content (unquoted value) matches (by case sensitive comparison) a <genericFamilyName> must not be interpreted as that <genericFamilyName>, but as the actual name of a non-generic font family.

> **Note:**
>
> The syntactic element *`char`* is to be interpreted according to the `Char` production defined by [XML 1.0] §2.2.

> **Note:**
>
> The `{unicode}` escape mechanism defined by [CSS2] §4.1.1 is not supported by this syntax; rather, authors are expected to either (1) directly encode the character using the document encoding or (2) use an XML character reference according to [XML 1.0] §4.1. When a syntactically significant character needs to be used without its normal syntactic interpretation, it may be be escaped using the backslash (reverse solidus) `escape` non-terminal specified above.
>
> When using the backslash (reverse solidus) `escape` non-terminal, the above syntax does not place any restriction on what character may be escaped, e.g., `\\[\n\r\f0-9a-f]` are permitted. If one of these latter escapes appears in a <familyName> expression, then it will need to be converted to a `{unicode}` escape if it is to be used with a standard XSL-FO or CSS parser. In particular, a backslash followed by a newline is ignored by CSS, while it is not ignored by the above syntax. Such an unignored escaped newline would need to be represented using an equivalent `{unicode}` escape, such as `\a`, to order to express in CSS.

8.3.6 <genericFamilyName>

A <genericFamilyName> expression specifies a font family using a general token that indicates a class of font families.

The resolution of a generic family name to a concrete font instance is considered to be implementation dependent, both in the case of content authoring and content interpretation.

Syntax Representation – <genericFamilyName>

```
<genericFamilyName>
  : "default"
  | "monospace"
  | "sansSerif"
  | "serif"
  | "monospaceSansSerif"
  | "monospaceSerif"
  | "proportionalSansSerif"
  | "proportionalSerif"
```

The mapping between a generic (font) family name and an actual font is not determined by this specification; however, the distinction of monospace versus proportional and serif versus sans-serif should be maintained if possible when performing presentation.

If a generic (font) family name of `monospace` is specified, then it may be interpreted as equivalent to either `monospaceSansSerif` or `monospaceSerif`. The generic family names `sansSerif` and `serif` are to be interpreted as equivalent to `proportionalSansSerif` and `proportionalSerif`, respectively.

If the generic family name `default` is specified (or implied by an initial value), then its typographic characteristics are considered to be implementation dependent; however, it is recommended that this default font family be mapped to a monospaced, sans-serif font.

8.3.7 <hexDigit>

A <hexDigit> is used to express integers and other types of numbers or tokens that employ base 16 arithmetic.

For the purpose of parsing, a distinction must not be made between lower and upper case.

Syntax Representation – <hexDigit>

```
<hexDigit>
  : <digit>
  | "a" | "b" | "c" | "d" | "e" | "f"
  | "A" | "B" | "C" | "D" | "E" | "F"
```

8.3.8 <integer>

An <integer> expression is used to express an arbitrary, signed integral value.

Syntax Representation – <integer>

```
<integer>
  : ( "+" | "-" )? <digit>+
```

8.3.9 <length>

A <length> expression is used to express either a coordinate component of point in a cartesian space or a distance between two points in a cartesian space.

Syntax Representation – <length>

```
<length>
  : scalar
  | percentage

scalar
  : number units

percentage
  : number "%"

sign
  : "+" | "-"

number
  : sign? non-negative-number

non-negative-number
  : non-negative-integer
  | non-negative-real

non-negative-integer
  : <digit>+

non-negative-real
  : <digit>* "." <digit>+

units
  : "px"
  | "em"
  | "c"                                     // abbreviation of "cell"
```

It is an error to omit the units component of a scalar length value.

The semantics of the unit of measure `px` (pixel) are as defined by [XSL 1.1], § 5.9.13.

When specified relative to a font whose size is expressed as a single length measure or as two length measures of equal length, the unit of measure `em` is considered to be identical to that defined by [XSL 1.1], § 5.9.13; however, when specified relative to a font whose size is expressed as two length measures of non-equal lengths, then one `em` is equal to the inline progression dimension of the anamorphically scaled font when used to specify lengths in the inline progression direction and equal to the block progression dimension of the scaled font when used to specify lengths in the block progression direction.

The semantics of the unit of measure `c` (cell) are defined by the parameter **6.2.1 ttp:cellResolution**.

8.3.10 <namedColor>

A <namedColor> is used to express an RGBA color with a convenient name, and where the applicable color space is defined by [SRGB].

For the purpose of parsing, a distinction must not be made between lower and upper case.

Syntax Representation – <namedColor>

```
<namedColor>
  : "transparent"                           // #00000000
  | "black"                                 // #000000ff
  | "silver"                                // #c0c0c0ff
  | "gray"                                  // #808080ff
  | "white"                                 // #ffffffff
  | "maroon"                                // #800000ff
  | "red"                                   // #ff0000ff
  | "purple"                                // #800080ff
  | "fuchsia"                               // #ff00ffff
  | "magenta"                               // #ff00ffff (= fuchsia)
  | "green"                                 // #008000ff
  | "lime"                                  // #00ff00ff
  | "olive"                                 // #808000ff
  | "yellow"                                // #ffff00ff
  | "navy"                                  // #000080ff
  | "blue"                                  // #0000ffff
  | "teal"                                  // #008080ff
  | "aqua"                                  // #00ffffff
  | "cyan"                                  // #00ffffff (= aqua)
```

> **Note:**
>
> Except for `transparent`, the set of named colors specified above constitutes a proper subset of the set of named colors specified by [SVG 1.1], § 4.2.

8.3.11 <quotedString>

This section is non-normative.

> **Note:**
>
> The information from this section has been incorporated into **8.3.5 <familyName>**. This section is retained in its empty form in order to prevent section renumbering.

8.3.12 <string>

This section is non-normative.

> **Note:**
>
> The information from this section has been incorporated into **8.3.5 <familyName>**. This section is retained in its empty form in order to prevent section renumbering.

8.4 Style Resolution

This section defines the semantics of style resolution in terms of a standard processing model as follows:

- **8.4.1 Style Association**
- **8.4.2 Style Inheritance**
- **8.4.3 Style Resolution Value Categories**
- **8.4.4 Style Resolution Processing**

Any implementation of this model is permitted provided that the externally observable results are consistent with the results produced by this model.

> **Note:**
>
> The semantics of style resolution employed here are based upon [XSL 1.1], § 5.

8.4.1 Style Association

Style association is a sub-process of **8.4.4 Style Resolution Processing** used to determine the specified style set of each content and layout element.

Style matter may be associated with content and layout matter in a number of ways:

- inline styling
- referential styling
- chained referential styling

In addition to the above, style matter may be associated with layout matter using:

- nested styling

8.4.1.1 Inline Styling

Style properties may be expressed in an inline manner by direct specification of an attribute from the TT Style Namespace on the affected element. When expressed in this manner, the association of style information is referred to as *inline styling*.

Style properties associated by inline styling are afforded a higher priority than all other forms of style association.

Example – Inline Styling

```
<p tts:color="white">White 1 <span tts:color="yellow">Yellow</span> White 2</p>
```

> **Note:**
>
> In the above example, the two text fragments `"White 1 "` and `" White 2"`, which are interpreted as anonymous spans, are not associated with a color style property; rather, they inherit their color style from their parent `p` element as described in **8.4.2.1 Content Style Inheritance** below.

8.4.1.2 Referential Styling

Style properties may be expressed in an out-of-line manner and referenced by the affected element using the `style` attribute. When expressed in this manner, the association of style information is referred to as *referential styling*.

If a `style` attribute specifies multiple references, then those references are evaluated in the specified order, and that order applies to resolution of the value of a style property in the case that it is specified along multiple reference paths.

The use of referential styling is restricted to making reference to `style` element descendants of a `styling` element. It is considered an error to reference a `style` element that is a descendant of a `layout` element.

> **Note:**
>
> The use of referential styling encourages the reuse of style specifications while sacrificing locality of reference.

> **Note:**
>
> A single Content element may be associated with style properties by a hybrid mixture of inline and referential styling, in which case inline styling is given priority as described above by **8.4.1.1 Inline Styling**.

Example – Referential Styling

```
<style xml:id="s1" tts:color="white"/>
<style xml:id="s2" tts:color="yellow"/>
...
<p style="s1">White 1 <span style="s2">Yellow</span> White 2</p>
```

> **Note:**
>
> In the above example, the two text fragments `"White 1 "` and `" White 2"`, which are interpreted as anonymous spans, are not associated with a color style property; rather, they inherit their color style from their parent `p` element as described in **8.4.2.1 Content Style Inheritance** below.

8.4.1.3 Chained Referential Styling

Style properties may be expressed in an out-of-line manner and may themselves reference other out-of-line style properties, thus creating a chain of references starting at the affected element. When expressed in this manner, the association of style information is referred to as *chained referential styling*.

A loop in a sequence of chained style references must be considered an error.

The use of referential styling is restricted to making reference to `style` element descendants of a `styling` element. It is considered an error to reference a `style` element that is a descendant of a `layout` element.

> **Note:**
>
> The use of chained referential styling encourages the grouping of style specifications into general and specific sets, which further aids in style specification reuse.

> **Note:**
>
> A single Content element may be associated with style properties by a hybrid mixture of inline, referential styling, and chained referential styling, in which case inline styling is given priority as described above by **8.4.1.1 Inline Styling**.

Example – Chained Referential Styling

```
<style xml:id="s1" tts:color="white" tts:fontFamily="monospaceSerif"/>
<style xml:id="s2" style="s1" tts:color="yellow"/>
...
<p style="s1">White Monospace</p>
<p style="s2">Yellow Monospace</p>
```

> **Note:**
>
> In the above example, the text of the second paragraph is yellow, since `tts:color='yellow'` effectively overwrites (is merged over) the `tts:color='white'` that style `s2` obtains by a reference to style `s1`.

8.4.1.4 Nested Styling

Style properties may be expressed in a nested manner by direct specification of one or more `style` element children of the affected element. When expressed in this manner, the association of style information is referred to as *nested styling*.

Style properties associated by nested styling are afforded a lower priority than inline styling but with higher priority than referential styling.

Example – Nested Styling

```
<region xml:id="r1">
  <style tts:extent="128px 66px"/>
  <style tts:origin="0px 0px"/>
  <style tts:displayAlign="center"/>
</region>
```

> **Note:**
>
> In this version of this specification, nested styling applies only to the `region` element.

8.4.2 Style Inheritance

Style inheritance is a sub-process of **8.4.4 Style Resolution Processing** used to determine the specified style set of each content and layout element.

Styles are further propagated to content matter using:

- content style inheritance
- region style inheritance

For the purpose of determining inherited styles, the element hierarchy of an intermediate synchronic document form of a *Document Instance* must be used, where such intermediate forms are defined by **9.3.3 Intermediate Synchronic Document Construction**.

> **Note:**
>
> The intermediate synchronic document form is utilized rather than the original form in order to facilitate region inheritance processing.

8.4.2.1 Content Style Inheritance

Style properties are inherited from ancestor Content elements within an intermediate synchronic document if a style property is not associated with a Content element (or an anonymous span) and the style property is designated as inheritable.

> **Note:**
>
> The `tt` element is not a Content element; consequently, the `body` element is the outermost element from which content style inheritance occurs.

If a style property is determined to require inheritance, then the inherited value must be the value of the same named style property in the computed style set of the element's immediate ancestor element within the applicable intermediate synchronic document.

Example – Content Style Inheritance

```
<p tts:fontFamily="monospaceSansSerif">
  <span tts:color="yellow">Yellow Monospace</span>
</p>
```

> **Note:**
>
> In the above example, the `span` element that encloses the character items `Yellow Monospace` is not associated with a `tts:fontFamily` style property and this property is inheritable; therefore, the value of the `tts:fontFamily` style is inherited from the computed style set of the ancestor `p` element, and is added to the specified style set of the `span` element.

8.4.2.2 Region Style Inheritance

Style properties are inherited from a region element in the following case:

1. if a style property *P* is not associated with a Content element or an anonymous span *E* and the style property is designated as inheritable, and
2. if that style property *P* is in the computed style set of region *R*, and
3. if that element *E* is flowed into (presented within) region *R*.

Example – Region Style Inheritance

```
<region xml:id="r1">
  <style tts:color="yellow"/>
  <style tts:fontFamily="monospaceSerif"/>
</region>
...
<p region="r1">Yellow Monospace</p>
```

> **Note:**
>
> In the above example, the anonymous span that encloses the character items `Yellow Monospace` effectively inherits the `tts:color` and `tts:fontFamily` styles specified on the `region` element into which the `p` element is flowed (presented).

8.4.3 Style Resolution Value Categories

During style resolution, layout, and presentation processing, three categories of style property values are distinguished as follows:

- specified values
- computed values
- used values
- actual values

8.4.3.1 Specified Values

Values of style properties that are associated with or inherited by an element or anonymous span are referred to as *specified values*. The set of all specified style properties of a given element is referred to as the *specified style set* of that element.

8.4.3.2 Computed Values

When style properties are specified using relative value expressions, such as a named color, a relative unit (e.g., cell), or a percentage, then they need to be further resolved into absolute units, such as an RGB triple, pixels, etc.

During the style resolution process, all specified style values are reinterpreted (or recalculated) in absolute terms, and then recorded as *computed values*. The set of all computed style properties of a given element is referred to as the *computed style set* of that element.

When a style value is inherited, either explicitly or implicitly, it is the computed value of the style that is inherited from an ancestor element. This is required since the resolution of certain relative units, such as percentage, require evaluating the expression in the immediate (local) context of reference, and not in a distant (remote) context of reference where the related (resolving) expression is not available.

8.4.3.3 Used Values

Subsequent to style inheritance, a computed value may require further resolution at layout or presentation time when such resolution is required by the semantics of a specific property or the information required to perform the resolution is not available until layout or presentation processing occurs. Such a value is referred to as a *used value*, which need not be the same value as the final actual value described below.

> **Note:**
>
> The determination of a used value is always preceded by the inheritance of a computed value; that is to say, used value computation is nominally performed after the **8.4.4 Style Resolution Processing** completes.

> **Note:**
>
> See also [CSS2], §6.1.3..

8.4.3.4 Actual Values

During the actual presentation process, other transformations occur that map some value expressions to concrete, physical values. For example, the colors of computed style values are further subjected to closest color approximation and gamma correction during the display process. In addition, length value expressions that use pixels in computed style values are considered to express logical rather than physical (device) pixels. Consequently, these logical pixels are subject to being further transformed or mapped to physical (device) pixels during presentation.

The final values that result from the logical to device mapping process are referred to as *actual values*. The set of all actual style properties of a given element is referred to as the *actual style set* of that element.

> **Note:**
>
> More than one set of actual values may be produced during the process of presentation. For example, a TTML presentation processor device may output an RGBA component video signal which is then further transformed by an NTSC or PAL television to produce a final image. In this case, both color and dimensions may further be modified prior to presentation.

> **Note:**
>
> In general, a TTML presentation processor will not have access to actual style set values; as a consequence, no further use or reference to actual values is made below when formally describing the style resolution process.

8.4.4 Style Resolution Processing

The process of style resolution is defined herein as the procedure (and results thereof) for resolving (determining) the computed values of all style properties that apply to content and layout elements:

The process described here forms an integral sub-process of **9.3 Region Layout and Presentation**.

8.4.4.1 Conceptual Definitions

For the purpose of interpreting the style resolution processing model specified below, the following conceptual definitions apply:

[style property]

a style property, *P*, is considered to consist of a tuple `[name, value]`, where the name of the property is a tuple `[namespace value, unqualified name]` and the value of the property is a tuple `[category, type, value expression]`

Example – conceptual style property

```
[
  ["http://www.w3.org/ns/ttml#styling", "color"],
  ["specified", color, "red"]
]
```

[style (property) set]

a style (property) set consists of an unordered collection of style properties, where no two style properties within the set have an identical name, where by "identical name" is meant equality of namespace value of name tuple and unqualified name of name tuple;

in a specified style (property) set, the category of each style property is "specified"; a specified style (property) set of an element *E* is referred to as *SSS(E)*;

Example – conceptual (specified) style (property) set

```
{
  [
    ["http://www.w3.org/ns/ttml#styling", "backgroundColor"],
    ["specified", color, 0x00FF00 ]
  ],
  [
    ["http://www.w3.org/ns/ttml#styling", "color"],
    ["specified", color, "red" ]
  ],
  [
    ["http://www.w3.org/ns/ttml#styling", "fontSize"],
    ["specified", length, "1c" ]
  ],
  [
    ["http://www.w3.org/ns/ttml#styling", "lineHeight"],
    ["specified", length, "117%" ]
  ]
}
```

in a computed style (property) set, the category of each style property is either "specified" or "computed"; a computed style (property) set of an element *E* is referred to as *CSS(E)*;

Example – conceptual (computed) style (property) set

```
{
  [
    ["http://www.w3.org/ns/ttml#styling", "backgroundColor"],
    ["specified", color, 0x00FF00 ]
  ],
  [
    ["http://www.w3.org/ns/ttml#styling", "color"],
    ["computed", color, 0xFF0000 ]
  ],
  [
    ["http://www.w3.org/ns/ttml#styling", "fontSize"],
    ["computed", length, "24px" ]
  ],
  [
    ["http://www.w3.org/ns/ttml#styling", "lineHeight"],
    ["computed", length, "28px" ]
  ]
}
```

[style (property) merging]

a style property P_{new} is merged into a style (property) set, SS, as follows: if a style property P_{old} is already present in SS where the name of P_{new} is identical to the name of P_{old}, then replace P_{old} in SS with P_{new}; otherwise, add P_{new} to SS;

[style (property) set merging]

a style (property) set SS_{new} is merged into an existing style (property) set SS_{old} as follows: for each style property P_{new} in SS_{new}, merge P_{new} into SS_{old};

8.4.4.2 Specified Style Set Processing

The specified style set SSS of an element or anonymous span E, $SSS(E)$, is determined according to the following ordered rules:

1. **[initialization]** initialize the specified style set SSS of E to the empty set;
2. **[referential and chained referential styling]** for each `style` element S_{REF} referenced by a `style` attribute specified on E, and in the order specified in the `style` attribute, then, if S_{REF} is a descendant of a `styling` element, merge the specified style set of S_{REF}, $SSS(S_{REF})$, into the specified style set of E, $SSS(E)$;
3. **[nested styling]** for each nested `style` element child S_{NEST} of E, and in the specified order of child elements, merge the specified style set of S_{NEST}, $SSS(S_{NEST})$, into the specified style set of E, $SSS(E)$;
4. **[inline styling]** for each style property P expressed as a specified styling attribute of E, merge P into the specified style set of E, $SSS(E)$;
5. **[animation styling]** for each style property P expressed as a specified styling attribute of an immediate animation (`set`) element child of element E, merge P into the specified style set of E, $SSS(E)$;
6. **[implicit inheritance]** if the element type of E is not the styling element type `style`, then for each style property P in the set of style properties defined above in **8.2 Styling Attribute Vocabulary**, perform the following ordered sub-steps:
   a. if P is present in the specified style set of E, $SSS(E)$, then continue to the next style property;
   b. if P is inheritable and E is a `region` element, or if P is not inheritable and applies to E, then set P' to the initial value of property P if such an initial value is defined, where the initial value of a property is determined according to the specific property definition found above in **8.2 Styling Attribute Vocabulary**;

c. if *P* is inheritable and the element type of *E* is a Content element type or anonymous span, then set *P′* to the result of looking up the value of *P* in the computed style set of the immediate ancestor element of *E*, i.e., *CSS(PARENT(E))*;

d. if the value of *P′* is not undefined, then merge *P′* into the specified style set of *E*, *SSS(E)*.

8.4.4.3 Computed Style Set Processing

The computed style set *CSS* of an element or anonymous span *E*, *CSS(E)*, is determined according to the following ordered rules:

1. **[resolve specified styles]** determine (obtain) the specified style set *SSS* of *E*, namely, *SSS(E)*, in accordance with **8.4.4.2 Specified Style Set Processing**;
2. **[initialization]** initialize *CSS(E)* to a (deep) copy of *SSS(E)*;
3. **[filter]** if *E* is a `style` element, then return *CSS(E)* as the resulting computed style set without further resolution; otherwise, continue with the next rule;
4. **[relative value resolution]** for each style property *P* in *CSS(E)*, where the value type of *P* is relative, perform the following ordered sub-steps:
   a. replace the relative value of *P* with an equivalent, non-relative (computed) value;

   b. set the category of *P* to "computed";

> **Note:**
>
> As a result of the filtering rule above, the computed style set of a `style` element includes only specified values, in which case relative value expressions remain relative; consequently, the resolution of relative value expressions (that may be assigned by means of referential style association) always takes place in the context of a layout or Content element which has a presentation context, and not in the non-presentation, declaration context of a referentiable `style` element.

> **Note:**
>
> See individual style properties for additional inheritance semantics, if any.

8.4.4.4 Style Resolution Process

The top-level style resolution process is defined as follows: using a preorder traversal of each element and anonymous span, *E*, of an intermediate synchronic document, DOC_{inter}, perform the following ordered sub-steps:

1. **[filter]** if the element type of *E* is not the styling element type `style`, is not the layout element type `region`, and is not one of the Content element types `body`, `div`, `p`, `span`, `br`, or anonymous span, then continue to the next element in the preorder traversal;
2. **[resolve computed styles]** determine (obtain) the computed style set *CSS* of *E*, namely, *CSS(E)*, in accordance with **8.4.4.3 Computed Style Set Processing**.

9 Layout

This section specifies the *layout* matter of the core vocabulary catalog, where layout is to be understood as a separable layer of information that applies to content and that denotes authorial intentions about the presentation of that content.

> **Note:**
>
> The two layers of layout and style matter are considered to be independently separable. Layout matter specifies one or more spaces or areas into which content is intended to be presented, while style matter specifies the manner in which presentation occurs within the layout.
>
> In certain cases, a content author may choose to embed (inline) style matter directly into layout or content matter. In such cases, an alternative exists – use of referential styling – in which the style matter is not embedded (inlined).

9.1 Layout Element Vocabulary

The following elements specify the structure and principal layout aspects of a *Document Instance*:

- **9.1.1 layout**
- **9.1.2 region**

9.1.1 layout

The `layout` element is a container element used to group layout matter, including metadata that applies to layout matter.

The `layout` element accepts as its children zero or more elements in the `Metadata.class` element group, followed by zero or more `region` elements.

XML Representation – Element Information Item: layout

```
<layout
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*, region*
</layout>
```

To the extent that time semantics apply to the content of the `layout` element, the implied time interval of this element is defined to be coterminous with the *Root Temporal Extent*.

9.1.2 region

The `region` element is used to define a rectangular space or area into which content is to be flowed for the purpose of presentation.

> **Note:**
>
> The rectangular area of a region is explicitly not constrained to be contained within the *Root Container Region*. In particular, the origin components of a region may be negative, and the extent (width and height) components of a region may be greater than the width and height of the *Root Container Region*. Whether a presentation processor clips such a region to the *Root Container Region* is implementation dependent, and not prescribed by this specification.

In addition, and in accordance with **8.4.2.2 Region Style Inheritance**, the `region` element may be used to specify inheritable style properties to be inherited by content that is flowed into it.

The `region` element accepts as its children zero or more elements in the `Metadata.class` element group, followed by zero or more elements in the `Animation.class` element group, followed by zero or more `style` elements.

Any metadata specified by children in the `Metadata.class` element group applies semantically to the `region` element and its descendants as a whole. Any animation elements specified by children in the `Animation.class` element group apply semantically to the `region` element. Any `style` child element must be considered a local style definition that applies only to the containing `region` element, i.e., does not apply for resolving referential styling (but does apply for region style inheritance).

XML Representation – Element Information Item: region

```
<region
  begin = <timeExpression>
  dur = <timeExpression>
  end = <timeExpression>
  style = IDREFS
  timeContainer = (par|seq)
  ttm:role = string
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute in TT Style namespace}
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*, Animation.class*, style*
</region>
```

If `begin` and (or) `end` attributes are specified on a `region` element, then they specify the beginning and (or) ending points of a time interval during which the region is eligible for activation and with respect to which animation child elements of the region are timed. If specified, these begin and end points are relative to the time interval of the nearest ancestor element associated with a time interval, irregardless of whether that interval is explicit or implied.

The nearest ancestor element of a `region` element that is associated with a time interval is the `layout` element.

If a `dur` attribute is specified on the `region` element, then it specifies the simple duration of the region.

For the purpose of determining the semantics of presentation processing, a region that is temporally inactive must not produce any visible marks when presented on a visual medium.

> **Note:**
>
> A `region` element may be associated with a time interval for two purposes: (1) in order to temporally bound the presentation of the region and its content, and (2) to provide a temporal context in which animations of region styles may be effected.
>
> For example, an author may wish to specify a region that is otherwise empty, but may have a visible background color to be presented starting at some time and continuing over the region's duration. The simple duration of the region serves additionally to scope the presentation effects of content that is targeted to the region. An author may also wish to move a region within the *Root Container Region* or change a region's background color by means of animation effects. In both of these cases, it is necessary to posit an active time interval for a region.

If no `timeContainer` attribute is specified on a `region` element, then it must be interpreted as having *parallel* time containment semantics.

If a `ttm:role` attribute is specified on a `region` element, then it must adhere to the value syntax defined by **Syntax Representation – ttm:role**, and where the roles identified by this attribute express the semantic roles of the region independently from the semantic roles of any content targeted to (associated with) the region.

9.2 Layout Attribute Vocabulary

This section defines the **9.2.1 region** attribute used with Content elements.

9.2.1 region

The `region` attribute is used to reference a `region` element which defines a space or area into which a Content element is intended to be flowed.

If specified, the value of a `region` attribute must adhere to the `IDREF` data type defined by [XML Schema Part 2], § 3.3.9, and, furthermore, this IDREF must reference a `region` element which has a `layout` element as an ancestor.

The `region` attribute may be specified by an instance of the following element types:

- `body`
- `div`
- `p`
- `span`

> **Note:**
>
> See **9.3 Region Layout and Presentation** below for further information on content flow in a region.

9.3 Region Layout and Presentation

This section defines the semantics of region layout and presentation in terms of a standard processing model as follows:

- **9.3.1 Default Region**
- **9.3.3 Intermediate Synchronic Document Construction**
- **9.3.4 Synchronic Flow Processing**
- **9.3.5 Elaborated Example (Informative)**

Any implementation is permitted provided that the externally observable results are consistent with the results produced by this model.

9.3.1 Default Region

If a *Document Instance* does not specify a `region` element, then a *default region* is implied with the following characteristics:

- the identity of the default region is considered to be anonymous;
- the extent of the default region is the same as the extent of the *Root Container Region*;
- the temporal interval of the default region is the same as the interval defined by the *Root Temporal Extent*;

Furthermore, if no `region` element is specified, then the `region` attribute must not be specified on any Content element in the *Document Instance*.

If a default region is implied for a given *Document Instance*, then the `body` element is implicitly targeted to (associated with) the default region.

When implying a default region, the *Document Instance* is to be treated as if a `region` element and its parent `layout` element were specified in a `head` element, and a matching `region` attribute were specified on the `body` element as shown in the following example:

Example – Implied Default Region

```
<tt xml:lang="" xmlns="http://www.w3.org/ns/ttml">
  <head>
    <layout>
      <region xml:id="anonymous"/>
    </layout>
  </head>
  <body region="anonymous"/>
</tt>
```

> **Note:**
>
> In the above example, a default `region` element and `region` attribute are implied. In addition, a `layout` container element is implied for the implied `region` element.

9.3.2 Anonymous Span Construction

For the purposes of performing presentation processing, anonymous spans are created according to the following **[construct anonymous spans]** procedure:

[construct anonymous spans]

1. for each significant text node in a Content element, synthesize an anonymous span to enclose the text node, substituting the new anonymous span for the original text node child in its sibling and parent hierarchy;
2. for each contiguous sequence of anonymous spans, replace the sequence with a single anonymous span which contains a sequence of text nodes representing the individual text node children of the original sequence of anonymous spans;
3. for each span element whose child is a single anonymous span, replace the anonymous span with its sequence of child text nodes;

9.3.3 Intermediate Synchronic Document Construction

For the purposes of performing presentation processing, the active time duration of a *Document Instance* is divided into a sequence of time coordinates where at each time coordinate, some element becomes temporally active or inactive, then, at each such time coordinate, a *Document Instance* is mapped from its original, source form, DOC_{source} , to an intermediate synchronic document form, DOC_{inter} , according to the **[construct intermediate document]** procedure:

[construct intermediate document]

1. for each temporally active region *R*, replicate the sub-tree of DOC_{source} headed by the `body` element;
2. evaluating this sub-tree in a postorder traversal, prune elements if any of the following conditions is true:
   a. they are neither Content nor Animation elements; or
   b. they are temporally inactive; or
   c. they are empty and neither Animation elements nor br elements; or
   d. they are Content elements and aren't associated with region *R* according to the **[associate region]** procedure.
3. if the pruned sub-tree is non-empty, then reparent it to the *R* element;
4. finally, after completing the above steps, prune the original `body` element from the intermediate document, then prune all `region`, `begin`, `end`, and `dur` attributes, which are no longer semantically relevant;

> **Note:**
>
> In this section, the term *prune*, when used in reference to an element, means that the element is to be removed from its parent's children, which, in turn, implies that the descendants of the pruned element will no longer be descendants of the element's parent. When *prune* is used in reference to an attribute, it means that attribute is to be removed from its associated (owning) element node.

[associate region]

A Content element is associated with a region according to the following ordered rules, where the first rule satisfied is used and remaining rules are skipped:

1. if the element specifies a `region` attribute, then the element is associated with the region referenced by that attribute;
2. if some ancestor of that element specifies a `region` attribute, then the element is associated with the region referenced by the most immediate ancestor that specifies this attribute;
3. if the element contains a descendant element that specifies a `region` attribute, then the element is associated with the region referenced by that attribute;
4. if a default region was implied (due to the absence of any `region` element), then the element is associated with the default region;
5. the element is not associated with any region.

The result of performing the processing described above will be a sequence of *N* intermediate synchronic *Document Instances*, $DOC_{inter_0} \ldots DOC_{inter_{N-1}}$.

> **Note:**
>
> Where an implementation is able to detect significant similarity between two adjacent synchronic *Document Instances*, DOC_{inter_N} $DOC_{inter_{N-1}}$, then the implementation can apply processing to make the transition between presenting the two instances as smooth as possible, e.g., as described by [CEA-608-E], § C.3, and [CC-DECODER-REQ].

9.3.4 Synchronic Flow Processing

Subsequent to performing a temporal (synchronic) slice and subsequent remapping of regionally selected content hierarchy, the resulting intermediate synchronic document is subjected to a flow transformation step that produces a rooted flow object tree represented as an XSL FO document instance as defined by [XSL 1.1], and semantically extended by TTML specific style properties that have no XSL FO counterpart.

> **Note:**
>
> In this section, the use of XSL FO is intended to be conceptual only, employed solely for the purpose of defining the normative presentation semantics of TTML. An actual implementation of this algorithm is not required to create or process XSL-FO representations. In particular, it is possible to implement these semantics using alternative presentation models, such as Cascading Style Sheets (CSS).

Each intermediate synchronic document produced by **9.3.3 Intermediate Synchronic Document Construction** is mapped to an XSL FO document instance, *F*, as follows:

1. perform the **[construct anonymous spans]** procedure;
2. resolve styles according to **8.4.4.4 Style Resolution Process**;

3. map the `tt` element to an `fo:root` element, populated initially with an `fo:layout-master-set` element that contains a valid `fo:simple-page-master` that, in turn, contains an `fo:region-body` child, where the extent of the *Root Container Region* expressed on the `tt` element is mapped to `page-width` and `page-height` attributes on the `fo:simple-page-master` element;
4. map the `layout` element to an `fo:page-sequence` element and a child `fo:flow` element that reference the page master and page region defined by the simple page master produced above;
5. map each non-empty `region` element to an `fo:block-container` element with an `absolute-position` attribute with value `absolute`, with `top`, `left`, `bottom`, and `right` attributes that express a rectangle equivalent to the region's origin, extent (including padding), and with a `line-stacking-strategy` attribute with value `line-height`;

   > **Note:**
   >
   > The region's extent corresponds with the allocation rectangle of the block area generated by the `fo:block-container`.

6. for each `body`, `div`, and `p` element that is not associated with a `tts:display` style property with the value `none`, map the element to a distinct `fo:block` element, populating the style properties of `fo:block` by using the computed style set associated with each original TTML Content element;
7. for the resulting `fo:block` formatting object produced in the previous step that corresponds to the `body` element, perform the following ordered sub-steps:
   a. if the `display-align` style property of this `fo:block` has the value `center` or `after`, then synthesize and insert as the first child of this `fo:block` an empty `fo:block` with the following attributes: `space-after.optimum`, `space-after.maximum`, and `space-after.conditionality`, where the value of the former two attributes is the height or width of the containing `fo:block-container` element, whichever of these is designated as the block progression dimension, and where the value of the last is `retain`;
   b. if the `display-align` style property of this `fo:block` has the value `center` or `before`, then synthesize and insert as the last child of this `fo:block` an empty `fo:block` with the following attributes: `space-after.optimum`, `space-after.maximum`, and `space-after.conditionality`, where the value of the former two attributes is the height or width of the containing `fo:block-container` element, whichever of these is designated as the block progression dimension, and where the value of the last is `retain`;
8. for each `span` element that is not associated with a `tts:display` style property with the value `none` and for each anonymous span that is a child of a `p` or `span` element, map the element or sequence of character items to a distinct `fo:inline` element, populating the style properties of `fo:inline` by using the computed style set associated with each original TTML Content element or anonymous span;
9. for each `br` element that is not associated with a `tts:display` style property with the value `none`, map the element to a distinct `fo:character` element having the following properties:
   - `character="&#x000A;"`
   - `suppress-at-line-break="retain"`
10. for each TTML style property attribute in some computed style set that has no counterpart in [XSL 1.1], map that attribute directly through to the relevant formatting object produced by the input TTML Content element to which the style property applies;
11. optionally, synthesize a unique `id` attribute on each resulting formatting object element that relates that element to the input element that resulted in that formatting object element;

For each resulting document instance F, if processing requires presentation on a visual medium, then apply formatting and rendering semantics consistent with that prescribed by [XSL 1.1].

> **Note:**
>
> In an XSL FO area tree produced by formatting *F* using an [XSL 1.1] formatting processor, the `page-viewport-area`, which is generated by `fo:page-sequence` element by reference to the sole generated `fo:simple-page-master` element, would correspond to the *Root Container Region* defined above in **2 Definitions**.

> **Note:**
>
> When mapping a `region` element to `fo:block-container`, it may be necessary to use a negative offset as a value for one or more of the `top`, `left`, `bottom`, and `right` XSL-FO properties in case the region extends outside of its containing block.

> **Note:**
>
> Due to the possible presence of TTML style properties or style property values in a given *Document Instance* for which there is no [XSL 1.1] counterpart, Implementors should recognize that it is the layout model of [XSL 1.1] that is being referenced by this specification, not the requirement to use a compliant [XSL 1.1] formatting processor, since such would not necessarily be sufficient to satisfy the full presentation semantics defined by this specification, and would contain a large number of features not needed to implement the presentation semantics of TTML.

> **Note:**
>
> The purpose of inserting additional, collapsible space in the block progression dimension of the `fo:block` that corresponds with the `body` element is to ensure that the before and after edges of this `fo:block` are coincident with the before and after edges of the `fo:block-container` that corresponds to the containing `region`, while simultaneously taking into account the needs to satisfy alignment in the block progression dimension. For example, this assures that the background color associated with the `body` element, if not `transparent`, will fill the containing region wholly.

9.3.5 Elaborated Example (Informative)

An example of the processing steps described above is elaborated below, starting with **Example – Sample Source Document**.

Example – Sample Source Document

```
<tt tts:extent="640px 480px" xml:lang="en"
  xmlns="http://www.w3.org/ns/ttml"
  xmlns:tts="http://www.w3.org/ns/ttml#styling">
  <head>
    <layout>
      <region xml:id="r1">
        <style tts:origin="10px 100px"/>
        <style tts:extent="620px 96px"/>
        <style tts:fontSize="40px"/>
        <style tts:fontWeight="bold"/>
        <style tts:backgroundColor="black"/>
        <style tts:color="red"/>
        <style tts:textAlign="center"/>
        <style tts:displayAlign="center"/>
      </region>
      <region xml:id="r2">
        <style tts:origin="10px 300px"/>
        <style tts:extent="620px 96px"/>
        <style tts:fontSize="40px"/>
        <style tts:fontWeight="bold"/>
        <style tts:backgroundColor="black"/>
        <style tts:color="yellow"/>
        <style tts:textAlign="center"/>
        <style tts:displayAlign="center"/>
      </region>
    </layout>
  </head>
  <body xml:id="b1">
    <div xml:id="d1" begin="0s" dur="2s">
      <p xml:id="p1" region="r1">Text 1</p>
      <p xml:id="p2" region="r2">Text 2</p>
    </div>
    <div xml:id="d2" begin="1s" dur="2s">
      <p xml:id="p3" region="r2">Text 3</p>
      <p xml:id="p4" region="r1">Text 4</p>
    </div>
  </body>
</tt>
```

In the above document, the content hierarchy consists of two divisions, each containing two paragraphs. This content is targeted (associated with) one of two non-overlapping regions that are styled identically except for their position and their foreground colors, the latter of which is inherited by and applies to the (and, in this case, anonymous) spans reparented into the regions.

The following, first intermediate document shows the synchronic state for time interval [0,1), during which time only division `d1` is temporally active, and where paragraphs `p1` and `p2` (and their ancestors) are associated with regions `r1` and `r2`, respectively.

> **Note:**
>
> The intermediate documents shown below are not valid *Document Instances*, but rather, are representations of possible internal processing states used for didactic purposes.

Example – Intermediate Document – [0s,1s)

```
<tt tts:extent="640px 480px" xml:lang="en"
  xmlns="http://www.w3.org/ns/ttml"
  xmlns:tts="http://www.w3.org/ns/ttml#styling">
  <head>
    <layout>
      <region xml:id="r1">
        <style tts:origin="10px 100px"/>
        <style tts:extent="620px 96px"/>
        <style tts:fontSize="40px"/>
        <style tts:fontWeight="bold"/>
        <style tts:backgroundColor="black"/>
        <style tts:color="red"/>
        <style tts:textAlign="center"/>
        <style tts:displayAlign="center"/>
        <body xml:id="b1-1">
          <div xml:id="d1-1">
            <p xml:id="p1">Text 1</p>
          </div>
        </body>
      </region>
      <region xml:id="r2">
        <style tts:origin="10px 300px"/>
        <style tts:extent="620px 96px"/>
        <style tts:fontSize="40px"/>
        <style tts:fontWeight="bold"/>
        <style tts:backgroundColor="black"/>
        <style tts:color="yellow"/>
        <style tts:textAlign="center"/>
        <style tts:displayAlign="center"/>
        <body xml:id="b1-2">
          <div xml:id="d1-2">
            <p xml:id="p2">Text 2</p>
          </div>
        </body>
      </region>
    </layout>
  </head>
</tt>
```

An XSL FO document instance that would yield rendering consistent with TTML, and which may be produced by performing flow processing upon the first intermediate document is illustrated below.

Example – XSL FO Document – [0s,1s)

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="m1"
      page-width="640px" page-height="480px">
      <fo:region-body/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="m1">
    <fo:flow flow-name="xsl-region-body">
      <!-- region (r1) -->
      <fo:block-container id="r1" absolute-position="absolute"
        left="10px" top="100px" width="620px" height="96px"
        background-color="black" display-align="center">
        <!-- body (b1) -->
        <fo:block id="b1-1">
          <!-- body's space (before) filler -->
          <fo:block
            space-after.optimum="96px"
            space-after.maximum="96px"
            space-after.conditionality="retain"/>
          <!-- div (d1) -->
          <fo:block id="d1-1">
            <!-- p (p1) -->
            <fo:block id="p1" text-align="center">
              <fo:inline font-size="40px" font-weight="bold"
              color="red">Text 1</fo:inline>
            </fo:block>
          </fo:block>
          <!-- body's space (after) filler -->
          <fo:block
            space-after.optimum="96px"
            space-after.maximum="96px"
            space-after.conditionality="retain"/>
        </fo:block>
      </fo:block-container>
      <!-- region (r2) -->
      <fo:block-container id="r2" absolute-position="absolute"
        left="10px" top="300px" width="620px" height="96px"
        background-color="black" display-align="center">
        <!-- body (b1) -->
        <fo:block id="b1-2">
          <!-- body's space (before) filler -->
          <fo:block
            space-after.optimum="96px"
            space-after.maximum="96px"
            space-after.conditionality="retain"/>
          <!-- div (d1) -->
          <fo:block id="d1-2">
            <!-- p (p2) -->
            <fo:block id="p2" text-align="center">
              <fo:inline font-size="40px" font-weight="bold"
              color="yellow">Text 2</fo:inline>
            </fo:block>
          </fo:block>
          <!-- body's space (after) filler -->
          <fo:block
            space-after.optimum="96px"
            space-after.maximum="96px"
            space-after.conditionality="retain"/>
        </fo:block>
      </fo:block-container>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

The following, second intermediate document shows the synchronic state for time interval [1,2), during which time both divisions `d1` and `d2` are temporally active, and where paragraphs `p1` and `p4` (and their ancestors) are associated with region `r1` and paragraphs `p2` and `p3` (and their ancestors) are associated with region `r2`.

Example – Intermediate Document – [1s,2s)

```
<tt tts:extent="640px 480px" xml:lang="en"
  xmlns="http://www.w3.org/ns/ttml"
  xmlns:tts="http://www.w3.org/ns/ttml#styling">
  <head>
    <layout>
      <region xml:id="r1">
        <style tts:origin="10px 100px"/>
        <style tts:extent="620px 96px"/>
        <style tts:fontSize="40px"/>
        <style tts:fontWeight="bold"/>
        <style tts:backgroundColor="black"/>
        <style tts:color="red"/>
        <style tts:textAlign="center"/>
        <style tts:displayAlign="center"/>
        <body xml:id="b1-1">
          <div xml:id="d1-1">
            <p xml:id="p1">Text 1</p>
          </div>
          <div xml:id="d2-1">
            <p xml:id="p4">Text 4</p>
          </div>
        </body>
      </region>
      <region xml:id="r2">
        <style tts:origin="10px 300px"/>
        <style tts:extent="620px 96px"/>
        <style tts:fontSize="40px"/>
        <style tts:fontWeight="bold"/>
        <style tts:backgroundColor="black"/>
        <style tts:color="yellow"/>
        <style tts:textAlign="center"/>
        <style tts:displayAlign="center"/>
        <body xml:id="b1-2">
          <div xml:id="d1-2">
            <p xml:id="p2">Text 2</p>
          </div>
          <div xml:id="d2-2">
            <p xml:id="p3">Text 3</p>
          </div>
        </body>
      </region>
    </layout>
  </head>
</tt>
```

The following, third intermediate document shows the synchronic state for time interval [2,3), during which time only division d2 is temporally active, and where paragraphs p4 and p3 (and their ancestors) are associated with regions r1 and r2, respectively.

Example – Intermediate Document – [2s,3s)

```
<tt tts:extent="640px 480px" xml:lang="en"
  xmlns="http://www.w3.org/ns/ttml"
  xmlns:tts="http://www.w3.org/ns/ttml#styling">
  <head>
    <layout>
      <region xml:id="r1">
        <style tts:origin="10px 100px"/>
        <style tts:extent="620px 96px"/>
        <style tts:fontSize="40px"/>
        <style tts:fontWeight="bold"/>
        <style tts:backgroundColor="black"/>
        <style tts:color="red"/>
        <style tts:textAlign="center"/>
        <style tts:displayAlign="center"/>
        <body xml:id="b1-1">
          <div xml:id="d2-1">
            <p xml:id="p4">Text 4</p>
          </div>
        </body>
      </region>
      <region xml:id="r2">
        <style tts:origin="10px 300px"/>
        <style tts:extent="620px 96px"/>
        <style tts:fontSize="40px"/>
        <style tts:fontWeight="bold"/>
        <style tts:backgroundColor="black"/>
        <style tts:color="yellow"/>
        <style tts:textAlign="center"/>
        <style tts:displayAlign="center"/>
        <body xml:id="b1-2">
          <div xml:id="d2-2">
            <p xml:id="p3">Text 3</p>
          </div>
        </body>
      </region>
    </layout>
  </head>
</tt>
```

9.4 Line Layout

If a profile that applies to a *Document Instance* requires use of the #lineBreak-uax14 feature (i.e., the value attribute for the feature is specified as use), then the recommendations defined by Line Breaking Algorithm [UAX14] apply when performing line layout on the content of the *Document Instance*.

10 Timing

This section specifies the *timing* matter of the core vocabulary catalog, where timing is to be understood as a separable layer of information that applies to content and that denotes authorial intentions about the temporal presentation of that content.

10.1 Timing Element Vocabulary

No timing related element vocabulary is defined for use in the core vocabulary catalog.

10.2 Timing Attribute Vocabulary

This section defines the following basic timing attributes for use with timed elements:

- **10.2.1 begin**
- **10.2.2 end**
- **10.2.3 dur**

In addition, this section defines the **10.2.4 timeContainer** attribute for use with timed elements that serve simultaneously as timing containers.

10.2.1 begin

The `begin` attribute is used to specify the begin point of a temporal interval associated with a timed element. If specified, the value of a `begin` attribute must adhere to a <timeExpression> specification as defined by **10.3.1 <timeExpression>**.

The begin point of a temporal interval is included in the interval; i.e., the interval is left-wise closed.

The semantics of the `begin` attribute are those defined by [SMIL 2.1], § 10.4.1, while taking into account any overriding semantics defined by this specification.

> **Note:**
>
> See [SMIL 2.1], § 10.4.1, *Begin value semantics* when no `begin` attribute is specified.

10.2.2 end

The `end` attribute is used to specify the ending point of a temporal interval associated with a timed element. If specified, the value of an `end` attribute must adhere to a <timeExpression> specification as defined by **10.3.1 <timeExpression>**.

The ending point of a temporal interval is not included in the interval; i.e., the interval is right-wise open.

The presentation effects of a non-empty active temporal interval include the frame immediately prior to the frame (or tick) equal to or immediately following the time specified by the ending point, but do not extend into this latter frame (or tick).

> **Note:**
>
> For example, if an active interval is [10s,10.33333s), and the frame rate is 30 frames per second, then the presentation effects of the interval are limited to frames 300 through 309 only (assuming that 0s corresponds with frame 0). The same holds if the active interval is specified as [300f,310f).

The semantics of the `end` attribute are those defined by [SMIL 2.1], § 10.4.1, while taking into account any overriding semantics defined by this specification.

10.2.3 dur

The `dur` attribute is used to specify the duration of a temporal interval associated with a timed element. If specified, the value of a `dur` attribute must adhere to a <timeExpression> specification as defined by **10.3.1 <timeExpression>**.

> **Note:**
>
> When the `clock-time` form of a <timeExpression> specification is used with a `dur` attribute, it is intended to be interpreted as a difference between two implied clock time expressions.

When a *Document Instance* specifies the use of the `smpte` time base and `discontinuous` marker mode, a (well-formed) `dur` attribute must not be specified on any element.

The semantics of the `dur` attribute are those defined by [SMIL 2.1], § 10.4.1, while taking into account any overriding semantics defined by this specification. In a deliberate divergence from [SMIL 2.1], § 10.4.1, the value of the `dur` attribute is permitted to be zero (0).

> **Note:**
>
> In the context of the subset of [SMIL 2.1] semantics supported by this specification, the active duration of an element that specifies both `end` and `dur` attributes is equal to the lesser of the value of the `dur` attribute and the difference between the value of the `end` attribute and the element's begin time.

10.2.4 timeContainer

The `timeContainer` attribute is used to specify a local temporal context by means of which timed child elements are temporally situated.

If specified, the value of a `timeContainer` attribute must be one of the following:

- `par`
- `seq`

If the time container semantics of an element instance is `par`, then the temporal intervals of child elements are considered to apply in parallel, i.e., simultaneously in time. Furthermore, the specification of the time interval of each child element is considered to be relative to the temporal interval of the container element instance. For the purpose of determining the [SMIL 2.1] `endsync` semantics of a `par` time container, a default value of `all` applies.

> **Note:**
>
> The use of a default value of `all` for the `endsync` behavior is distinct from [SMIL 2.1] which uses a default value of `last`.

If the time container semantics of an element instance is `seq`, then the temporal intervals of child elements are considered to apply in sequence, i.e., sequentially in time. Furthermore, the specification of the time interval of each child element is considered to be relative to the temporal interval of its sibling elements, unless it is the first child element, in which case it is considered to be relative to the temporal interval of the container element instance.

Each time container is considered to constitute an independent time base, i.e., time coordinate system.

If a `timeContainer` attribute is not specified on an element that has time container semantics, then `par` time container semantics must apply.

Time container semantics applies only to the following element types:

- `body`

- div
- p
- region
- span

The semantics of parallel and sequential time containment are those defined by [SMIL 2.1], § 10.4.2, while taking into account any overriding semantics defined by this specification.

10.3 Time Value Expressions

Timing attribute values include the use of the following expressions:

10.3.1 <timeExpression>

A <timeExpression> is used to specify a coordinate within some time base, where the applicable time base is determined by the `ttp:timeBase` parameter, and where the semantics defined by **N Time Expression Semantics** apply.

> **Note:**
>
> See **6.2.4 ttp:frameRate**, **6.2.9 ttp:subFrameRate**, **6.2.10 ttp:tickRate**, and **6.2.11 ttp:timeBase** for further information on explicit specification of frame rate, sub-frame rate, tick rate, and time base.

Syntax Representation – <timeExpression>

```
<timeExpression>
  : clock-time
  | offset-time

clock-time
  : hours ":" minutes ":" seconds ( fraction | ":" frames ( "." sub-frames )? )?

offset-time
  : time-count fraction? metric

hours
  : <digit> <digit>
  | <digit> <digit> <digit>+

minutes | seconds
  : <digit> <digit>

frames
  : <digit> <digit>
  | <digit> <digit> <digit>+

sub-frames
  : <digit>+

fraction
  : "." <digit>+

time-count
```

```
    : <digit>+

  metric
    : "h"                 // hours
    | "m"                 // minutes
    | "s"                 // seconds
    | "ms"                // milliseconds
    | "f"                 // frames
    | "t"                 // ticks
```

If a <timeExpression> is expressed in terms of a *clock-time*, then leading zeroes are used when expressing hours, minutes, seconds, and frames less than 10. Minutes are constrained to [0…59], while seconds (including any fractional part) are constrained to the closed interval [0,60], where the value 60 applies only to leap seconds. Except when `ttp:timeBase` is `clock` and `ttp:clockMode` is `local` or `utc`, use of the special value of 60 seconds to denote a leap second is undefined and should not be used.

If a <timeExpression> is expressed in terms of a *clock-time* and a *frames* term is specified, then the value of this term must be constrained to the interval [0…*F-1*], where *F* is the frame rate determined by the `ttp:frameRate` parameter as defined by **6.2.4 ttp:frameRate**. It is considered an error if a *frames* term or `f` (frames) metric is specified when the `clock` time base applies.

If a <timeExpression> is expressed in terms of a *clock-time* and a *sub-frames* term is specified, then the value of this term must be constrained to the interval [0…*S-1*], where *S* is the sub-frame rate determined by the `ttp:subFrameRate` parameter as defined by **6.2.9 ttp:subFrameRate**. It is considered an error if a *sub-frames* term is specified when the `clock` time base applies.

If the governing time base is `smpte`, then (1) the *offset-time* form is not defined and should not be used, and (2) the *fraction* seconds component in a *clock-time* form is not defined and should not be used.

10.4 Time Intervals

The semantics of time containment, durations, and intervals defined by [SMIL 2.1] apply to the interpretation of like-named timed elements and timing vocabulary defined by this specification, given the following constraints:

- The implicit duration of an anonymous span, a `br` element, a `set` element, or a `span` element whose children are exclusively text nodes, is defined as follows: if the parent time container is a parallel time container, then the implicit duration is equivalent to the `indefinite` duration value as defined by [SMIL 2.1]; if the parent time container is a sequential time container, then the implicit duration is equivalent to zero.
- The implicit duration of a `body` element, `div` element, `p` element, or `span` element whose children are not exclusively text nodes, is determined in accordance to (1) whether the element is a parallel or sequential time container, (2) the default `endsync` semantics defined above by **10.2.4 timeContainer**, and (3) the semantics of [SMIL 2.1] as applied to these time containers.
- The implicit duration of the `region` element is defined to be equivalent to the `indefinite` duration value as defined by [SMIL 2.1].
- If the governing time base is `clock`, then time expressions are considered to be equivalent to wall-clock based timing in [SMIL 2.1], where the specific semantics of **N.1 Clock Time Base** apply.
- If the governing time base is `media`, then time expressions are considered to be equivalent to offset based timing in [SMIL 2.1], where the specific semantics of **N.2 Media Time Base** apply.
- If the governing time base is `smpte`, then time expressions are considered to be equivalent to either offset based timing or event based timing in [SMIL 2.1], where the specific semantics of **N.3 SMPTE Time Base** apply.

For the purposes of performing presentation processing, the **[construct anonymous spans]** procedure is applied before resolving begin and end times of content elements.

The following example illustrates the timing semantics of anonymous spans that are children of sequential and parallel time containers. The words *Hello* and *Allo* are both contained in anonymous spans that are children of a sequential time container (the `p` element). As such, neither *Hello* nor *Allo* will be displayed since the implicit duration of the anonymous spans is 0. In contrast, the anonymous span that contains the word *Guten* and the innermost `span` that contains the word *Tag* are children of an element with parallel time container semantics (the outermost `span` element). As a result, both *Guten* and *Tag* will be displayed since the implicit duration of their respective parent anonymous span and `span` element are indefinite, resulting in the implicit duration of its `span`, `p`, `div` and `body` ancestors to also be indefinite.

Example Fragment – Timing of Anonymous Spans in a Sequential and Parallel Time Containers

```
...
<body>
  <div>
    <p timeContainer="seq">
      Hello
      <span>Guten <span>Tag</span></span>
      Allo
    </p>
  </div>
</body>
```

11 Animation

This section specifies the *animation* matter of the core vocabulary catalog, where animation is to be understood as a separable layer of information that combines timing and styling in order to denote authorial intention about (temporally) dynamic styling of content.

11.1 Animation Element Vocabulary

The following elements specify the structure and principal animation aspects of a *Document Instance*:

- **11.1.1 set**

11.1.1 set

The `set` element is used as a child element of a Content element or a `region` element in order to express a discrete change of some style parameter value that applies over some time interval.

The `set` element accepts as its children zero or more elements in the `Metadata.class` element group.

XML Representation – Element Information Item: set

```
<set
  begin = <timeExpression>
  dur = <timeExpression>
  end = <timeExpression>
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {a single attribute in TT Style namespace}
  {any attribute not in default or any TT namespace}>
  Content: Metadata.class*
</set>
```

> **Note:**
>
> The use of multiple `set` element children may be used to effect fade and position transitions.

An example of using the `set` element to animate content styling is illustrated below:

Example Fragment – Content Style Animation

```
...
<p dur="5s" tts:color="yellow">
<set begin="1s" dur="1s" tts:color="red"/>
<set begin="2s" dur="1s" tts:color="green"/>
<set begin="3s" dur="1s" tts:color="red"/>
Text with Flashing Colors!
</p>
...
```

> **Note:**
>
> In the above example, the foreground color of the content "Text with Flashing Colors" is animated from yellow, to red, to green, to red, then back to yellow over a 5 second period.

An example of using the `set` element to animate region styling is illustrated below:

Example Fragment – Region Style Animation

```
<tt xml:lang="" xmlns="http://www.w3.org/ns/ttml"
    xmlns:ttp="http://www.w3.org/ns/ttml#parameter"
    xmlns:tts="http://www.w3.org/ns/ttml#styling"
    ttp:cellResolution="40 16">
    <head>
      <layout>
        <region xml:id="r1" timeContainer="seq">
          <set dur="10s" tts:origin=" 8c 14c"/>
          <set dur="2s"  tts:origin=" 2c  2c"/>
          <set dur="3s"  tts:origin=" 8c 14c"/>
          <set dur="2s"  tts:origin="14c  4c"/>
          <set dur="10s" tts:origin=" 8c 14c"/>
          <style tts:extent="24c 2c"/>
        </region>
      </layout>
    </head>
    <body region="r1">...</body>
</tt>
```

> **Note:**
>
> In the above example, the *Root Container Region* is divided into a cell grid of 40 columns and 16 rows. A region, `r1`, with dimensions of 24 columns and 2 rows is then positioned within the *Root Container Region*, with its position varying over time in order to create an effect of moving the region, which may be desirable so as to avoid obscuring characters in an underlying video with captions.

> **Note:**
>
> The semantics of the `set` element are based upon that defined by [SVG 1.1], § 6.2.13.

11.2 Animation Attribute Vocabulary

No animation related attribute vocabulary is defined for use in the core vocabulary catalog.

12 Metadata

This section specifies the *metadata* matter of the core vocabulary catalog, where metadata is to be understood as a separable layer of information that applies to parameters, content, style, layout, timing, and even metadata itself, where the information represented by metadata takes one of two forms: (1) metadata defined by this specification for standardized use in a *Document Instance*, and (2) arbitrary metadata defined outside of the scope of this specification, whose use and semantics depend entirely upon an application's use of TTML Content.

12.1 Metadata Element Vocabulary

The **12.1.1 metadata** element serves as a generic container element for grouping metadata information.

In addition, the following elements, all defined in the TT Metadata Namespace, provide standard representations for metadata that is expected to be commonly used in a *Document Instances*:

- **12.1.2 ttm:title**
- **12.1.3 ttm:desc**
- **12.1.4 ttm:copyright**
- **12.1.5 ttm:agent**
- **12.1.6 ttm:name**
- **12.1.7 ttm:actor**

12.1.1 metadata

The `metadata` element functions as a generic container for metadata information.

Metadata information may be expressed with a `metadata` element by specifying (1) one or more metadata attributes on the `metadata` element, (2) one or more metadata child elements in the `metadata` element, or (3) a combination of metadata attributes and metadata child elements. Both types of metadata information are referred to in this document as *metadata items*.

XML Representation – Element Information Item: metadata

```
<metadata
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute in TT Metadata namespace}
  {any attribute not in default or any TT namespace}>
  Content: ({any element in TT Metadata namespace}|{any element not in any TT namespace})*
</metadata>
```

> **Note:**
>
> The meaning of a specific metadata item must be evaluated in the context where it appears. The core vocabulary catalog permits an arbitrary number of `metadata` element children on any Content element type. See specific element vocabulary definitions for any constraints that apply to such usage.

The use of document metadata is illustrated by the following example.

Example Fragment – Document Metadata

```
...
<head>
  <metadata xmlns:ttm="http://www.w3.org/ns/ttml#metadata">
    <ttm:title>Document Metadata Example</ttm:title>
    <ttm:desc>This document employs document metadata.</ttm:desc>
  </metadata>
</head>
...
```

The use of element metadata is illustrated by the following example.

Example Fragment – Element Metadata

```
...
<div>
  <metadata xmlns:ttm="http://www.w3.org/ns/ttml#metadata">
    <ttm:title>Chapter 6 – Sherlock Holmes Gives a Demonstration</ttm:title>
    <ttm:desc>Holmes shows Watson how the murderer entered the window.</ttm:desc>
  </metadata>
</div>
...
```

The use of metadata attribute items is illustrated by the following example.

Example Fragment – Foreign Metadata Attribute Items

```
...
<div xmlns:ext="http://example.org/ttml#metadata">
  <metadata ext:ednote="remove this division prior to publishing"/>
</div>
...
```

> **Note:**
>
> In the above example, a global attribute from a foreign (external) namespace is used to express a metadata attribute that applies semantically to the containing `div` element. Note that the attribute may also be expressed directly on the `div` element; however, in this case the author wishes to segregate certain metadata attributes by expressing them indirectly on `metadata` elements.

The use of foreign element metadata is illustrated by the following example.

Example Fragment – Foreign Element Metadata

```
...
<metadata
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dc:title>Foreign Element Metadata Example</dc:title>
  <dc:description>Express metadata using elements in foreign namespace.</dc:description>
  <dc:format xsi:type="dcterms:IMT">application/ttml+xml</dc:format>
</metadata>
...
```

> **Note:**
>
> In the above example, a number of elements defined by the Dublin Core metadata vocabulary are used to express document level metadata.

12.1.2 ttm:title

The `ttm:title` element is used to express a human-readable title of a specific element instance.

XML Representation – Element Information Item: ttm:title

```
<ttm:title
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute not in default or any TT namespace}>
  Content: #PCDATA
</ttm:title>
```

> **Note:**
>
> No specific use of the `ttm:title` element is defined by this specification.

Examples of the `ttm:title` element are shown above in **Example Fragment – Document Metadata** and **Example Fragment – Element Metadata**.

12.1.3 ttm:desc

The `ttm:desc` element is used to express a human-readable description of a specific element instance.

XML Representation – Element Information Item: ttm:desc

```
<ttm:desc
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute not in default or any TT namespace}>
  Content: #PCDATA
</ttm:desc>
```

> **Note:**
>
> No specific use of the `ttm:desc` element is defined by this specification.

Examples of the `ttm:desc` element are shown above in **Example Fragment – Document Metadata** and **Example Fragment – Element Metadata**.

12.1.4 ttm:copyright

The `ttm:copyright` element is used to express a human-readable copyright that applies to some scoping level.

A copyright statement that applies to a document as a whole should appear as a child of the `head` element.

XML Representation – Element Information Item: ttm:copyright

```
<ttm:copyright
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute not in default or any TT namespace}>
  Content: #PCDATA
</ttm:copyright>
```

> **Note:**
>
> No specific use of the `ttm:copyright` element is defined by this specification.

12.1.5 ttm:agent

The `ttm:agent` element is used to define an agent for the purpose of associating content information with an agent who is involved in the production or expression of that content.

The `ttm:agent` element accepts as its children zero or more `ttm:name` elements followed by zero or one `ttm:actor` element.

At least one `ttm:name` element child should be specified that expresses a name for the agent, whether it be the name of a person, character, group, or organization.

XML Representation – Element Information Item: ttm:agent

```
<ttm:agent
  type = (person|character|group|organization|other)
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute not in default or any TT namespace}>
  Content: ttm:name*, ttm:actor?
</ttm:agent>
```

A `type` attribute must be specified on each `ttm:agent` element, and, if specified, must have one of the following values:

- `person`
- `character`

- `group`
- `organization`
- `other`

If the value of the `type` attribute is `character`, then the `ttm:agent` element instance should specify a `ttm:actor` child that specifies the agent that plays the role of the actor.

A `ttm:agent` metadata item is considered to be significant only when specified as a child of the `head` element or as a child of a `metadata` element child of the `head` element.

> **Note:**
>
> A `ttm:agent` element instance is typically referenced using a `ttm:agent` attribute on a Content element type.

> **Note:**
>
> If a character agent is played by multiple actors, then multiple character agents may be specified (and referenced) wherein different definitions of the character specify different actors.

The use of agent metadata is illustrated by the following example.

Example Fragment – Agent Metadata

```
<tt xml:lang="en" xmlns="http://www.w3.org/ns/ttml" xmlns:ttm="http://www.w3.org/ns/ttml#metadata">
  <head>
    <ttm:agent xml:id="connery" type="person">
      <ttm:name type="family">Connery</ttm:name>
      <ttm:name type="given">Thomas Sean</ttm:name>
      <ttm:name type="alias">Sean</ttm:name>
      <ttm:name type="full">Sir Thomas Sean Connery</ttm:name>
    </ttm:agent>
    <ttm:agent xml:id="bond" type="character">
      <ttm:name type="family">Bond</ttm:name>
      <ttm:name type="given">James</ttm:name>
      <ttm:name type="alias">007</ttm:name>
      <ttm:actor agent="connery"/>
    </ttm:agent>
  </head>
  <body>
    <div>
      ...
      <p ttm:agent="bond">I travel, a sort of licensed troubleshooter.</p>
      ...
    </div>
  </body>
</tt>
```

> **Note:**
>
> In the above example, two agents, a real (person) agent, Sean Connery, and a fictitious (character) agent, James Bond, are defined, where the latter is linked to the former by means of the a `ttm:actor` element. A reference is then made from content (the `p` element) to the character agent associated with (responsible for producing) that content. Note that in this example the `ttm:agent` metadata items are specified as immediate children of the document's `head` element rather than being placed in a container `metadata` element.

12.1.6 ttm:name

The `ttm:name` element is used to specify a name of a person, character, group, or organization.

XML Representation – Element Information Item: ttm:name

```
<ttm:name
  type = (full|family|given|alias|other)
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute not in default or any TT namespace}>
  Content: #PCDATA
</ttm:name>
```

A `type` attribute must be specified on each `ttm:name` element, and, if specified, must have one of the following values:

- `full`
- `family`
- `given`
- `alias`
- `other`

The relationship between the type of a name and the syntactic expression of the name is not defined by this specification.

Two examples of the `ttm:name` element are shown above in **Example Fragment – Agent Metadata**.

12.1.7 ttm:actor

The `ttm:actor` element is used to link the definition of a (role-based) character agent with another agent that portrays the character.

XML Representation – Element Information Item: ttm:actor

```
<ttm:actor
  agent = IDREF
  xml:id = ID
  xml:lang = string
  xml:space = (default|preserve)
  {any attribute not in default or any TT namespace}>
  Content: EMPTY
</ttm:actor>
```

The `agent` attribute of a `ttm:actor` element must reference a *significant* `ttm:agent` element that denotes the agent acting the part of a character.

An `ttm:actor` element referencing its parent `ttm:agent` element should be considered an error.

> **Note:**
>
> The semantics of an `ttm:actor` element referencing its parent `ttm:agent` element are not defined. While discouraged, the relationship is syntactically permitted. It is therefore recommended that implementations be prepared to process this syntax, e.g. by avoiding infinite loops etc.

> **Note:**
>
> In the case of an actor playing themselves, like Steve Coogan in The Trip playing Steve Coogan, the expected practice is to use two `ttm:agent` elements. The first element (corresponding to Steve Coogan the person) has `type` attribute equal to `person`, while the second element (corresponding to Steve Coogan the character) has (i) a `type` attribute equal to `character` and (ii) a child `ttm:actor` element that references the first `ttm:agent` element.

An example of the `ttm:actor` element is shown above in **Example Fragment – Agent Metadata**.

12.2 Metadata Attribute Vocabulary

This section specifies the following attributes in the TT Metadata Namespace for use with the `metadata` element and with certain Content element types:

- **12.2.1 ttm:agent**
- **12.2.2 ttm:role**

> **Note:**
>
> Only certain metadata item attributes may be used with Content elements. See the definitions of Content elements to determine permissible usage.

12.2.1 ttm:agent

The `ttm:agent` attribute takes an `IDREFS` value, and is used with certain Content elements to designate the agents that perform or are involved in the performance of the content.

If specified, a `ttm:agent` attribute must reference *significant* `ttm:agent` element instances.

The same IDREF, *ID*, should not appear more than once in the value of a `ttm:agent` attribute.

> **Note:**
>
> This constraint is intended to discourage the use of redundant agent references.

An example of the `ttm:agent` attribute is shown above in **Example Fragment – Agent Metadata**.

12.2.2 ttm:role

The `ttm:role` attribute may be used by a content author to express the roles, functions, or characteristics of some Content element that is so labeled.

If specified, the value of this attribute must adhere to the following syntax, where the syntactic element *S* must adhere to production [3] `S` as defined by [XML 1.0] § 2.3:

Syntax Representation – ttm:role

```
ttm:role
  role ( S role )*

role
  : "action"
  | "caption"
  | "description"
  | "dialog"
  | "expletive"
  | "kinesic"
  | "lyrics"
  | "music"
  | "narration"
  | "quality"
  | "sound"
  | "source"
  | "suppressed"
  | "reproduction"
  | "thought"
  | "title"
  | "transcription"
  | extension-role

extension-role
  : "x-" token-char+

token-char
  : { XML NameChar }    // XML 1.1 Production [4a]
```

The same `role` token, *R*, should not appear more than once in the value of a `ttm:role` attribute.

> **Note:**
>
> This constraint is intended to discourage the use of redundant role tokens.

> **Note:**
>
> All values of `ttm:role` that do not start with the prefix `x-` are reserved for future standardization.

> **Note:**
>
> If using a custom `x-` prefixed form of `ttm:role`, it is recommended that an organization unique infix be used as well in order to prevent collisions. For example, `x-example-org-custom-role`. Furthermore, a registry for role values is available at http://www.w3.org/wiki/TTML/RoleRegistry in order to promote interoperability and collision avoidance.

A Reduced XML Infoset

This appendix is normative.

For the purposes of this specification, a *Reduced XML Infoset* is an XML Information Set [XML InfoSet] that consists of only the following information items and information item properties:

A.1 Document Information Item

- `[document element]`

A.2 Element Information Item

- `[namespace URI]`
- `[local name]`
- `[children]`
- `[attributes]`

Child information items `[children]` are reduced to only element and character information items.

A.3 Attribute Information Item

- `[namespace URI]`
- `[local name]`
- `[normalized value]`

A.4 Character Information Item

- `[character code]`

Contiguous character information items are not required to be represented distinctly, but may be aggregated (chunked) into a sequence of character codes (i.e., a character string).

B Schemas

This appendix is normative.

This appendix specifies the following schemas for use with *Document Instances*:

- Relax NG, Compact Syntax (RNC) Schema
- XML Schema Definition (XSD) Schema

In any case where a schema specified by this appendix differs from the normative definitions of document type, element type, or attribute type as defined by the body of this specification, then the body of this specification takes precedence.

B.1 Relax NG Compact (RNC) Schema

A Relax NG Compact Syntax (RNC) [RELAX NG] based schema for TTML Content is available at ZIP Archive. This schema does not normatively define the validity of Timed Text Markup Language content as defined by this specification. In particular, the formal validity of TTML Content is defined by **3.1 Content Conformance**.

B.2 XML Schema Definition (XSD) Schema

A W3C XML Schema Definition (XSD) [XML Schema Part 1] based schema for TTML Content is available at ZIP Archive. This schema does not normatively define the validity of Timed Text Markup Language content as defined by this specification. In particular, the formal validity of TTML Content is defined by **3.1 Content Conformance**.

C Media Type Registration

This appendix is normative.

This appendix registers a new media type, "application/ttml+xml" in conformance with BCP 13 and W3CRegMedia. The information in this appendix is being submitted to the Internet Engineering Steering Group (IESG) for review, approval, and registration with the Internet Assigned Numbers Authority (IANA).

Type name:
application

Subtype name:
ttml+xml

Required parameters:
None.

Optional parameters:

charset
If specified, the `charset` parameter must match the XML encoding declaration, or if absent, the actual encoding.

profile
The document profile of a TTMLDocument Instance may be specified using an optional `profile` parameter, which, if specified, the value of which must adhere to the syntax and semantics of `ttp:profile` parameter defined by Section **6.2.8 ttp:profile** of the published specification.

Encoding considerations:
binary

Security considerations:
As with other XML types and as noted in [XML Media Types] (http://www.rfc-editor.org/rfc/rfc3023.txt), Section 10, repeated expansion of maliciously constructed XML entities can be used to consume large amounts of memory, which may cause XML processors in constrained environments to fail.

In addition, because of the extensibility features for TTML and of XML in general, it is possible that "application/ttml+xml" may describe content that has security implications beyond those described here. However, TTML does not provide for any sort of active or executable content, and if the processor follows only the normative semantics of the published specification, this content will be outside TTML namespaces and may be ignored. Only in the case where the processor recognizes and processes the additional content, or where further processing of that content is dispatched to other processors, would security issues potentially arise. And in that case, they would fall outside the domain of this registration document.

Although not prohibited, there are no expectations that XML signatures or encryption would normally be employed.

Interoperability considerations:
The published specification describes processing semantics that dictate behavior that must be followed when dealing with, among other things, unrecognized elements and attributes, both in TTML namespaces and in other namespaces.

Because TTML is extensible, conformant "application/ttml+xml" processors may expect (and enforce) that content received is well-formed XML, but it cannot be guaranteed that the content is valid to a particular DTD or Schema or that the processor will recognize all of the elements and attributes in the document.

Published specification:

This media type registration is extracted from Appendix C Media Type Registration of the Timed Text Markup Language (TTML) 1.0 specification: Timed Text Markup Language 1 (TTML1).

Applications that use this media type:

TTML is used in the television industry for the purpose of authoring, transcoding and exchanging timed text information and for delivering captions, subtitles, and other metadata for television material repurposed for the Web or, more generally, the Internet.

There is partial and full support of TTML in components used by several Web browsers plugins, and in a number of caption authoring tools.

Additional information:

Magic number(s):

File extension(s):

.ttml

Macintosh file type code(s):

"TTML"

Fragment identifiers:

For documents labeled as application/ttml+xml, the fragment identifier notation is intended to be used with xml:id attributes, as described in section 7.2.1 of the Timed Text Markup Language 1 (TTML1) specification.

Person & email address to contact for further information:

Timed Text Working Group (public-tt@w3.org)

Intended usage:

COMMON

Restrictions on usage:

None

Author:

The published specification is a work product of the World Wide Web Consortium's Timed Text (TT) Working Group.

Change controller:

The W3C has change control over this specification.

D Features

This appendix is normative.

This appendix specifies (1) a set of feature designations, each of which labels one or more syntactic and/or semantic features defined by this specification, and (2) for each designated feature, whether the feature is mandatory or optional for a transformation or presentation processor.

> **Note:**
>
> A TTML processor is said to *implement the transformation semantics* or *implement the presentation semantics* of feature designation *F* if it satisfies the requirements of this appendix with respect to the definition of feature designation *F* as pertains to transformation or presentation processing, respectively.

D.1 Feature Designations

A feature designation is expressed as a string that adheres to the following form:

```
feature-designation
  : feature-namespace designation

feature-namespace
  : TT Feature Namespace                    // http://www.w3.org/ns/ttml/feature/

designation
  : "#" token-char+

token-char
  : { XML NameChar }                        // XML 1.1 Production [4a]
```

All values of `feature-designation` not defined by this specification are reserved for future standardization.

The following sub-sections define all feature designations, expressed as relative URIs (fragment identifiers) with respect to the TT Feature Namespace base URI.

D.1.1 #animation

A TTML transformation processor supports the `#animation` feature if it recognizes and is capable of transforming the following vocabulary defined by **11 Animation**:

- `set`

A TTML presentation processor supports the `#animation` feature if it implements presentation semantic support for the same vocabulary enumerated above.

D.1.2 #backgroundColor

A TTML transformation processor supports the `#backgroundColor` feature if it recognizes and is capable of transforming the `tts:backgroundColor` attribute.

A TTML presentation processor supports the `#backgroundColor` feature if it (1) implements presentation semantic support for the `tts:backgroundColor` attribute and (2) is capable of displaying or generating an output display signal that distinguishes between at least sixteen (16) values of color, including all primary and secondary colors of the SRGB color space.

> **Note:**
>
> Support for this feature is intended to imply support for the following features: `#backgroundColor-block`, `#backgroundColor-region`, and `#backgroundColor-inline`.

D.1.3 #backgroundColor-block

A TTML transformation processor supports the `#backgroundColor-block` feature if it recognizes and is capable of transforming all defined values of the `tts:backgroundColor` attribute when applied to a content element that would generate a block area during presentation processing.

A TTML presentation processor supports the `#backgroundColor-block` feature if it (1) implements presentation semantic support for the `tts:backgroundColor` attribute when applied to a content element that generates a block area and (2) is capable of displaying or generating an output display signal that distinguishes between at least sixteen (16) values of color, including all primary and secondary colors of the SRGB color space.

D.1.4 #backgroundColor-inline

A TTML transformation processor supports the `#backgroundColor-inline` feature if it recognizes and is capable of transforming all defined values of the `tts:backgroundColor` attribute when applied to a content element that would generate an inline area during presentation processing.

A TTML presentation processor supports the `#backgroundColor-inline` feature if it (1) implements presentation semantic support for the `tts:backgroundColor` attribute when applied to a content element that generates an inline area and (2) is capable of displaying or generating an output display signal that distinguishes between at least sixteen (16) values of color, including all primary and secondary colors of the SRGB color space.

D.1.5 #backgroundColor-region

A TTML transformation processor supports the `#backgroundColor-region` feature if it recognizes and is capable of transforming all defined values of the `tts:backgroundColor` attribute when applied to a `region` element.

A TTML presentation processor supports the `#backgroundColor-region` feature if it (1) implements presentation semantic support for the `tts:backgroundColor` attribute when applied to a `region` element and (2) is capable of displaying or generating an output display signal that distinguishes between at least sixteen (16) values of color, including all primary and secondary colors of the SRGB color space .

D.1.6 #bidi

A TTML processor supports the `#bidi` feature if it supports the following features:

- `#direction`
- `#unicodeBidi`
- `#writingMode-horizontal`

D.1.7 #cellResolution

A TTML transformation processor supports the `#cellResolution` feature if it recognizes and is capable of transforming the `ttp:cellResolution` attribute.

A TTML presentation processor supports the `#cellResolution` feature if it implements presentation semantic support for the `ttp:cellResolution` attribute.

D.1.8 #clockMode

A TTML transformation processor supports the `#clockMode` feature if it recognizes and is capable of transforming the `ttp:clockMode` attribute.

A TTML presentation processor supports the `#clockMode` feature if it implements presentation semantic support for the `ttp:clockMode` attribute.

D.1.9 #clockMode-gps

A TTML transformation processor supports the `#clockMode-gps` feature if it recognizes and is capable of transforming the `gps` value of the `ttp:clockMode` attribute.

A TTML presentation processor supports the `#clockMode-gps` feature if it implements presentation semantic support for the `gps` value of the `ttp:clockMode` attribute.

D.1.10 #clockMode-local

A TTML transformation processor supports the `#clockMode-local` feature if it recognizes and is capable of transforming the `local` value of the `ttp:clockMode` attribute.

A TTML presentation processor supports the `#clockMode-local` feature if it implements presentation semantic support for the `local` value of the `ttp:clockMode` attribute.

D.1.11 #clockMode-utc

A TTML transformation processor supports the `#clockMode-utc` feature if it recognizes and is capable of transforming the `utc` value of the `ttp:clockMode` attribute.

A TTML presentation processor supports the `#clockMode-utc` feature if it implements presentation semantic support for the `utc` value of the `ttp:clockMode` attribute.

D.1.12 #color

A TTML transformation processor supports the `#color` feature if it recognizes and is capable of transforming the `tts:color` attribute.

A TTML presentation processor supports the `#color` feature if it (1) implements presentation semantic support for the `tts:color` attribute and (2) is capable of displaying or generating an output display signal that distinguishes between at least sixteen (16) values of color, including all primary and secondary colors of the SRGB color space.

D.1.13 #content

A TTML transformation processor supports the `#content` feature if it recognizes and is capable of transforming the following vocabulary defined by **7 Content**:

- `body`
- `div`
- `p`
- `span`
- `br`

A TTML presentation processor supports the `#content` feature if it implements presentation semantic support for the same vocabulary enumerated above.

D.1.14 #core

A TTML transformation processor supports the `#core` feature if it recognizes and is capable of transforming the following core attributes vocabulary defined by **7 Content**:

- @`xml:id`
- @`xml:lang`
- @`xml:space`

A TTML presentation processor supports the `#core` feature if it implements presentation semantic support for the same vocabulary enumerated above.

D.1.15 #direction

A TTML transformation processor supports the `#direction` feature if it recognizes and is capable of transforming all defined values of the `tts:direction` attribute.

A TTML presentation processor supports the `#direction` feature if it implements presentation semantic support for all defined values of the `tts:direction` attribute.

D.1.16 #display

A TTML transformation processor supports the `#display` feature if it recognizes and is capable of transforming all defined values of the `tts:display` attribute.

A TTML presentation processor supports the `#display` feature if it implements presentation semantic support for all defined values of the `tts:display` attribute.

D.1.17 #display-block

A TTML transformation processor supports the `#display-block` feature if it recognizes and is capable of transforming all defined values of the `tts:display` attribute when applied to a content element that would generate a block area during presentation processing.

A TTML presentation processor supports the `#display-block` feature if it implements presentation semantic support for all defined values of the `tts:display` attribute when applied to a content element that generates a block area.

D.1.18 #display-inline

A TTML transformation processor supports the `#display-inline` feature if it recognizes and is capable of transforming all defined values of the `tts:display` attribute when applied to a content element that would generate an inline area during presentation processing.

A TTML presentation processor supports the `#display-inline` feature if it implements presentation semantic support for all defined values of the `tts:display` attribute when applied to a content element that generates an inline area.

D.1.19 #display-region

A TTML transformation processor supports the `#display-region` feature if it recognizes and is capable of transforming all defined values of the `tts:display` attribute when applied to a `region` element.

A TTML presentation processor supports the `#display-region` feature if it implements presentation semantic support for all defined values of the `tts:display` attribute when applied to a `region` element.

D.1.20 #displayAlign

A TTML transformation processor supports the `#displayAlign` feature if it recognizes and is capable of transforming all defined values of the `tts:displayAlign` attribute.

A TTML presentation processor supports the `#displayAlign` feature if it implements presentation semantic support for all defined values of the tts:displayAlign attribute.

D.1.21 #dropMode

A TTML transformation processor supports the #dropMode feature if it recognizes and is capable of transforming the ttp:dropMode attribute.

A TTML presentation processor supports the #dropMode feature if it implements presentation semantic support for the ttp:dropMode attribute.

D.1.22 #dropMode-dropNTSC

A TTML transformation processor supports the #dropMode-dropNTSC feature if it recognizes and is capable of transforming the dropNTSC value of the ttp:dropMode attribute.

A TTML presentation processor supports the #dropMode-dropNTSC feature if it implements presentation semantic support for the dropNTSC value of the ttp:dropMode attribute.

D.1.23 #dropMode-dropPAL

A TTML transformation processor supports the #dropMode-dropPAL feature if it recognizes and is capable of transforming the dropPAL value of the ttp:dropMode attribute.

A TTML presentation processor supports the #dropMode-dropPAL feature if it implements presentation semantic support for the dropPAL value of the ttp:dropMode attribute.

D.1.24 #dropMode-nonDrop

A TTML transformation processor supports the #dropMode-nonDrop feature if it recognizes and is capable of transforming the nonDrop value of the ttp:dropMode attribute.

A TTML presentation processor supports the #dropMode-nonDrop feature if it implements presentation semantic support for the nonDrop value of the ttp:dropMode attribute.

D.1.25 #extent

A TTML transformation processor supports the #extent feature if it recognizes and is capable of transforming the tts:extent attribute.

A TTML presentation processor supports the #extent feature if it implements presentation semantic support for the tts:extent attribute.

D.1.26 #extent-region

A TTML transformation processor supports the #extent-region feature if it recognizes and is capable of transforming the tts:extent attribute when applied to a region element.

A TTML presentation processor supports the #extent-region feature if it implements presentation semantic support for the tts:extent attribute when applied to a region element.

D.1.27 #extent-root

A TTML transformation processor supports the `#extent-root` feature if it recognizes and is capable of transforming the `tts:extent` attribute when applied to the `tt` element.

A TTML presentation processor supports the `#extent-root` feature if it implements presentation semantic support for the `tts:extent` attribute when applied to a `tt` element.

D.1.28 #fontFamily

A TTML transformation processor supports the `#fontFamily` feature if it recognizes and is capable of transforming the `tts:fontFamily` attribute.

A TTML presentation processor supports the `#fontFamily` feature if it implements presentation semantic support for the `tts:fontFamily` attribute.

D.1.29 #fontFamily-generic

A TTML transformation processor supports the `#fontFamily-generic` feature if it recognizes and is capable of transforming <genericFamilyName> values when used with the `tts:fontFamily` attribute.

A TTML presentation processor supports the `#fontFamily-generic` feature if it implements presentation semantic support for <genericFamilyName> values when used with the `tts:fontFamily` attribute.

D.1.30 #fontFamily-non-generic

A TTML transformation processor supports the `#fontFamily-non-generic` feature if it recognizes and is capable of transforming <familyName> values when used with the `tts:fontFamily` attribute.

A TTML presentation processor supports the `#fontFamily-non-generic` feature if it implements presentation semantic support for <familyName> values when used with the `tts:fontFamily` attribute.

D.1.31 #fontSize

A TTML transformation processor supports the `#fontSize` feature if it recognizes and is capable of transforming the `tts:fontSize` attribute.

A TTML presentation processor supports the `#fontSize` feature if it implements presentation semantic support for the `tts:fontSize` attribute.

D.1.32 #fontSize-anamorphic

A TTML transformation processor supports the `#fontSize-anamorphic` feature if it recognizes and is capable of transforming values of the `tts:fontSize` attribute that consist of two <length> specifications.

A TTML presentation processor supports the `#fontSize-anamorphic` feature if it implements presentation semantic support for defined values of the `tts:fontSize` attribute that consist of two <length> specifications.

D.1.33 #fontSize-isomorphic

A TTML transformation processor supports the `#fontSize-isomorphic` feature if it recognizes and is capable of transforming values of the tts:fontSize attribute that consist of a single <length> specification.

A TTML presentation processor supports the #fontSize-isomorphic feature if it implements presentation semantic support for defined values of the tts:fontSize attribute that consist of a single <length> specification.

D.1.34 #fontStyle

A TTML transformation processor supports the #fontStyle feature if it recognizes and is capable of transforming all defined values of the tts:fontStyle attribute.

A TTML presentation processor supports the #fontStyle feature if it implements presentation semantic support for all defined values of the tts:fontStyle attribute.

D.1.35 #fontStyle-italic

A TTML transformation processor supports the #fontStyle-italic feature if it recognizes and is capable of transforming the italic value of the tts:fontStyle attribute.

A TTML presentation processor supports the #fontStyle-italic feature if it implements presentation semantic support for the italic of the tts:fontStyle attribute.

D.1.36 #fontStyle-oblique

A TTML transformation processor supports the #fontStyle-oblique feature if it recognizes and is capable of transforming the oblique value of the tts:fontStyle attribute.

A TTML presentation processor supports the #fontStyle-oblique feature if it implements presentation semantic support for the oblique of the tts:fontStyle attribute.

D.1.37 #fontWeight

A TTML transformation processor supports the #fontWeight feature if it recognizes and is capable of transforming all defined values of the tts:fontWeight attribute.

A TTML presentation processor supports the #fontWeight feature if it implements presentation semantic support for all defined values of the tts:fontWeight attribute.

D.1.38 #fontWeight-bold

A TTML transformation processor supports the #fontWeight-bold feature if it recognizes and is capable of transforming bold value of the tts:fontWeight attribute.

A TTML presentation processor supports the #fontWeight-bold feature if it implements presentation semantic support for the bold of the tts:fontWeight attribute.

D.1.39 #frameRate

A TTML transformation processor supports the #frameRate feature if it recognizes and is capable of transforming the ttp:frameRate attribute.

A TTML presentation processor supports the #frameRate feature if it implements presentation semantic support for the ttp:frameRate attribute.

D.1.40 #frameRateMultiplier

A TTML transformation processor supports the #frameRateMultiplier feature if it recognizes and is capable of transforming the ttp:frameRateMultiplier attribute.

A TTML presentation processor supports the #frameRateMultiplier feature if it implements presentation semantic support for the ttp:frameRateMultiplier attribute.

D.1.41 #layout

A TTML transformation processor supports the #layout feature if it (1) recognizes and is capable of transforming the following vocabulary defined by **9 Layout**:

- layout
- region
- @region

and (2) supports the following attributes when applied to the region element:

- tts:extent
- tts:origin

A TTML presentation processor supports the #layout feature if it implements presentation semantic support for the same vocabulary and features enumerated above.

D.1.42 #length

A TTML transformation processor supports the #length feature if it recognizes and is capable of transforming all defined values of the <length> style value expression.

A TTML presentation processor supports the #length feature if it implements presentation semantic support for all defined values of the <length> style value expression.

> **Note:**
>
> Support for #length is intended to imply support for the following features: #length-integer, #length-real, #length-positive, #length-negative, #length-cell, #length-em, #length-percentage, and #length-pixel.

D.1.43 #length-cell

A TTML transformation processor supports the #length-cell feature if it recognizes and is capable of transforming scalar values of the <length> style value expression that use c (cell) units.

A TTML presentation processor supports the #length-cell feature if it implements presentation semantic support for scalar values of the <length> style value expression that use c (cell) units.

> **Note:**
>
> Support for `#length-cell` does not, by itself, imply support for `#length-integer`, `#length-real`, `#length-positive`, or `#length-negative` features.

D.1.44 #length-em

A TTML transformation processor supports the `#length-em` feature if it recognizes and is capable of transforming scalar values of the <length> style value expression that use `em` (EM) units.

A TTML presentation processor supports the `#length-em` feature if it implements presentation semantic support for scalar values of the <length> style value expression that use `em` (EM) units.

> **Note:**
>
> Support for `#length-em` does not, by itself, imply support for `#length-integer`, `#length-real`, `#length-positive`, or `#length-negative` features.

D.1.45 #length-integer

A TTML transformation processor supports the `#length-integer` feature if it recognizes and is capable of transforming integer values of the <length> style value expression.

A TTML presentation processor supports the `#length-integer` feature if it implements presentation semantic support for integer values of the <length> style value expression.

> **Note:**
>
> Support for `#length-integer` does not, by itself, imply support for `#length-positive` or `#length-negative` features.

D.1.46 #length-negative

A TTML transformation processor supports the `#length-negative` feature if it recognizes and is capable of transforming negative values of the <length> style value expression.

A TTML presentation processor supports the `#length-negative` feature if it implements presentation semantic support for negative values of the <length> style value expression.

> **Note:**
>
> Support for `#length-negative` does not, by itself, imply support for `#length-integer` or `#length-real` features.

D.1.47 #length-percentage

A TTML transformation processor supports the `#length-percentage` feature if it recognizes and is capable of transforming percentage values of the <length> style value expression.

A TTML presentation processor supports the `#length-percentage` feature if it implements presentation semantic support for percentage values of the <length> style value expression.

> **Note:**
>
> Support for `#length-percentage` does not, by itself, imply support for `#length-integer`, `#length-real`, `#length-positive`, or `#length-negative` features.

D.1.48 #length-pixel

A TTML transformation processor supports the `#length-pixel` feature if it recognizes and is capable of transforming scalar values of the <length> style value expression that use `px` (pixel) units.

A TTML presentation processor supports the `#length-pixel` feature if it implements presentation semantic support for scalar values of the <length> style value expression that use `px` (pixel) units.

> **Note:**
>
> Support for `#length-pixel` does not, by itself, imply support for `#length-integer`, `#length-real`, `#length-positive`, or `#length-negative` features.

D.1.49 #length-positive

A TTML transformation processor supports the `#length-positive` feature if it recognizes and is capable of transforming positive values of the <length> style value expression.

A TTML presentation processor supports the `#length-positive` feature if it implements presentation semantic support for positive values of the <length> style value expression.

> **Note:**
>
> Support for `#length-positive` is intended to imply support for zero valued <length> style value expressions.

> **Note:**
>
> Support for `#length-positive` does not, by itself, imply support for `#length-integer` or `#length-real` features.

D.1.50 #length-real

A TTML transformation processor supports the `#length-real` feature if it recognizes and is capable of transforming real values of the <length> style value expression.

A TTML presentation processor supports the `#length-real` feature if it implements presentation semantic support for real values of the <length> style value expression.

> **Note:**
>
> Support for `#length-real` is intended to imply support for integer valued <length> style value expressions as well as real valued expressions.

> **Note:**
>
> Support for `#length-real` does not, by itself, imply support for `#length-positive` or `#length-negative` features.

D.1.51 #lineBreak-uax14

A TTML transformation processor supports the `#lineBreak-uax14` feature if it recognizes and is capable of transforming requirements expressed by [UAX14] into its target document space.

A TTML presentation processor supports the `#lineBreak-uax14` feature if it implements presentation semantic support for [UAX14] as applies to line breaking.

D.1.52 #lineHeight

A TTML transformation processor supports the `#lineHeight` feature if it recognizes and is capable of transforming the `tts:lineHeight` attribute.

A TTML presentation processor supports the `#lineHeight` feature if it implements presentation semantic support for the `tts:lineHeight` attribute.

D.1.53 #markerMode

A TTML transformation processor supports the `#markerMode` feature if it recognizes and is capable of transforming the `ttp:markerMode` attribute.

A TTML presentation processor supports the `#markerMode` feature if it implements presentation semantic support for the `ttp:markerMode` attribute.

D.1.54 #markerMode-continuous

A TTML transformation processor supports the `#markerMode-continuous` feature if it recognizes and is capable of transforming the `continuous` value of the `ttp:markerMode` attribute.

A TTML presentation processor supports the `#markerMode-continuous` feature if it implements presentation semantic support for the `continuous` value of the `ttp:markerMode` attribute.

D.1.55 #markerMode-discontinuous

A TTML transformation processor supports the `#markerMode-discontinuous` feature if it recognizes and is capable of transforming the `discontinuous` value of the `ttp:markerMode` attribute.

A TTML presentation processor supports the `#markerMode-discontinuous` feature if it implements presentation semantic support for the `discontinuous` value of the `ttp:markerMode` attribute.

D.1.56 #metadata

A TTML transformation processor supports the `#metadata` feature if it recognizes and is capable of transforming the following vocabulary defined by **12 Metadata**:

- `metadata`
- `ttm:title`
- `ttm:desc`
- `ttm:copyright`
- `ttm:agent`
- `ttm:name`
- `ttm:actor`
- `@ttm:agent`
- `@ttm:role`

A TTML presentation processor supports the `#metadata` feature if it recognizes and is capable of presenting the information expressed by the same vocabulary enumerated above.

> **Note:**
>
> This specification does not define a standardized form for the presentation of metadata information. The presentation or ability to present metadata information is considered to be implementation dependent.

D.1.57 #nested-div

A TTML transformation processor supports the `#nested-div` feature if it recognizes and is capable of transforming nested `div` elements.

A TTML presentation processor supports the `#nested-div` feature if it implements presentation semantic support for nested `div` elements.

D.1.58 #nested-span

A TTML transformation processor supports the `#nested-span` feature if it recognizes and is capable of transforming nested `span` elements.

A TTML presentation processor supports the `#nested-span` feature if it implements presentation semantic support for nested `span` elements.

D.1.59 #opacity

A TTML transformation processor supports the `#opacity` feature if it recognizes and is capable of transforming the `tts:opacity` attribute.

A TTML presentation processor supports the `#opacity` feature if it (1) implements presentation semantic support for the `tts:opacity` attribute and (2) is capable of displaying or generating an output display signal that distinguishes between at least eight (8) values of opacity.

D.1.60 #origin

A TTML transformation processor supports the #origin feature if it recognizes and is capable of transforming the tts:origin attribute.

A TTML presentation processor supports the #origin feature if it implements presentation semantic support for the tts:origin attribute.

D.1.61 #overflow

A TTML transformation processor supports the #overflow feature if it recognizes and is capable of transforming all defined values of the tts:overflow attribute.

A TTML presentation processor supports the #overflow feature if it implements presentation semantic support for all defined values of the tts:overflow attribute.

D.1.62 #overflow-visible

A TTML transformation processor supports the #overflow-visible feature if it recognizes and is capable of transforming the visible value of the tts:overflow attribute.

A TTML presentation processor supports the #overflow-visible feature if it implements presentation semantic support for the visible value of the tts:overflow attribute.

D.1.63 #padding

A TTML transformation processor supports the #padding feature if it recognizes and is capable of transforming the tts:padding attribute.

A TTML presentation processor supports the #padding feature if it implements presentation semantic support for the tts:padding attribute.

D.1.64 #padding-1

A TTML transformation processor supports the #padding-1 feature if it recognizes and is capable of transforming values of the tts:padding attribute that consist of one <length> specification.

A TTML presentation processor supports the #padding-1 feature if it implements presentation semantic support for values of the tts:padding attribute that consist of one <length> specification.

D.1.65 #padding-2

A TTML transformation processor supports the #padding-2 feature if it recognizes and is capable of transforming values of the tts:padding attribute that consist of two <length> specification.

A TTML presentation processor supports the #padding-2 feature if it implements presentation semantic support for values of the tts:padding attribute that consist of two <length> specification.

D.1.66 #padding-3

A TTML transformation processor supports the #padding-3 feature if it recognizes and is capable of transforming values of the tts:padding attribute that consist of three <length> specification.

A TTML presentation processor supports the #padding-3 feature if it implements presentation semantic support for values of the tts:padding attribute that consist of three <length> specification.

D.1.67 #padding-4

A TTML transformation processor supports the #padding-4 feature if it recognizes and is capable of transforming values of the tts:padding attribute that consist of four <length> specification.

A TTML presentation processor supports the #padding-4 feature if it implements presentation semantic support for values of the tts:padding attribute that consist of four <length> specification.

D.1.68 #pixelAspectRatio

A TTML transformation processor supports the #pixelAspectRatio feature if it recognizes and is capable of transforming the ttp:pixelAspectRatio attribute.

A TTML presentation processor supports the #pixelAspectRatio feature if it implements presentation semantic support for the ttp:pixelAspectRatio attribute.

D.1.69 #presentation

A TTML processor supports the #presentation feature if it (1) satisfies the generic processor criteria defined by **3.2.1 Generic Processor Conformance**, (2) implements support for the region and line layout semantics defined by **9.3 Region Layout and Presentation** and **9.4 Line Layout**, respectively, and (3) implements presentation semantics for the following features:

- #content
- #core
- #profile
- #structure
- #time-offset
- #timing

In addition, a TTML processor that supports the #presentation feature should satisfy the user agent accessibility guidelines specified by [UAAG].

D.1.70 #profile

A TTML transformation processor supports the #profile feature if it recognizes and is capable of transforming the ttp:profile attribute on the tt element and transforming the following vocabulary defined by **6.1 Parameter Element Vocabulary**:

- ttp:profile
- ttp:features
- ttp:feature
- ttp:extensions
- ttp:extension

A TTML presentation processor supports the #profile feature if it implements presentation semantic support for the same vocabulary specified above.

D.1.71 #showBackground

A TTML transformation processor supports the `#showBackground` feature if it recognizes and is capable of transforming all defined values of the `tts:showBackground` attribute.

A TTML presentation processor supports the `#showBackground` feature if it implements presentation semantic support for all defined values of the `tts:showBackground` attribute.

D.1.72 #structure

A TTML transformation processor supports the `#structure` feature if it recognizes and is capable of transforming the following vocabulary defined by **7 Content**:

- `tt`
- `head`

A TTML presentation processor supports the `#structure` feature if it implements presentation semantic support for the same vocabulary enumerated above.

D.1.73 #styling

A TTML transformation processor supports the `#styling` feature if it recognizes and is capable of transforming the following vocabulary defined by **8 Styling**:

- `styling`
- `style`
- `@style`

A TTML presentation processor supports the `#styling` feature if it implements presentation semantic support for the same vocabulary enumerated above.

D.1.74 #styling-chained

A TTML transformation processor supports the `#styling-chained` feature if it recognizes and is capable of transforming chained style association as defined by **8.4.1.3 Chained Referential Styling**.

A TTML presentation processor supports the `#styling-chained` feature if it implements presentation semantic support for chained style association as defined by **8.4.1.3 Chained Referential Styling**.

D.1.75 #styling-inheritance-content

A TTML transformation processor supports the `#styling-inheritance` feature if it recognizes and is capable of transforming content style inheritance as defined by **8.4.2.1 Content Style Inheritance**.

A TTML presentation processor supports the `#styling-inheritance-content` feature if it implements presentation semantic support for content style inheritance as defined by **8.4.2.1 Content Style Inheritance**.

D.1.76 #styling-inheritance-region

A TTML transformation processor supports the `#styling-inheritance` feature if it recognizes and is capable of transforming region style inheritance as defined by **8.4.2.2 Region Style Inheritance**.

A TTML presentation processor supports the `#styling-inheritance-region` feature if it implements presentation semantic support for region style inheritance as defined by **8.4.2.2 Region Style Inheritance**.

D.1.77 #styling-inline

A TTML transformation processor supports the `#styling-inline` feature if it recognizes and is capable of transforming inline style association as defined by **8.4.1.1 Inline Styling**.

A TTML presentation processor supports the `#styling-inline` feature if it implements presentation semantic support for inline style association as defined by **8.4.1.1 Inline Styling**.

D.1.78 #styling-nested

A TTML transformation processor supports the `#styling-nested` feature if it recognizes and is capable of transforming nested style association as defined by **8.4.1.4 Nested Styling**.

A TTML presentation processor supports the `#styling-nested` feature if it implements presentation semantic support for nested style association as defined by **8.4.1.4 Nested Styling**.

D.1.79 #styling-referential

A TTML transformation processor supports the `#styling-referential` feature if it recognizes and is capable of transforming referential style association as defined by **8.4.1.2 Referential Styling**.

A TTML presentation processor supports the `#styling-referential` feature if it implements presentation semantic support for referential style association as defined by **8.4.1.2 Referential Styling**.

D.1.80 #subFrameRate

A TTML transformation processor supports the `#subFrameRate` feature if it recognizes and is capable of transforming the `ttp:subFrameRate` attribute.

A TTML presentation processor supports the `#subFrameRate` feature if it implements presentation semantic support for the `ttp:subFrameRate` attribute.

D.1.81 #textAlign

A TTML transformation processor supports the `#textAlign` feature if it recognizes and is capable of transforming all defined values of the `tts:textAlign` attribute.

A TTML presentation processor supports the `#textAlign` feature if it implements presentation semantic support for all defined values of the `tts:textAlign` attribute.

D.1.82 #textAlign-absolute

A TTML transformation processor supports the `#textAlign-absolute` feature if it recognizes and is capable of transforming the `left`, `center`, and `right` values of the `tts:textAlign` attribute.

A TTML presentation processor supports the `#textAlign-absolute` feature if it implements presentation semantic support for the `left`, `center`, and `right` values of the `tts:textAlign` attribute.

D.1.83 #textAlign-relative

A TTML transformation processor supports the `#textAlign-relative` feature if it recognizes and is capable of transforming the `start`, `center`, and `end` values of the `tts:textAlign` attribute.

A TTML presentation processor supports the `#textAlign-relative` feature if it implements presentation semantic support for the `start`, `center`, and `end` values of the `tts:textAlign` attribute.

D.1.84 #textDecoration

A TTML transformation processor supports the `#textDecoration` feature if it recognizes and is capable of transforming all defined values of the `tts:textDecoration` attribute.

A TTML presentation processor supports the `#textDecoration` feature if it implements presentation semantic support for all defined values of the `tts:textDecoration` attribute.

D.1.85 #textDecoration-over

A TTML transformation processor supports the `#textDecoration-over` feature if it recognizes and is capable of transforming the `overline` and `noOverline` values of the `tts:textDecoration` attribute.

A TTML presentation processor supports the `#textDecoration-over` feature if it implements presentation semantic support for the `overline` and `noOverline` values of the `tts:textDecoration` attribute.

D.1.86 #textDecoration-through

A TTML transformation processor supports the `#textDecoration-through` feature if it recognizes and is capable of transforming the `lineThrough` and `noLineThrough` values of the `tts:textDecoration` attribute.

A TTML presentation processor supports the `#textDecoration-through` feature if it implements presentation semantic support for the `lineThrough` and `noLineThrough` values of the `tts:textDecoration` attribute.

D.1.87 #textDecoration-under

A TTML transformation processor supports the `#textDecoration-under` feature if it recognizes and is capable of transforming the `underline` and `noUnderline` values of the `tts:textDecoration` attribute.

A TTML presentation processor supports the `#textDecoration-under` feature if it implements presentation semantic support for the `underline` and `noUnderline` values of the `tts:textDecoration` attribute.

D.1.88 #textOutline

A TTML processor supports the `#textOutline` feature if it supports the following features:

- `#textOutline-blurred`
- `#textOutline-unblurred`

D.1.89 #textOutline-blurred

A TTML transformation processor supports the `#textOutline-blurred` feature if it recognizes and is capable of transforming values of the `tts:textOutline` attribute that includes a blur radius specification.

A TTML presentation processor supports the `#textOutline-blurred` feature if it implements presentation semantic support for values of the `tts:textOutline` attribute that includes a blur radius specification.

D.1.90 #textOutline-unblurred

A TTML transformation processor supports the `#textOutline-unblurred` feature if it recognizes and is capable of transforming values of the `tts:textOutline` attribute that does not include a blur radius specification.

A TTML presentation processor supports the `#textOutline-unblurred` feature if it implements presentation semantic support for values of the `tts:textOutline` attribute that does not include a blur radius specification.

D.1.91 #tickRate

A TTML transformation processor supports the `#tickRate` feature if it recognizes and is capable of transforming the `ttp:tickRate` attribute.

A TTML presentation processor supports the `#tickRate` feature if it implements presentation semantic support for the `ttp:tickRate` attribute.

D.1.92 #timeBase-clock

A TTML transformation processor supports the `#timeBase-clock` feature if it recognizes and is capable of transforming the `clock` value of the `ttp:timeBase` attribute and if it supports the `#clockMode` feature.

A TTML presentation processor supports the `#timeBase-clock` feature if it implements presentation semantic support for the `clock` value of the `ttp:timeBase` attribute and if it supports the `#clockMode` feature.

D.1.93 #timeBase-media

A TTML transformation processor supports the `#timeBase-media` feature if it recognizes and is capable of transforming the `media` value of the `ttp:timeBase` attribute.

A TTML presentation processor supports the `#timeBase-media` feature if it implements presentation semantic support for the `media` value of the `ttp:timeBase` attribute.

D.1.94 #timeBase-smpte

A TTML transformation processor supports the `#timeBase-smpte` feature if it recognizes and is capable of transforming the `smpte` value of the `ttp:timeBase` attribute and if it supports the `#dropMode` feature.

A TTML presentation processor supports the `#timeBase-smpte` feature if it implements presentation semantic support for the `smpte` value of the `ttp:timeBase` attribute and if it supports the `#dropMode` feature.

D.1.95 #timeContainer

A TTML transformation processor supports the `#timeContainer` feature if it recognizes and is capable of transforming the timeContainer attribute.

A TTML presentation processor supports the #timeContainer feature if it implements presentation semantic support for the timeContainer attribute.

D.1.96 #time-clock

A TTML transformation processor supports the #time-clock feature if it recognizes and is capable of transforming all values of the <timeExpression> that satisfy the following subset of time expression syntax:

```
<timeExpression>
  : hours ":" minutes ":" seconds ( fraction )?
```

A TTML presentation processor supports the #time-clock feature if it implements presentation semantic support for the same syntax specified above.

D.1.97 #time-clock-with-frames

A TTML transformation processor supports the #time-clock-with-frames feature if it supports the #frameRate, #frameRateMultiplier, and #subFrameRate features and if it recognizes and is capable of transforming all values of the <timeExpression> that satisfy the following subset of time expression syntax:

```
<timeExpression>
  : hours ":" minutes ":" seconds ( fraction | ":" frames ( "." sub-frames )? )?
```

A TTML presentation processor supports the #time-clock-with-frames feature if it implements presentation semantic support for the same features and syntax specified above.

D.1.98 #time-offset

A TTML transformation processor supports the #time-offset feature if it recognizes and is capable of transforming all values of the <timeExpression> that satisfy the following subset of time expression syntax:

```
<timeExpression>
  : time-count fraction? ( "h" | "m" | "s" | "ms" )
```

A TTML presentation processor supports the #time-offset feature if it implements presentation semantic support for the same syntax specified above.

D.1.99 #time-offset-with-frames

A TTML transformation processor supports the #time-offset-with-frames feature if it supports the #frameRate, #frameRateMultiplier, and #subFrameRate features and if it recognizes and is capable of transforming all values of the <timeExpression> that satisfy the following subset of time expression syntax:

```
<timeExpression>
  : time-count fraction? "f"
```

A TTML presentation processor supports the #time-offset-with-frames feature if it implements presentation semantic support for the same features and syntax specified above.

D.1.100 #time-offset-with-ticks

A TTML transformation processor supports the #time-offset-with-ticks feature if it supports the #tickRate feature and if it recognizes and is capable of transforming all values of the <timeExpression> that satisfy the following subset of time expression syntax:

```
<timeExpression>
  : time-count fraction? "t"
```

A TTML presentation processor supports the #time-offset-with-ticks feature if it implements presentation semantic support for the same features and syntax specified above.

D.1.101 #timing

A TTML transformation processor supports the #timing feature if it recognizes and is capable of transforming the following vocabulary defined by **10 Timing**:

- @begin
- @dur
- @end

A TTML presentation processor supports the #timing feature if it implements presentation semantic support for the same vocabulary enumerated above.

D.1.102 #transformation

A TTML processor supports the #transformation feature if it (1) satisfies the generic processor criteria defined by **3.2.1 Generic Processor Conformance** and (2) implements the transformation semantics of the following features:

- #content
- #core
- #profile
- #structure
- #time-offset
- #timing

D.1.103 #unicodeBidi

A TTML transformation processor supports the #unicodeBidi feature if it recognizes and is capable of transforming all defined values of the tts:unicodeBidi attribute.

A TTML presentation processor supports the #unicodeBidi feature if it implements presentation semantic support for all defined values of the tts:unicodeBidi attribute.

D.1.104 #visibility

A TTML transformation processor supports the #visibility feature if it recognizes and is capable of transforming all defined values of the tts:visibility attribute.

A TTML presentation processor supports the #visibility feature if it implements presentation semantic support for all defined values of the `tts:visibility` attribute.

D.1.105 #visibility-block

A TTML transformation processor supports the `#visibility-block` feature if it recognizes and is capable of transforming all defined values of the `tts:visibility` attribute when applied to a content element that would generate a block area during presentation processing.

A TTML presentation processor supports the `#visibility-block` feature if it implements presentation semantic support for all defined values of the `tts:visibility` attribute when applied to a content element that generates a block area.

D.1.106 #visibility-inline

A TTML transformation processor supports the `#visibility-inline` feature if it recognizes and is capable of transforming all defined values of the `tts:visibility` attribute when applied to a content element that would generate an inline area during presentation processing.

A TTML presentation processor supports the `#visibility-inline` feature if it implements presentation semantic support for all defined values of the `tts:visibility` attribute when applied to a content element that generates an inline area.

D.1.107 #visibility-region

A TTML transformation processor supports the `#visibility-region` feature if it recognizes and is capable of transforming all defined values of the `tts:visibility` attribute when applied to a `region` element.

A TTML presentation processor supports the `#visibility-region` feature if it implements presentation semantic support for all defined values of the `tts:visibility` attribute when applied to a `region` element.

D.1.108 #wrapOption

A TTML transformation processor supports the `#wrapOption` feature if it recognizes and is capable of transforming all defined values of the `tts:wrapOption` attribute.

A TTML presentation processor supports the `#wrapOption` feature if it implements presentation semantic support for all defined values of the `tts:wrapOption` attribute.

D.1.109 #writingMode

A TTML processor supports the `#writingMode` feature if it supports the following features:

- `#writingMode-horizontal`
- `#writingMode-vertical`

D.1.110 #writingMode-vertical

A TTML transformation processor supports the `#writingMode-vertical` feature if it recognizes and is capable of transforming the `tbrl`, `tblr`, and `tb` values of the `tts:writingMode` attribute.

A TTML presentation processor supports the `#writingMode-vertical` feature if it implements presentation semantic support for the `tbrl`, `tblr`, and `tb` values of the `tts:writingMode` attribute.

D.1.111 #writingMode-horizontal

A TTML processor supports the `#writingMode-horizontal` feature if it supports the following features:

- `#writingMode-horizontal-lr`
- `#writingMode-horizontal-rl`

D.1.112 #writingMode-horizontal-lr

A TTML transformation processor supports the `#writingMode-horizontal` feature if it recognizes and is capable of transforming the `lrtb` and `lr` values of the `tts:writingMode` attribute.

A TTML presentation processor supports the `#writingMode-horizontal-lr` feature if it implements presentation semantic support for the `lrtb` and `lr` values of the `tts:writingMode` attribute.

D.1.113 #writingMode-horizontal-rl

A TTML transformation processor supports the `#writingMode-horizontal` feature if it recognizes and is capable of transforming the `rltb` and `rl` values of the `tts:writingMode` attribute.

A TTML presentation processor supports the `#writingMode-horizontal-rl` feature if it implements presentation semantic support for the `rltb` and `rl` values of the `tts:writingMode` attribute.

D.1.114 #zIndex

A TTML transformation processor supports the `#zIndex` feature if it recognizes and is capable of transforming the `tts:zIndex` attribute.

A TTML presentation processor supports the `#zIndex` feature if it implements presentation semantic support for the `tts:zIndex` attribute.

D.2 Feature Support

The following table, **Table D-1 – Feature Support**, enumerates every defined feature designation (expressed without the TT Feature Namespace), and, for each designated feature, specifies whether the feature must be implemented, i.e., is mandatory (M), or may be implemented, i.e., is optional (O), for transformation and presentation processors.

Table D-1 – Feature Support

Feature	Transformation	Presentation
`#animation`	O	O
`#backgroundColor`	O	O
`#backgroundColor-block`	O	O
`#backgroundColor-inline`	O	O
`#backgroundColor-region`	O	O
`#bidi`	O	O
`#cellResolution`	O	O
`#clockMode`	O	O
`#clockMode-gps`	O	O
`#clockMode-local`	O	O
`#clockMode-utc`	O	O

#color	O	O
#content	M	M
#core	M	M
#direction	O	O
#display	O	O
#display-block	O	O
#display-inline	O	O
#display-region	O	O
#displayAlign	O	O
#dropMode	O	O
#dropMode-dropNTSC	O	O
#dropMode-dropPAL	O	O
#dropMode-nonDrop	O	O
#extent	O	O
#extent-region	O	O
#extent-root	O	O
#fontFamily	O	O
#fontFamily-generic	O	O
#fontFamily-non-generic	O	O
#fontSize	O	O
#fontSize-anamorphic	O	O
#fontSize-isomorphic	O	O
#fontStyle	O	O
#fontStyle-italic	O	O
#fontStyle-oblique	O	O
#fontWeight	O	O
#fontWeight-bold	O	O
#frameRate	O	O
#frameRateMultiplier	O	O
#layout	O	O
#length	O	O
#length-cell	O	O
#length-em	O	O
#length-integer	O	O
#length-negative	O	O
#length-percentage	O	O
#length-pixel	O	O
#length-positive	O	O
#length-real	O	O
#lineBreak-uax14	O	O
#lineHeight	O	O
#markerMode	O	O
#markerMode-continuous	O	O
#markerMode-discontinuous	O	O
#metadata	O	O
#nested-div	O	O
#nested-span	O	O
#opacity	O	O
#origin	O	O

#overflow	O	O
#overflow-visible	O	O
#padding	O	O
#padding-1	O	O
#padding-2	O	O
#padding-3	O	O
#padding-4	O	O
#pixelAspectRatio	O	O
#presentation	O	M
#profile	M	M
#showBackground	O	O
#structure	M	M
#styling	O	O
#styling-chained	O	O
#styling-inheritance-content	O	O
#styling-inheritance-region	O	O
#styling-inline	O	O
#styling-nested	O	O
#styling-referential	O	O
#subFrameRate	O	O
#textAlign	O	O
#textAlign-absolute	O	O
#textAlign-relative	O	O
#textDecoration	O	O
#textDecoration-over	O	O
#textDecoration-through	O	O
#textDecoration-under	O	O
#textOutline	O	O
#textOutline-blurred	O	O
#textOutline-unblurred	O	O
#tickRate	O	O
#timeBase-clock	O	O
#timeBase-media	O	O
#timeBase-smpte	O	O
#timeContainer	O	O
#time-clock	O	O
#time-clock-with-frames	O	O
#time-offset	M	M
#time-offset-with-frames	O	O
#time-offset-with-ticks	O	O
#timing	M	M
#transformation	M	O
#unicodeBidi	O	O
#visibility	O	O
#visibility-block	O	O
#visibility-inline	O	O
#visibility-region	O	O
#wrapOption	O	O
#writingMode	O	O

#writingMode-vertical	O	O
#writingMode-horizontal	O	O
#writingMode-horizontal-lr	O	O
#writingMode-horizontal-rl	O	O
#zIndex	O	O

For the sake of convenience, the following table, **Table D-2 – Mandatory Features - Transformation**, enumerates all mandatory features for a TTML Transformation Processor, providing additional comments to summarize the context of usage or the nature of the feature. The *Profile Definition Document* that defines the corresponding DFXP Transformation Profile is specified in **F.1 DFXP Transformation Profile**.

Table D-2 – Mandatory Features - Transformation

Feature	**Comments**
#content	body, div, p, span, br
#core	@xml:id, @xml:lang, @xml:space
#profile	
#structure	tt, head
#time-offset	
#timing	@begin, @dur, @end
#transformation	

For the sake of convenience, the following table, **Table D-3 – Mandatory Features - Presentation**, enumerates all mandatory features for a TTML Presentation Processor, providing additional comments to summarize the context of usage or the nature of the feature. The *Profile Definition Document* that defines the corresponding DFXP Presentation Profile is specified in **F.2 DFXP Presentation Profile**.

Table D-3 – Mandatory Features - Presentation

Feature	**Comments**
#content	body, div, p, span, br
#core	@xml:id, @xml:lang, @xml:space
#profile	
#presentation	
#structure	tt, head
#time-offset	
#timing	@begin, @dur, @end

E Extensions

This appendix is normative.

This appendix specifies the syntactic form of extension designations, which are used to express authorial intent regarding the support for extension mechanisms in a TTML processor.

E.1 Extension Designations

An extension designation is expressed as a string that adheres to the following form:

```
extension-designation
  : extension-namespace designation

extension-namespace
```

```
  : TT Extension Namespace                  // http://www.w3.org/ns/ttml/extension/
  | Other Extension Namespace               // expressed as an absolute URI

designation
  : "#" token-char+

token-char
  : { XML NameChar }                        // XML 1.1 Production [4a]
```

If the extension namespace of an extension designation is the TT Extension Namespace, then all values of the following `designation` token are reserved for future standardization.

If the extension namespace of an extension designation is not the TT Extension Namespace, i.e., is an *Other Extension Namespace*, then the extension namespace must be expressed as an absolute URI capable of serving as a base URI used in combination with a `designation` token that takes the form of a fragment identifier.

F Profiles

This appendix is normative.

This appendix specifies the following standard TTML profiles:

- **F.1 DFXP Transformation Profile**
- **F.2 DFXP Presentation Profile**
- **F.3 DFXP Full Profile**

The SDP US profile is defined in TTML Simple Delivery Profile for Closed Captions (US).

Each TTML profile is defined in terms of a *Profile Definition Document*, which is expressed as an XML document wherein the root element adheres to **6.1.1 ttp:profile**.

F.1 DFXP Transformation Profile

The DFXP Transformation Profile is intended to be used to express minimum compliance for transformation processing.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- this file defines the "dfxp-transformation" profile of ttml -->
<profile xmlns="http://www.w3.org/ns/ttml#parameter">
  <features xml:base="http://www.w3.org/ns/ttml/feature/">
    <!-- required (mandatory) feature support -->
    <feature value="required">#content</feature>
    <feature value="required">#core</feature>
    <feature value="required">#profile</feature>
    <feature value="required">#structure</feature>
    <feature value="required">#time-offset</feature>
    <feature value="required">#timing</feature>
    <feature value="required">#transformation</feature>
    <!-- optional (voluntary) feature support -->
    <feature value="optional">#animation</feature>
    <feature value="optional">#backgroundColor-block</feature>
    <feature value="optional">#backgroundColor-inline</feature>
    <feature value="optional">#backgroundColor-region</feature>
    <feature value="optional">#backgroundColor</feature>
    <feature value="optional">#bidi</feature>
    <feature value="optional">#cellResolution</feature>
```

```
<feature value="optional">#clockMode</feature>
<feature value="optional">#clockMode-gps</feature>
<feature value="optional">#clockMode-local</feature>
<feature value="optional">#clockMode-utc</feature>
<feature value="optional">#color</feature>
<feature value="optional">#direction</feature>
<feature value="optional">#display</feature>
<feature value="optional">#display-block</feature>
<feature value="optional">#display-inline</feature>
<feature value="optional">#display-region</feature>
<feature value="optional">#displayAlign</feature>
<feature value="optional">#dropMode</feature>
<feature value="optional">#dropMode-dropNTSC</feature>
<feature value="optional">#dropMode-dropPAL</feature>
<feature value="optional">#dropMode-nonDrop</feature>
<feature value="optional">#extent</feature>
<feature value="optional">#extent-region</feature>
<feature value="optional">#extent-root</feature>
<feature value="optional">#fontFamily</feature>
<feature value="optional">#fontFamily-generic</feature>
<feature value="optional">#fontFamily-non-generic</feature>
<feature value="optional">#fontSize</feature>
<feature value="optional">#fontSize-anamorphic</feature>
<feature value="optional">#fontSize-isomorphic</feature>
<feature value="optional">#fontStyle</feature>
<feature value="optional">#fontStyle-italic</feature>
<feature value="optional">#fontStyle-oblique</feature>
<feature value="optional">#fontWeight</feature>
<feature value="optional">#fontWeight-bold</feature>
<feature value="optional">#frameRate</feature>
<feature value="optional">#frameRateMultiplier</feature>
<feature value="optional">#layout</feature>
<feature value="optional">#length</feature>
<feature value="optional">#length-cell</feature>
<feature value="optional">#length-em</feature>
<feature value="optional">#length-integer</feature>
<feature value="optional">#length-negative</feature>
<feature value="optional">#length-percentage</feature>
<feature value="optional">#length-pixel</feature>
<feature value="optional">#length-positive</feature>
<feature value="optional">#length-real</feature>
<feature value="optional">#lineBreak-uax14</feature>
<feature value="optional">#lineHeight</feature>
<feature value="optional">#markerMode</feature>
<feature value="optional">#markerMode-continuous</feature>
<feature value="optional">#markerMode-discontinuous</feature>
<feature value="optional">#metadata</feature>
<feature value="optional">#nested-div</feature>
<feature value="optional">#nested-span</feature>
<feature value="optional">#opacity</feature>
<feature value="optional">#origin</feature>
<feature value="optional">#overflow</feature>
<feature value="optional">#overflow-visible</feature>
<feature value="optional">#padding</feature>
<feature value="optional">#padding-1</feature>
<feature value="optional">#padding-2</feature>
<feature value="optional">#padding-3</feature>
<feature value="optional">#padding-4</feature>
<feature value="optional">#pixelAspectRatio</feature>
```

```
    <feature value="optional">#presentation</feature>
    <feature value="optional">#showBackground</feature>
    <feature value="optional">#styling</feature>
    <feature value="optional">#styling-chained</feature>
    <feature value="optional">#styling-inheritance-content</feature>
    <feature value="optional">#styling-inheritance-region</feature>
    <feature value="optional">#styling-inline</feature>
    <feature value="optional">#styling-nested</feature>
    <feature value="optional">#styling-referential</feature>
    <feature value="optional">#subFrameRate</feature>
    <feature value="optional">#textAlign</feature>
    <feature value="optional">#textAlign-absolute</feature>
    <feature value="optional">#textAlign-relative</feature>
    <feature value="optional">#textDecoration</feature>
    <feature value="optional">#textDecoration-over</feature>
    <feature value="optional">#textDecoration-through</feature>
    <feature value="optional">#textDecoration-under</feature>
    <feature value="optional">#textOutline</feature>
    <feature value="optional">#textOutline-blurred</feature>
    <feature value="optional">#textOutline-unblurred</feature>
    <feature value="optional">#tickRate</feature>
    <feature value="optional">#time-clock-with-frames</feature>
    <feature value="optional">#time-clock</feature>
    <feature value="optional">#time-offset-with-frames</feature>
    <feature value="optional">#time-offset-with-ticks</feature>
    <feature value="optional">#timeBase-clock</feature>
    <feature value="optional">#timeBase-media</feature>
    <feature value="optional">#timeBase-smpte</feature>
    <feature value="optional">#timeContainer</feature>
    <feature value="optional">#unicodeBidi</feature>
    <feature value="optional">#visibility</feature>
    <feature value="optional">#visibility-block</feature>
    <feature value="optional">#visibility-inline</feature>
    <feature value="optional">#visibility-region</feature>
    <feature value="optional">#wrapOption</feature>
    <feature value="optional">#writingMode</feature>
    <feature value="optional">#writingMode-horizontal-lr</feature>
    <feature value="optional">#writingMode-horizontal-rl</feature>
    <feature value="optional">#writingMode-horizontal</feature>
    <feature value="optional">#writingMode-vertical</feature>
    <feature value="optional">#zIndex</feature>
  </features>
  <extensions xml:base="http://www.w3.org/ns/ttml/extension/">
    <!-- required (mandatory) extension support -->
    <!-- optional (voluntary) extension support -->
  </extensions>
</profile>
```

F.2 DFXP Presentation Profile

The DFXP Presentation Profile is intended to be used to express minimum compliance for presentation processing.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- this file defines the "dfxp-presentation" profile of ttml -->
<profile xmlns="http://www.w3.org/ns/ttml#parameter">
  <features xml:base="http://www.w3.org/ns/ttml/feature/">
```

```
<!-- required (mandatory) feature support -->
<feature value="required">#content</feature>
<feature value="required">#core</feature>
<feature value="required">#presentation</feature>
<feature value="required">#profile</feature>
<feature value="required">#structure</feature>
<feature value="required">#time-offset</feature>
<feature value="required">#timing</feature>
<!-- optional (voluntary) feature support -->
<feature value="optional">#animation</feature>
<feature value="optional">#backgroundColor</feature>
<feature value="optional">#backgroundColor-block</feature>
<feature value="optional">#backgroundColor-inline</feature>
<feature value="optional">#backgroundColor-region</feature>
<feature value="optional">#bidi</feature>
<feature value="optional">#cellResolution</feature>
<feature value="optional">#clockMode</feature>
<feature value="optional">#clockMode-gps</feature>
<feature value="optional">#clockMode-local</feature>
<feature value="optional">#clockMode-utc</feature>
<feature value="optional">#color</feature>
<feature value="optional">#direction</feature>
<feature value="optional">#display</feature>
<feature value="optional">#display-block</feature>
<feature value="optional">#display-inline</feature>
<feature value="optional">#display-region</feature>
<feature value="optional">#displayAlign</feature>
<feature value="optional">#dropMode</feature>
<feature value="optional">#dropMode-dropNTSC</feature>
<feature value="optional">#dropMode-dropPAL</feature>
<feature value="optional">#dropMode-nonDrop</feature>
<feature value="optional">#extent</feature>
<feature value="optional">#extent-region</feature>
<feature value="optional">#extent-root</feature>
<feature value="optional">#fontFamily</feature>
<feature value="optional">#fontFamily-generic</feature>
<feature value="optional">#fontFamily-non-generic</feature>
<feature value="optional">#fontSize</feature>
<feature value="optional">#fontSize-anamorphic</feature>
<feature value="optional">#fontSize-isomorphic</feature>
<feature value="optional">#fontStyle</feature>
<feature value="optional">#fontStyle-italic</feature>
<feature value="optional">#fontStyle-oblique</feature>
<feature value="optional">#fontWeight</feature>
<feature value="optional">#fontWeight-bold</feature>
<feature value="optional">#frameRate</feature>
<feature value="optional">#frameRateMultiplier</feature>
<feature value="optional">#layout</feature>
<feature value="optional">#length</feature>
<feature value="optional">#length-cell</feature>
<feature value="optional">#length-em</feature>
<feature value="optional">#length-integer</feature>
<feature value="optional">#length-negative</feature>
<feature value="optional">#length-percentage</feature>
<feature value="optional">#length-pixel</feature>
<feature value="optional">#length-positive</feature>
<feature value="optional">#length-real</feature>
<feature value="optional">#lineBreak-uax14</feature>
<feature value="optional">#lineHeight</feature>
```

```
  <feature value="optional">#markerMode</feature>
  <feature value="optional">#markerMode-continuous</feature>
  <feature value="optional">#markerMode-discontinuous</feature>
  <feature value="optional">#metadata</feature>
  <feature value="optional">#nested-div</feature>
  <feature value="optional">#nested-span</feature>
  <feature value="optional">#opacity</feature>
  <feature value="optional">#origin</feature>
  <feature value="optional">#overflow</feature>
  <feature value="optional">#overflow-visible</feature>
  <feature value="optional">#padding</feature>
  <feature value="optional">#padding-1</feature>
  <feature value="optional">#padding-2</feature>
  <feature value="optional">#padding-3</feature>
  <feature value="optional">#padding-4</feature>
  <feature value="optional">#pixelAspectRatio</feature>
  <feature value="optional">#showBackground</feature>
  <feature value="optional">#styling</feature>
  <feature value="optional">#styling-chained</feature>
  <feature value="optional">#styling-inheritance-content</feature>
  <feature value="optional">#styling-inheritance-region</feature>
  <feature value="optional">#styling-inline</feature>
  <feature value="optional">#styling-nested</feature>
  <feature value="optional">#styling-referential</feature>
  <feature value="optional">#subFrameRate</feature>
  <feature value="optional">#textAlign</feature>
  <feature value="optional">#textAlign-absolute</feature>
  <feature value="optional">#textAlign-relative</feature>
  <feature value="optional">#textDecoration</feature>
  <feature value="optional">#textDecoration-over</feature>
  <feature value="optional">#textDecoration-through</feature>
  <feature value="optional">#textDecoration-under</feature>
  <feature value="optional">#textOutline</feature>
  <feature value="optional">#textOutline-blurred</feature>
  <feature value="optional">#textOutline-unblurred</feature>
  <feature value="optional">#tickRate</feature>
  <feature value="optional">#time-clock</feature>
  <feature value="optional">#time-clock-with-frames</feature>
  <feature value="optional">#time-offset-with-frames</feature>
  <feature value="optional">#time-offset-with-ticks</feature>
  <feature value="optional">#timeBase-clock</feature>
  <feature value="optional">#timeBase-media</feature>
  <feature value="optional">#timeBase-smpte</feature>
  <feature value="optional">#timeContainer</feature>
  <feature value="optional">#transformation</feature>
  <feature value="optional">#unicodeBidi</feature>
  <feature value="optional">#visibility</feature>
  <feature value="optional">#visibility-block</feature>
  <feature value="optional">#visibility-inline</feature>
  <feature value="optional">#visibility-region</feature>
  <feature value="optional">#wrapOption</feature>
  <feature value="optional">#writingMode</feature>
  <feature value="optional">#writingMode-horizontal-lr</feature>
  <feature value="optional">#writingMode-horizontal-rl</feature>
  <feature value="optional">#writingMode-horizontal</feature>
  <feature value="optional">#writingMode-vertical</feature>
  <feature value="optional">#zIndex</feature>
</features>
<extensions xml:base="http://www.w3.org/ns/ttml/extension/">
```

```
    <!-- required (mandatory) extension support -->
    <!-- optional (voluntary) extension support -->
  </extensions>
</profile>
```

F.3 DFXP Full Profile

The DFXP Full Profile is intended to be used to express maximum compliance for both transformation and presentation processing.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- this file defines the "dfxp-full" profile of ttml -->
<profile xmlns="http://www.w3.org/ns/ttml#parameter">
  <features xml:base="http://www.w3.org/ns/ttml/feature/">
    <!-- required (mandatory) feature support -->
    <feature value="required">#animation</feature>
    <feature value="required">#backgroundColor-block</feature>
    <feature value="required">#backgroundColor-inline</feature>
    <feature value="required">#backgroundColor-region</feature>
    <feature value="required">#backgroundColor</feature>
    <feature value="required">#bidi</feature>
    <feature value="required">#cellResolution</feature>
    <feature value="required">#clockMode-gps</feature>
    <feature value="required">#clockMode-local</feature>
    <feature value="required">#clockMode-utc</feature>
    <feature value="required">#clockMode</feature>
    <feature value="required">#color</feature>
    <feature value="required">#content</feature>
    <feature value="required">#core</feature>
    <feature value="required">#direction</feature>
    <feature value="required">#display</feature>
    <feature value="required">#display-block</feature>
    <feature value="required">#display-inline</feature>
    <feature value="required">#display-region</feature>
    <feature value="required">#displayAlign</feature>
    <feature value="required">#dropMode</feature>
    <feature value="required">#dropMode-dropNTSC</feature>
    <feature value="required">#dropMode-dropPAL</feature>
    <feature value="required">#dropMode-nonDrop</feature>
    <feature value="required">#extent</feature>
    <feature value="required">#extent-region</feature>
    <feature value="required">#extent-root</feature>
    <feature value="required">#fontFamily</feature>
    <feature value="required">#fontFamily-generic</feature>
    <feature value="required">#fontFamily-non-generic</feature>
    <feature value="required">#fontSize</feature>
    <feature value="required">#fontSize-anamorphic</feature>
    <feature value="required">#fontSize-isomorphic</feature>
    <feature value="required">#fontStyle</feature>
    <feature value="required">#fontStyle-italic</feature>
    <feature value="required">#fontStyle-oblique</feature>
    <feature value="required">#fontWeight</feature>
    <feature value="required">#fontWeight-bold</feature>
    <feature value="required">#frameRate</feature>
    <feature value="required">#frameRateMultiplier</feature>
    <feature value="required">#layout</feature>
    <feature value="required">#length</feature>
    <feature value="required">#length-cell</feature>
    <feature value="required">#length-em</feature>
    <feature value="required">#length-integer</feature>
    <feature value="required">#length-negative</feature>
    <feature value="required">#length-percentage</feature>
    <feature value="required">#length-pixel</feature>
    <feature value="required">#length-positive</feature>
    <feature value="required">#length-real</feature>
    <feature value="required">#lineBreak-uax14</feature>
    <feature value="required">#lineHeight</feature>
    <feature value="required">#markerMode</feature>
    <feature value="required">#markerMode-continuous</feature>
    <feature value="required">#markerMode-discontinuous</feature>
    <feature value="required">#metadata</feature>
    <feature value="required">#nested-div</feature>
    <feature value="required">#nested-span</feature>
    <feature value="required">#opacity</feature>
    <feature value="required">#origin</feature>
    <feature value="required">#overflow</feature>
    <feature value="required">#overflow-visible</feature>
    <feature value="required">#padding</feature>
    <feature value="required">#padding-1</feature>
    <feature value="required">#padding-2</feature>
    <feature value="required">#padding-3</feature>
    <feature value="required">#padding-4</feature>
    <feature value="required">#pixelAspectRatio</feature>
    <feature value="required">#presentation</feature>
    <feature value="required">#profile</feature>
    <feature value="required">#showBackground</feature>
    <feature value="required">#structure</feature>
    <feature value="required">#styling</feature>
    <feature value="required">#styling-chained</feature>
    <feature value="required">#styling-inheritance-content</feature>
    <feature value="required">#styling-inheritance-region</feature>
    <feature value="required">#styling-inline</feature>
    <feature value="required">#styling-nested</feature>
    <feature value="required">#styling-referential</feature>
    <feature value="required">#subFrameRate</feature>
    <feature value="required">#textAlign</feature>
    <feature value="required">#textAlign-absolute</feature>
    <feature value="required">#textAlign-relative</feature>
    <feature value="required">#textDecoration</feature>
    <feature value="required">#textDecoration-over</feature>
    <feature value="required">#textDecoration-through</feature>
    <feature value="required">#textDecoration-under</feature>
    <feature value="required">#textOutline</feature>
    <feature value="required">#textOutline-blurred</feature>
    <feature value="required">#textOutline-unblurred</feature>
    <feature value="required">#tickRate</feature>
    <feature value="required">#time-clock</feature>
    <feature value="required">#time-clock-with-frames</feature>
    <feature value="required">#time-offset</feature>
    <feature value="required">#time-offset-with-frames</feature>
    <feature value="required">#time-offset-with-ticks</feature>
    <feature value="required">#timeBase-clock</feature>
    <feature value="required">#timeBase-media</feature>
    <feature value="required">#timeBase-smpte</feature>
    <feature value="required">#timeContainer</feature>
```

```
      <feature value="required">#timing</feature>
      <feature value="required">#transformation</feature>
      <feature value="required">#unicodeBidi</feature>
      <feature value="required">#visibility</feature>
      <feature value="required">#visibility-block</feature>
      <feature value="required">#visibility-inline</feature>
      <feature value="required">#visibility-region</feature>
      <feature value="required">#wrapOption</feature>
      <feature value="required">#writingMode-horizontal</feature>
      <feature value="required">#writingMode-horizontal-lr</feature>
      <feature value="required">#writingMode-horizontal-rl</feature>
      <feature value="required">#writingMode-vertical</feature>
      <feature value="required">#writingMode</feature>
      <feature value="required">#zIndex</feature>
      <!-- optional (voluntary) feature support -->
    </features>
    <extensions xml:base="http://www.w3.org/ns/ttml/extension/">
      <!-- required (mandatory) extension support -->
      <!-- optional (voluntary) extension support -->
    </extensions>
  </profile>
```

G References

This appendix is normative.

CSS3 Color

Tantek Çelik and Chris Lilley, *CSS Color Module Level 3*, W3C Recommendation, 07 June 2011. (See http://www.w3.org/TR/2011/REC-css3-color-20110607/.)

GPS

Global Positioning System, US Naval Observatory. (See http://tycho.usno.navy.mil/gpsinfo.html.)

Media Types

Ned Freed and Nathaniel Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, RFC 2046, November 1996, IETF.(See http://www.rfc-editor.org/rfc/rfc2046.txt.)

RELAX NG

ISO/IEC 19757-2, *Information technology – Document Schema Definition Language (DSDL) – Part 2: Regular-grammar-based validation – RELAX NG*, International Organization for Standardization (ISO).

SMIL 2.1

Dick Bultermann, et al., *Synchronized Multimedia Integration Language (SMIL 2.1)*, W3C Recommendation, 13 December 2005. (See http://www.w3.org/TR/2005/REC-SMIL2-20051213/.)

SMPTE 12M

ANSI/SMPTE 12M, *Television, Audio and Film – Time and Control Code*, SMPTE Standard.

SRGB

IEC 61966-2-1, *Multimedia systems and equipment – Colour measurement and management – Part 2-1: Colour management – Default RGB colour space – sRGB*, International Electrotechnical Commission (IEC).

UAAG

Ian Jacobs, Jon Gunderson, and Eric Hansen, Eds., *User Agent Accessibility Guidelines 1.0*, W3C Recommendation, 17 December 2002. (See http://www.w3.org/TR/2002/REC-UAAG10-20021217/.)

UAX9

M. Davis, A. Lanin, and A. Glass, *Unicode Bidirectional Algorithm*, Unicode Consortium, 14 May 2017. (See http://www.unicode.org/reports/tr9/tr9-37.html.)

UAX14

Asmus Freytag, *Line Breaking Properties*, Unicode Consortium, 29 August 2005. (See http://www.unicode.org/reports/tr14/tr14-17.html.)

UTC

Recommendation TF.460, *Standard-Frequency and Time-Signal Emissions*, International Telecommunciations Union, Radio Sector (ITU-R).

WCAG

Ben Caldwell, et al., Eds., *Web Content Accessibility Guidelines (WCAG) 2.0*, W3C Recommendation, 11 December 2008. (See http://www.w3.org/TR/2008/REC-WCAG20-20081211/.)

XML 1.0

Tim Bray, et al. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, W3C Recommendation, 26 November 2008. (See http://www.w3.org/TR/2008/REC-xml-20081126/.)

XML 1.1

Tim Bray, et al. *Extensible Markup Language (XML) 1.1 (Second Edition)*, W3C Recommendation, 16 August 2006, edited in place 29 September 2006. (See http://www.w3.org/TR/2006/REC-xml11-20060816/.)

XML Base

Jonathan Marsh and Richard Tobin, Eds., *XML Base (Second Edition)*, W3C Recommendation, 28 January 2009. (See http://www.w3.org/TR/2009/REC-xmlbase-20090128/.)

XML ID

Jonathan Marsh, Daniel Veillard, Norman Walsh, Eds., *xml:id Version 1.0*, W3C Recommendation, 09 September 2005. (See http://www.w3.org/TR/2005/REC-xml-id-20050909/.)

XML InfoSet

John Cowan and Richard Tobin, Eds., *XML Information Set (Second Edition)*, W3C Recommendation, 04 February 2004. (See http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.)

XML Media Types

Makato Murata, Simon St. Laurent, Kan Khon, Eds., *XML Media Types*, RFC 3023, January 2001, IETF.(See http://www.rfc-editor.org/rfc/rfc3023.txt.)

XML Namespaces 1.0

Tim Bray, et al. *Namespaces in XML 1.0 (Third Edition)*, W3C Recommendation, 8 December 2009. (See http://www.w3.org/TR/2009/REC-xml-names-20091208/.)

XML Schema Part 1

Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, Eds., *XML Schema Part 1: Structures*, W3C Recommendation, 28 October 2004. (See http://www.w3.org/TR/xmlschema-1/.)

XML Schema Part 2

Paul Biron and Ashok Malhotra, *XML Schema Part 2: Datatypes*, W3C Recommendation, 28 October 2004. (See http://www.w3.org/TR/xmlschema-2/.)

XSL 1.1

Anders Berglund, *Extensible Stylesheet Language (XSL) Version 1.1*, W3C Recommendation, 05 December 2006. (See http://www.w3.org/TR/2006/REC-xsl11-20061205/.)

H Other References (Non-Normative)

CC-DECODER-REQ

Closed Caption Decoder Requirements for Analog Television Receivers, United States Code of Federal Regulations, Title 47, Volume 1, Part 79, § 79.101. (See http://www.gpo.gov/fdsys/pkg/CFR-2012-title47-vol4/pdf/CFR-2012-title47-vol4-sec79-101.pdf.)

CEA-608-E

EIA/CEA-608-E, *Line 21 Data Services*, EIA/CEA Standard, Consumer Electronics Association (CEA).

CEA-708-E

CEA-708-E, *Digital Television (DTV) Closed Captioning*, CEA Standard, Consumer Electronics Association (CEA).

CSS2

Bert Bos et al., *Cascading Style Sheets, Level 2 Revision 1*, W3C Recommendation, 07 June 2011. (See http://www.w3.org/TR/2011/REC-CSS2-20110607/.)

MPEG DASH
ISO/IEC 23009-1, *Information technology – Dynamic adaptive streaming over HTTP (DASH)*, International Organization for Standardization (ISO).

DCMES 1.1
Dublin Core Metadata Initiative, *Dublin Core Metadata Element Set, Version 1.1: Reference Description*. (See http://dublincore.org/documents/dces/.)

EBU-TT Live
EBU Tech3370, *EBU-TT Part 3 Live Subtitling*, European Broadcasting Union (EBU)

ISOBMFF
ISO/IEC 14496-30, *Information technology – Coding of audio-visual objects – Part 30: Timed text and other visual overlays in ISO base media file format*, International Organization for Standardization (ISO).

MPEG7-5
ISO/IEC 15938-5, *Information technology – Multimedia content description interface – Part 5: Multimedia description schemes*, International Organization for Standardization (ISO).

NSState
Norman Walsh, Ed., *The Disposition of Names in an XML Namespace*, TAG Finding, 9 January 2006. (See http://www.w3.org/2001/tag/doc/namespaceState-2006-01-09.)

QAF SG
Karl Dubost, et al., *QA Framework: Specifications Guidelines*, W3C Recommendation, 17 August 2005. (See http://www.w3.org/TR/qaframe-spec/.)

SMPTE 170M
ANSI/SMPTE 170M, *Television – Composite Analog Video Signal – NTSC for Studio Applications*, SMPTE Standard.

SMPTE 2052-11
SMPTE 2052-11, *Conversion from CEA-708 Caption Data to SMPTE-TT*, SMPTE Recommended Practice.

SVG 1.1
Jon Ferraiolo, Jun Fujisawa, Dean Jackson, Eds., *Scalable Vector Graphics (SVG) 1.1 Specification*, W3C Recommendation, 14 January 2003. (See http://www.w3.org/TR/2003/REC-SVG11-20030114/.)

TTAF1-REQ
Glenn Adams, Ed., *Timed Text (TT) Authoring Format 1.0 Use Cases and Requirements*, W3C Working Group Note, 27 April 2006. (See http://www.w3.org/TR/2006/NOTE-ttaf1-req-20060427/.)

TTML10
Glenn Adams, Ed., *Timed Text Markup Language (TTML) 1.0*, W3C Recommendation, 18 November 2010. (See http://www.w3.org/TR/2010/REC-ttaf1-dfxp-20101118/.)

XHTML 1.0
Steven Pemberton, et al., *XHTML™ 1.0, The Extensible Hypertext Markup Language (Second Edition)*, W3C Recommendation, 01 August 2002. (See http://www.w3.org/TR/2002/REC-xhtml1-20020801/.)

I Requirements (Non-Normative)

The Timed Text Markup Language (TTML), also known as the Distribution Format Exchange Profile (DFXP) of the Timed Text Authoring Format (TTAF), satisfies a subset of the requirements established by [TTAF1-REQ]. The following table enumerates these requirements and indicates the extent to which they are satisfied by this specification, where *S* denotes a requirement is satisfied, *P* denotes a requirement is partially satisfied, and *N* denotes a requirement is not satisfied.

Table I-1 – Requirement Satisfaction

ID	Name	Status	Comments
R100	Specification Format	S	
R101	Specification Modularity	S	

R102	Specification Organization	S	
R103	Core and Periphery	S	TT extension namespaces
R104	Evolution of Core	S	TT extension namespaces
R105	Ownership of Core	S	TT namespaces
R106	Surjection of Core	S	
R107	Evolution of Periphery	S	TT extension namespaces
R108	Ownership of Periphery	S	Non-TT namespaces
R109	Transformation	S	Supports 3GPP, QText, RealText, SAMI
R110	Streamable Transformation	S	Progressive decoding
R111	Accessibility – Content	S	Alternative document instances
R112	Accessibility – Authoring System	S	
R200	Authorability	S	
R201	Multiple Natural Languages	S	Alternative document instances
R202	Natural Language Coverage	S	Unicode 4.0
R203	Natural Language Association Granularity	S	See `xml:lang`
R204	Minimum Character Representability	S	Unicode 4.0
R205	Intrinsic and Extrinsic Text Content	P	Intrinsic only
R206	Markup Association	P	Intrinsic only
R207	Conditional Content	N	
R208	Flowed Text	S	
R209	Logical Flowed Text Vocabulary	S	
R210	Presentational Flowed Text Vocabulary	S	Implied mapping from logical flowed text.
R211	Flowed Text Vocabulary Relationship	S	
R212	Flowed Text Vocabulary Separation	N	
R213	Non-Flowed Text	N	
R214	Non-Flowed Text Vocabulary	N	
R215	Hybrid Flowed and Non-Flowed Text	N	
R216	Hyperlinking	N	Can support via XLink
R217	Embedded Graphics	N	
R218	Non-Embedded Graphics	N	
R219	Embedded Fonts	N	
R220	Non-Embedded Fonts	N	
R221	Descriptive Vocabulary	S	See `ttm:agent`, `ttm:role`
R222	Embedded Audio	N	
R223	Non-Embedded Audio	N	
R290	Markup Format	S	
R291	Markup Format and Unicode Interaction	S	
R292	Extrinsic Resource References	N	No extrinsic references
R293	Schema Validity Specification	S	
R300	Inline Styling	S	
R301	Inline Styling Form	P	Inline and referential styling
R301	Out-of-Line Styling	N	
R301	Out-of-Line Styling Form	N	
R304	Styling Prioritization	S	
R305	Style Parameters – Aural	N	
R306	Style Parameters – Visual	P	Supports absolute position, background color, color, display none, display alignment, font family, font size, font style, font weight, height, line height, origin, opacity, overflow,

			padding (before, after, start, end), text alignment, text shadow (as outline), visibility, width, writing mode, z-index
R390	Style Parameter Symmetry	S	
R391	Style Parameter Definitions	S	
R392	Style Parameter Shorthands	S	
R401	Inline Timing	S	
R402	Out-of-Line Timing	N	
R403	Synchronization Parameters	P	Supports begin, end, dur
R404	Synchronization Parameter Value Spaces	P	Supports offset values, media marker values (SMPTE 12M), wall-clock values
R405	Time Containment Semantics	P	Supports sequential, parallel
R500	Animation Modes	P	Supports discrete
R502	Highlight Animation	S	`<set tts:backgroundColor="..."/>`
R503	Fade Transition Animation	S	`<set tts:opacity="..."/>`
R504	Animated Style Parameters – Aural	N	
R505	Animated Style Parameters – Visual	P	Supports animating background color, color, display, opacity, origin, visibility
N506	Animated Content	S	
R600	Metadata Item Association	S	See `metadata`, `Metadata.class`
R601	Metadata Item Constituents	P	Supports name, value
R602	Metadata Item Value Representation	P	See `metadata`
R603	Metadata Item Extensibility	S	See `metadata`
R604	Metadata Item Validation	S	See `metadata`
R690	Dublin Core Preference	N	Uses `ttm:copyright`, `ttm:desc`, `ttm:title`

J Vocabulary Derivation (Non-Normative)

This appendix provides information about the derivation of TTML vocabulary, separately describing derivation of elements and attributes.

J.1 Element Derivation

The first column of **Table J-1 – Elements** specifies a TTML element vocabulary item; the second column specifies the syntactic and/or semantic model on which the vocabulary item is based; the third column specifies the reference that defines the model (if a model is indicated); the fourth column specifies details about the derivation; the last column refers to additional notes describing the nature of the derivation.

In the fourth column, which describes details of derivation, a notation is use to indicate the addition or removal of an attribute. For example, in the derivation of the `tt:div` element, the details column includes "-@class", which denotes that the `class` attribute that is specified for use with the `xhtml:div` model element is not specified for use with the corresponding TTML element; in contrast, the details column includes "+@begin", which denotes that a `begin` attribute is added that is not specified for use with the `xhtml:div` model element.

Table J-1 – Elements

Element	**Model**	**Reference**	**Details**	**Notes**
`tt:body`	`xhtml:body`	[XHTML 1.0]	-@class, -@dir, -@lang, -@on*, -@title; +@begin, +@dur, +@end, +@region, +@timeContainer, +@ttm:*, +@tts:*; content model subsetted to zero or more	1,2

			division (div) children, and supersetted by optional metadata and animation children	
tt:br	xhtml:br	[XHTML 1.0]	-@class, -@title; +@ttm:*, +@tts:*, +@xml:lang, +@xml:space; content model supersetted by optional metadata and animation children for congruity with other content vocabulary	1,2
tt:div	xhtml:div	[XHTML 1.0]	-@class, -@dir, -@lang, -@on*, -@title; +@begin, +@dur, +@end, +@region, +@timeContainer, +@ttm:*, +@tts:*, +@xml:space; content model subsetted to zero or more paragraph (p) children, and supersetted by optional metadata and animation children	1,2,3
tt:head	xhtml:head	[XHTML 1.0]	-@dir, -@lang, -@profile; +@id, +@xml:space; content model changed to optional metadata children, followed by optional styling child, followed by optional layout child	1,3
tt:layout	fo:simple-page-master	[XSL 1.1]	conceptual derivation	4
tt:metadata	svg:metadata	[SVG 1.1]	-@xml:base; +@ttm:*, +@xml:lang, +@xml:space; content model subsetted to foreign namespace element content only (no #PCDATA)	3,5
tt:p	xhtml:p	[XHTML 1.0]	-@class, -@dir, -@lang, -@on*, -@title; +@begin, +@dur, +@end, +@region, +@timeContainer, +@ttm:*, +@tts:*, +@xml:space; content model subsetted to zero or more span children, and supersetted by optional metadata and animation children	1,2,3
tt:region	fo:region-*	[XSL 1.1]	conceptual derivation	4
tt:set	svg:set	[SVG 1.1]	-@* except begin, dur, end; +@tts:*, +@xml:lang, +@xml:space	3,6
tt:span	xhtml:span	[XHTML 1.0]	-@class, -@dir, -@lang, -@on*, -@title; +@begin, +@dur, +@end, +@region, +@timeContainer, +@ttm:*, +@tts:*, +@xml:space; content model subsetted to zero or more #PCDATA or break (br) children, and supersetted by optional metadata and animation children	1,2,3
tt:style	*style specification*	[CSS2]	XML representation of identified set of pairs of style property name and value, with optional inclusion of other styles by reference to other style elements	7
tt:styling	xhtml:style	[XHTML 1.0]	XML representation of a set of style specifications sets, each represented by a style child element	1,7
tt:tt	xhtml:html	[XHTML 1.0]	-@dir, -@lang; +@id, +@ttp:*, +@xml:space; content model subsetted by permitting body and/or head to be	1,8

			optional	
`ttm:actor`	`mpeg7:Creator`	[MPEG7-5]	conceptual derivation	4
`ttm:agent`	`mpeg7:Agent`	[MPEG7-5]	conceptual derivation	4
`ttm:copyright`	`mpeg7:CopyrightString`	[MPEG7-5]	conceptual derivation	4
`ttm:desc`	`svg:desc`	[SVG 1.1]	-@class, -@style, -@xml:base	2,5,9
`ttm:name`	`mpeg7:Name`	[MPEG7-5]	conceptual derivation	4
`ttm:title`	`svg:title`	[SVG 1.1]	-@class, -@style, -@xml:base	2,5,9
`ttp:extension`	`@requiredExtensions`	[SVG 1.1]	conceptual derivation	10
`ttp:extensions`	`@requiredExtensions`	[SVG 1.1]	conceptual derivation	10
`ttp:feature`	`@requiredFeatures`	[SVG 1.1]	conceptual derivation	10
`ttp:features`	`@requiredFeatures`	[SVG 1.1]	conceptual derivation	10
`ttp:profile`	`@baseProfile`	[SVG 1.1]	conceptual derivation	11

Note:

1. Derivation is indicated with respect to the strict DTD defined by [XHTML 1.0], §A.1.
2. The `class` attribute is effectively replaced by the `style` attribute, which, instead of specifying an inline style, refers indirectly to one or more `style` elements that define a set of style specification sets.
3. The `xml:lang` and `xml:space` attributes are defined for all element types in order to support their inheritance semantics to operate in the context of foreign namespace elements.
4. Derivation is conceptual (notional) only.
5. The `xml:base` attribute is not used since there are no external references from core vocabulary.
6. The `attributeName` and `to` attributes of `svg:set` are replaced by the direct expression of the target attribute name and value by use of a `tts:*` attribute.
7. CSS style specification syntax is mapped to XML by use of attributes defined in the TT Style Namespace.
8. The `xml:id` attribute is defined for use on all element types.
9. The `style` attribute is supported only on Content elements.
10. Derived from the use of `@requiredExtensions` and `@requiredFeatures` on the `svg:svg` element, but extended to support distinct specification of optionality.
11. Derived from the use of `@baseProfile` and `@version` on the `svg:svg` element.

J.2 Attribute Derivation

The first column of **Table J-2 – Attributes** specifies a TTML attribute vocabulary item; the second column specifies the syntactic and/or semantic model on which the vocabulary item is based; the third column specifies the reference that defines the model (if a model is indicated); the fourth column specifies details about the derivation; the last column refers to additional notes describing the nature of the derivation.

In the fourth column, which describes details of derivation, a notation is use to indicate the addition or removal of an attribute value. For example, in the derivation of the `timeContainer` attribute, the details column includes "-excl", which denotes that the `excl` value that is specified for use with the `timeContainer` model attribute is not specified for use with the corresponding TTML attribute; similarly, an "+*value*" in the details column indicates that the attribute's values have been extended to include *value*.

Only those attributes that are specified for use on more than one TTML element type are listed below. Those per-element namespace attributes that are uniquely defined for a specific TTML element type are not listed below, but are considered to be part of the specific element type's derivation described in **Table J-1 – Elements** above.

Table J-2 – Attributes

Attribute	Model	Reference	Details	Notes
`begin`	`begin`	[SMIL 2.1]	see notes	2,3,4
`dur`	`dur`	[SMIL 2.1]	see notes	2,3,4
`end`	`end`	[SMIL 2.1]	see notes	2,3,4
`region`	`master-reference`	[XSL 1.1]	conceptual derivation	
`style`	`class`	[CSS2]	dereferences style specification(s) directly	
`timeContainer`	`timeContainer`	[SMIL 2.1]	-excl, -none; no default attribute value	5
`ttm:agent`	*none*		used to attribute agent of content	
`ttm:role`	*none*		used to attribute role of content	
`ttp:cellResolution`	*none*		expresses uniform grid resolution for cell based coordinates	
`ttp:clockMode`	*none*		determines how to interpret time expressions	
`ttp:frameRate`	*none*		expresses integral frame rate	
`ttp:frameRateMultiplier`	*none*		used to express non-integral, rational frame rates	
`ttp:markerMode`	*none*		expresses marker continuity semantics	
`ttp:pixelAspectRatio`	*none*		expresses pixel aspect ratio of related media	
`ttp:profile`	*none*		expresses profile of TTML used by a *Document Instance*	
`ttp:dropMode`	*none*		expresses frame counting (drop) modes	
`ttp:subFrameRate`	*none*		expresses sub-frame rate	
`ttp:tickRate`	*none*		used to interpret tick based time expressions	
`ttp:timeBase`	*none*		used to interpret semantics of time expressions	
`tts:backgroundColor`	`background-color`	[XSL 1.1]	`-inherit`	1,6
`tts:color`	`color`	[XSL 1.1]	`-inherit`	6
`tts:direction`	`direction`	[XSL 1.1]	`-inherit`	
`tts:display`	`display`	[CSS2]	only `auto`, `none`	
`tts:displayAlign`	`display-align`	[XSL 1.1]	`-inherit`	1
`tts:extent`	`width`, `height`	[XSL 1.1]	shorthand property	
`tts:fontFamily`	`font-family`	[XSL 1.1]	`-inherit`, extends generic family names	1
`tts:fontSize`	`font-size`	[XSL 1.1]	`-inherit`	1,7
`tts:fontStyle`	`font-style`	[XSL 1.1]	`-inherit`, `-backslant`	1
`tts:fontWeight`	`font-weight`	[XSL 1.1]	`-inherit`, `-bolder`, `-lighter`, `-<number>`	1
`tts:lineHeight`	`line-height`	[XSL 1.1]	`-inherit`, `-<number>`, `-<space>`	1
`tts:opacity`	`opacity`	[CSS3 Color]	`-inherit`	
`tts:origin`	`top`, `left`	[XSL 1.1]	shorthand property	
`tts:overflow`	`overflow`	[XSL 1.1]	`-inherit`, `-auto`, `-error-if-overflow`	
`tts:padding`	`padding`	[XSL 1.1]	`-inherit`	8
`tts:showBackground`	`showBackground`	[SMIL 2.1]	`-inherit`	

tts:textAlign	text-align	[XSL 1.1]	-inherit	1
tts:textDecoration	text-decoration	[XSL 1.1]	-inherit	1,9,12
tts:textOutline	text-shadow	[XSL 1.1]	-inherit	10,12
tts:unicodeBidi	unicode-bidi	[XSL 1.1]	-inherit	1
tts:visibility	visibility	[XSL 1.1]	-inherit, -collapse	
tts:wrapOption	wrap-option	[XSL 1.1]	-inherit	1
tts:writingMode	writing-mode	[XSL 1.1]	-inherit, +tblr	1
tts:zIndex	z-index	[XSL 1.1]	-inherit	1
xml:id	xml:id	[XML ID]	complies with model	
xml:lang	xml:lang	[XML 1.0]	complies with model	
xml:space	xml:space	[XML 1.0]	see notes	11

Note:

1. Attribute name and/or value(s) are normalized to use *lowerCamelCase* naming convention.
2. Restricted to expressing a clock value that denotes one of the following in accordance to whether the parameter expressed by the ttp:timeBase attribute is media, smpte, or clock, respectively: (1) an offset from an implicit syncbase that is linked to a media time line, (2) an event time that represents the occurrence of an implicit media marker, or (3) a wall-clock time.
3. Syntactically subsets and supersets the [SMIL 2.1] Clock-value syntax as follows: (1) requires non-negative Full-clock-value or Timecount-value; (2) if Full-clock-value then *hours* must be two or more digits; (3) if Timecount-value, then *metric* must be specified; (4) uses m as alias for min metric to denote minutes; (5) adds f and t metrics denoting frames and ticks, respectively; (6) adds alternative expression of optional Fraction in Full-clock-value by specifying frame count or frame count with subframe count.
4. Interpretation of time expression is further constrained by parameters expressed by ttp:clockMode, ttp:dropMode, ttp:frameRate, ttp:frameRateMultiplier, ttp:markerMode, ttp:subFrameRate, ttp:tickRate, and ttp:timeBase attributes.
5. Uses subset of named colors from model to which two aliases are added as follows: magenta as fuchsia, and cyan as aqua.
6. If not specified, then parallel (par) container semantics apply to the element types specified by **10.2.4 timeContainer**.
7. Restricts size to length specification which can be a percentage; adds optional second length (or percentage) for specifying separate horizontal and vertical scaling of glyph's EM square.
8. Expressed in terms of writing mode relative padding properties rather than absolute padding properties.
9. Excludes blink and no-blink values.
10. Uses only one length specification instead of two, where one length defines distance of outline effect from nominal edge of glyph contour outline perpendicular to point of glyph contour. Percentage lengths are also added to express outline effect in relative to font size. Outline effects are intended to be drawn both outside of outer closed contours and inside of inner closed contours.
11. On root element, default attribute value specified as default, which is defined in terms of whitespace normalization. Semantics of preservation and default normalization are defined in terms of presentation semantics by **7.2.3 xml:space**.
12. Defined to be inheritable.

K QA Framework Compliance (Non-Normative)

This appendix specifies the compliance of this specification with the requirements and guidelines defined by QA Framework Specifications Guidelines [QAF SG].

K.1 Requirements

Table K-1 – QA Framework Requirements Checklist

Requirement	YES	NO	N/A	Notes
Requirement 01: Include a conformance clause	YES			
Requirement 02: Define the scope.	YES			
Requirement 03: Identify who or what will implement the specification.	YES			
Requirement 04: Make a list of normative references.	YES			
Requirement 05: Define the terms used in the normative parts of the specification.	YES			
Requirement 06: Create conformance labels for each part of the conformance model.	YES			
Requirement 07: Use a consistent style for conformance requirements and explain how to distinguish them.	YES			
Requirement 08: Indicate which conformance requirements are mandatory, which are recommended, and which are optional.	YES			
Requirement 09: If the technology is subdivided, then indicate which subdivisions are mandatory for conformance.	YES			
Requirement 10: If the technology is subdivided, then address subdivision constraints.	YES			
Requirement 11: Address Extensibility.	YES			
Requirement 12: Identify deprecated features.			N/A	1
Requirement 13: Define how each class of product handles each deprecated feature.			N/A	1

> **Note:**
>
> 1. No feature is deprecated by this version of this specification.

K.2 Guidelines

Table K-2 – QA Framework Guidelines Checklist

Guideline	YES	NO	N/A	Notes
Good Practice 01: Define the specification's conformance model in the conformance clause.	YES			
Good Practice 02: Specify in the conformance clause how to distinguish normative from informative content.	YES			
Good Practice 03: Provide the wording for conformance claims.	YES			
Good Practice 04: Provide an Implementation Conformance Statement Pro Forma.		NO		
Good Practice 05: Require an Implementation Conformance Statement as part of valid conformance claims.	YES			
Good Practice 06: Provide examples, use cases, and graphics.	YES			
Good Practice 07: Write sample code or tests.	YES			
Good Practice 08: When imposing requirements by normative references, address conformance dependencies.	YES			1
Good Practice 09: Define unfamiliar terms in-line and consolidate the definitions in a glossary section.	YES			
Good Practice 10: Use terms already defined without changing their definition.	YES			2
Good Practice 11: Use formal languages when possible.	YES			

Good Practice 12: Write Test Assertions.		NO		3
Good Practice 13: Create subdivisions of the technology when warranted.	YES			
Good Practice 14: If the technology is profiled, define rules for creating new profiles.	YES			
Good Practice 15:Use optional features as warranted.	YES			
Good Practice 16: Clearly identify optional features.	YES			
Good Practice 17: Indicate any limitations or constraints on optional features.	YES			
Good Practice 18: If extensibility is allowed, define an extension mechanism.	YES			
Good Practice 19: Warn extension creators to create extensions that do not interfere with conformance.	YES			
Good Practice 20: Define error-handling for unknown extensions.	YES			4
Good Practice 21: Explain how to avoid using a deprecated feature.			N/A	5
Good Practice 22: Identify obsolete features.			N/A	5
Good Practice 23: Define an error handling mechanism.	YES			

Note:

1. When making normative references to external specifications, specific clauses or sections are cited.
2. See also **J Vocabulary Derivation**.
3. Test assertions and test suites will be provided prior to entering Proposed Recommendation (PR) phase.
4. See criterion #3 in **3.2 Processor Conformance** and definition of TTML Abstract Document Instance.
5. No feature is deprecated or obsoleted by this version of this specification.

L Streaming TTML Content (Non-Normative)

TTML Content can be authored to meet the following characteristics that may be useful in streaming scenarios:

- can be progressively encoded (i.e., does not require computing subsequent data prior to sending current data);
- can be progressively decoded (if it does not require forward references, but uses only reverse references when necessary);
- does not require dereferencing (and subsequent loading) of any resources other than TTML Content;
- has timing structure compiled into inline format that makes possible a temporal ordering of content that follows temporal presentation order;

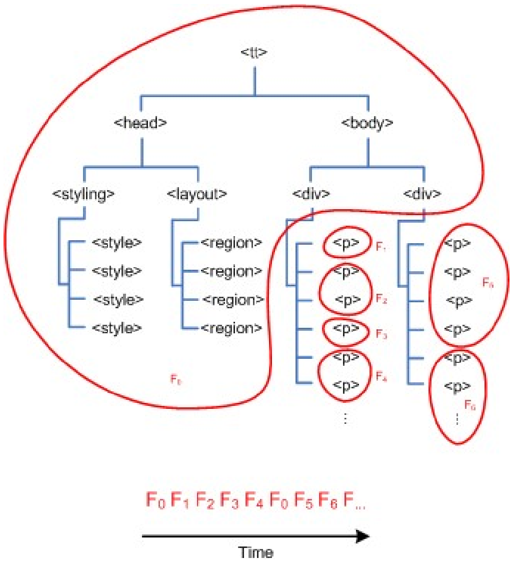
L.1 Root and Branch Fragmentation

One possible means by which TTML Content may be streamed is to partition a *Document Instance*'s information set into non-overlapping fragments, where one particular fragment, call it the *root fragment*, represents the front matter (head) of the *Document Instance* as well as its top level structural elements, and other fragments represent content whose time intervals are expected to be active in parallel.

In applications that require arbitrary (random) entry into a stream, i.e., the property of being able to start reading data at an arbitrary data access unit, the root fragment will be repetitively transmitted (inserted) into the stream in order to permit a decoder to resynchronize and acquire sufficient structural information in the information set in order to interpret subsequent content fragments.

An example of such a fragmentation of a *Document Instance* is shown in **Figure 3 – Fragment Streaming**.

Figure 3 – Fragment Streaming



> **Note:**
>
> This specification does not define a transport buffer model or a decoder capabilities model.

L.2 Temporal Fragmentation

Another means by which TTML Content may be streamed is to partition a *Document Instance*'s information set into temporally bound fragments, each of which is itself a document instance that contains all the front matter (head) and content required to present it, where the temporal interval for each fragment is constrained, either by the *Document Interchange Context* or by the *Document Processing Context* or by both, such that temporal overlaps with other fragments are resolved.

For example, some document processing contexts resolve temporal overlap by specifying that a maximum of one document instance can be active at any moment in the timeline, and additionally specify precedence rules to establish which document instance that is, in case there are multiple candidates.

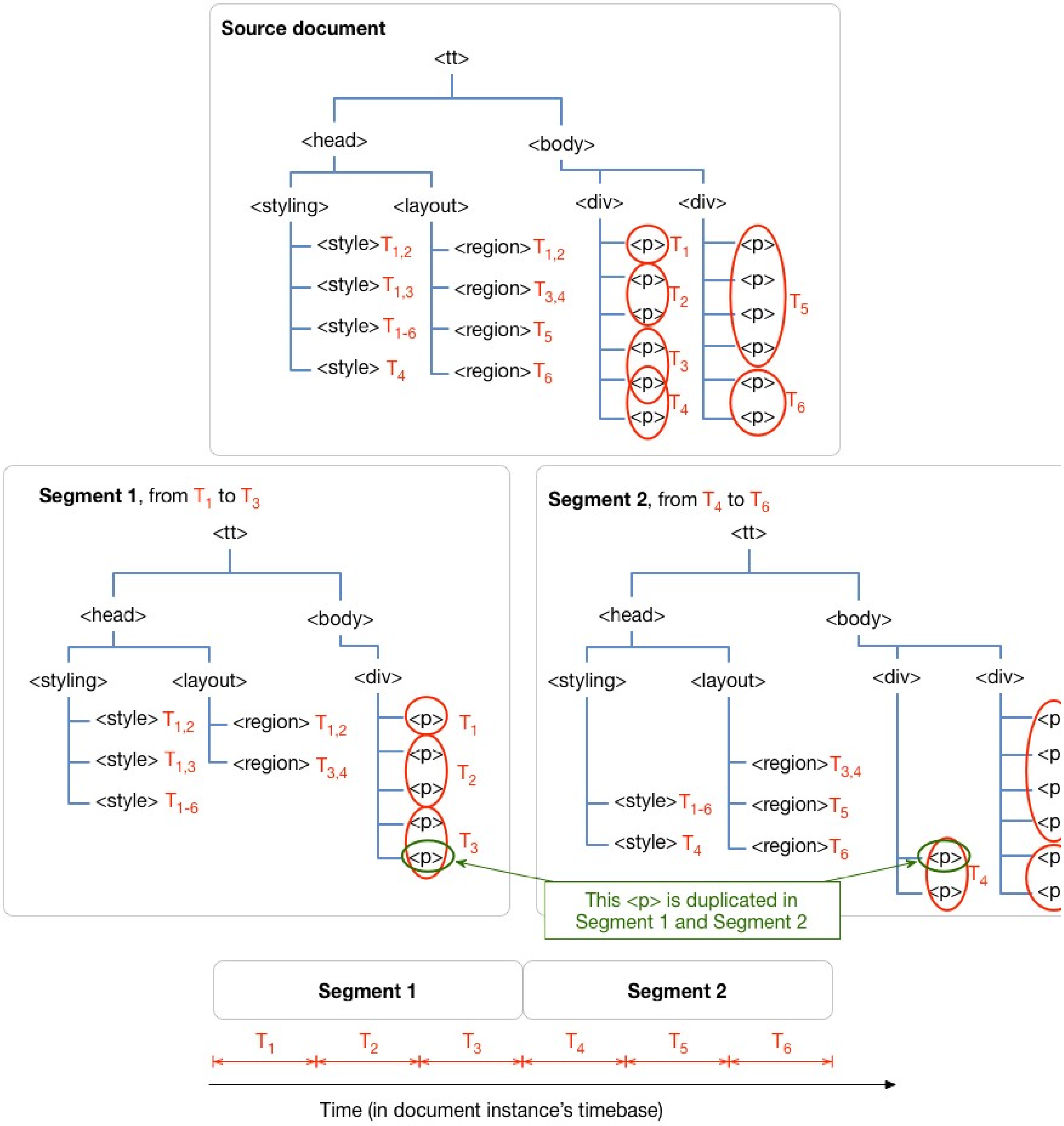
In order to ensure that all the content is correctly displayed, any content that falls into an Intermediate Synchronic Document (ISD) whose interval overlaps with more than one fragmentation interval might need to be duplicated within each fragment that is required to generate that ISD. It is possible for implementations to identify adjacent ISDs that are identical except in their intervals and replace them with a single ISD covering the combined interval.

The intent of this approach is that presentation processors are able to generate an equivalent sequence of ISDs to that which would be generated by a non-streamed version of the document instance, while taking advantage of the ability to avoid duplicating some controllable amount of front matter, by varying the segment duration. Reconstruction of the original source is not guaranteed to result in an identical document instance.

> **Note:**
>
> This technique is used for example in [ISOBMFF] and in [EBU-TT Live].

An example of such a fragmentation of a *Document Instance* is shown in **Figure 4 – Temporal fragmentation**.

Figure 4 – Temporal fragmentation



Note:

In the above example, a single document instance has content visible during intervals T_1 through T_6 and is fragmented into two document instances (segments), the first including all content and referenced styles and regions required during T_1 through T_3, the second including all content and referenced styles and regions required during T_4 through T_6.

M Concrete Encoding (Non-Normative)

In the absence of other requirements, it is recommended that a *Document Instance* be concretely encoded as a well-formed XML 1.0 [XML 1.0] document using the UTF-8 character encoding.

> **Note:**
>
> When using XML 1.0 [XML 1.0] as the concrete encoding of TTML, only the following named character entities are defined: `&amp;`, `&apos;`, `&lt;`, `&gt;`, and `&quot;`.

N Time Expression Semantics (Non-Normative)

This appendix describes the intended semantics for interpreting time expressions in *Document Instances*.

> **Note:**
>
> It is expected that the information in this appendix will be elevated to normative status in a future revision of this specification.

> **Note:**
>
> The phrase *local real time* as used below is intended to model a virtual real time clock in the document processing context, where *local* means in the immediate proximity of the implementation of this processing context. The intent of defining relationships with this virtual clock is to establish a locally valid physical realization of time for didactic purposes.

> **Note:**
>
> The phrase *play rate* as used below is intended to model a (possibly variable) parameter in the document processing context wherein the rate of playback (or interpretation) of time may artificially dilated or narrowed, for example, when slowing down or speeding up the rate of playback of a *Related Media Object*. Without loss of generality, the following discussion assumes a fixed play(back) rate. In the case of variable play rates, appropriate adjustments may need to be made to the resulting computations.

The *Document Processing Context* defines the applicable epoch and any epoch-related offsets to be used when establishing the *Synthetic Document Syncbase*.

N.1 Clock Time Base

When operating with the `clock` time base, the following semantics should be applied for interpreting time expressions, as defined by <timeExpression>, and their relationship to media time and local real time.

The clock time base **`C`** is related to local real time **`R`** expressed in the epoch **`E`** (defined by the *Document Processing Context*) as follows:

TTML Semantics – Clock Time and Real Time Relationship

```
R = C + epochOffset + discontinuityOffset
```

where `C ∈ ℜ`, `0 ≤ C < ∞`, `C` in seconds since the most immediately prior midnight of the reference clock base;

`epochOffset ∈ ℜ`, `0 ≤ epochOffset < ∞`, `epochOffset` in seconds, with 0 being the beginning of epoch `E`, and where the value of `epochOffset` is determined from the computed value of the `ttp:clockMode` parameter as follows:

(1) if `local`, then the difference between the local real time at the most immediately prior local midnight and the local real time at the beginning of epoch `E`, expressed in seconds;

(2) if `gps`, then the difference between the GPS time at the most immediately prior GPS midnight and the GPS time at the beginning of epoch `E`, expressed in seconds;

(3) if `utc`, then the difference between the UTC time at the most immediately prior UTC midnight and the UTC time at the beginning of epoch `E`, expressed in seconds;

`discontinuityOffset ∈ ℜ`, `−∞ < discontinuityOffset < ∞`, `discontinuityOffset` in seconds, and where the value of `discontinuityOffset` is equal to the sum of leap seconds (and fractions thereof) that have been added (or subtracted) since the most immediately prior midnight in the reference clock base;

and `epochOffset` and `discontinuityOffset` are determined once and only once prior to the beginning of the *Root Temporal Extent* such that during the period between value determination and the beginning of the *Root Temporal Extent* there occurs no local midnight or reference clock base discontinuity.

Time value expressions, as denoted by a <timeExpression>, are related to clock time **`C`** as follows:

TTML Semantics – Time Expressions and Clock Time Relationship

If a time expression uses the *clock-time* form or an *offset-time* form that doesn't use the ticks (`t`) metric, then:

```
C = 3600 * hours + 60 * minutes + seconds
```

where `hours`, `minutes`, `seconds` components are extracted from time expression if present, or zero if not present. Furthermore, a *fraction* component, if present, is added to the *seconds* component to form a real-valued *seconds* component.

Otherwise, if a time expression uses an *offset-time* form that uses the ticks (`t`) metric, then:

```
C = ticks / tickRate
```

> **Note:**
>
> The *frames* and *sub-frames* terms and the frames (`f`) metric of time expressions do not apply when using the `clock` time base.

The clock time base **`C`** is independent of media time **`M`**:

TTML Semantics – Clock Time and Media Time Relationship

```
M ¬∝ C
```

Note:

That is to say, timing is disconnected from (not necessarily proportional to) media time when the `clock` time base is used. For example, if the media play rate is zero (0), media playback is suspended; however, timing coordinates will continue to advance according to the natural progression of clock time in direct proportion to the reference clock base. Furthermore, if the media play rate changes during playback, presentation timing is not affected.

N.2 Media Time Base

When operating with the `media` time base, the following semantics should be applied for interpreting time expressions, as defined by <timeExpression>, and their relationship to media time, document time, and local real time.

N.2.1 Relationship to Local Real Time

The media time base **`M`** is related to local real time **`R`** expressed in the epoch **`E`** (defined by the *Document Processing Context*) as follows:

TTML Semantics – Media Time and Real Time Relationship

```
R = playRate * M + epochOffset
```

where `M ∈ ℜ`, `0 ≤ M < ∞`, `M` in seconds, with 0 corresponding to the beginning of the *Root Temporal Extent*;

`playRate ∈ ℜ`, `-∞ < playRate < ∞`, `playRate` is unit-less, and where the value of `playRate` is determined by the document processing context;

and `epochOffset ∈ ℜ`, `0 ≤ epochOffset < ∞`, `epochOffset` in seconds, with 0 corresponding to the beginning of an epoch `E`, and where the value of `epochOffset` is the difference between the local real time at the beginning of the *Root Temporal Extent* and the local real time at the the beginning of epoch `E`, expressed in seconds.

N.2.2 Relationship to Media Time

Time value expressions, as denoted by a <timeExpression>, are related to media time **`M`** in accordance to the `ttp:frameRate`, `ttp:subFrameRate`, and `ttp:frameRateMultiplier` parameters as follows:

TTML Semantics – Time Expressions and Media Time Relationship

If a time expression uses a *clock-time* form or an *offset-time* form that doesn't use the ticks (`t`) metric, then:

```
M = referenceBegin + 3600 * hours + 60 * minutes + seconds + ((frames + (subFrames /
subFrameRate)) / effectiveFrameRate)
```

where `referenceBegin` is determined according to whether the nearest ancestor time container employs parallel (`par`) or sequential (`seq`) semantics: if parallel or if sequential and no prior sibling timed element exists, then `referenceBegin` is the media time that corresponds to the beginning of the nearest ancestor time container or zero (0) if this time container is the *Root Temporal Extent*; otherwise, if sequential and a prior sibling timed element exists, then `referenceBegin` is the media time that corresponds to the active end of the immediate prior sibling timed element;

the `hours`, `minutes`, `seconds`, `frames`, `subFrames` components are extracted from time expression if present, or zero if not present; furthermore, if the time expression takes the form of a *clock-time* expression, then the *fraction* component, if present, is added to the *seconds* component to form a real-valued *seconds* component, or, if the time expression takes the form of an *offset-time* expression, then the *fraction* component, if present, is added to the *time-count* component to form a real-valued *time-count* component according to the specified offset metric;

`subFrameRate` is the computed value of the `ttp:subFrameRate` parameter;

and `effectiveFrameRate` (in frames per second) is `frameRate * frameRateMultiplier` where `frameRate` is the computed value of the `ttp:frameRate` parameter and `frameRateMultiplier` is the computed value of the `ttp:frameRateMultiplier` parameter.

Otherwise, if a time expression uses an *offset-time* form that uses the ticks (`t`) metric, then:

```
M = referenceBegin + ticks / tickRate
```

where `referenceBegin` is as described above;

the `ticks` component is extracted from time expression;

and `tickRate` is the computed value of the `ttp:tickRate` parameter;

Note:

If the computed `frameRateMultiplier` ratio is not integral, then `effectiveFrameRate` will be a non-integral rational.

Note:

The above formalisms assumes that the *Root Temporal Extent* corresponds with the beginning of a related media object. If this assumption doesn't hold, then an additional offset that accounts for the difference may be introduced when computing media time `M`.

N.2.3 Relationship to Document Time

The computed document times are used as the equivalent media times with no offset.

> **Note:**
>
> Any additional media time processing imposed by e.g. a wrapper format is out of scope of the TTML specification.
>
> For example [ISOBMFF] and [MPEG DASH] provide mechanisms for wrapping TTML documents as samples or segments and defining the offset of their media timelines relative to other media such as audio or video.

The following document fragments are equivalent in timing:

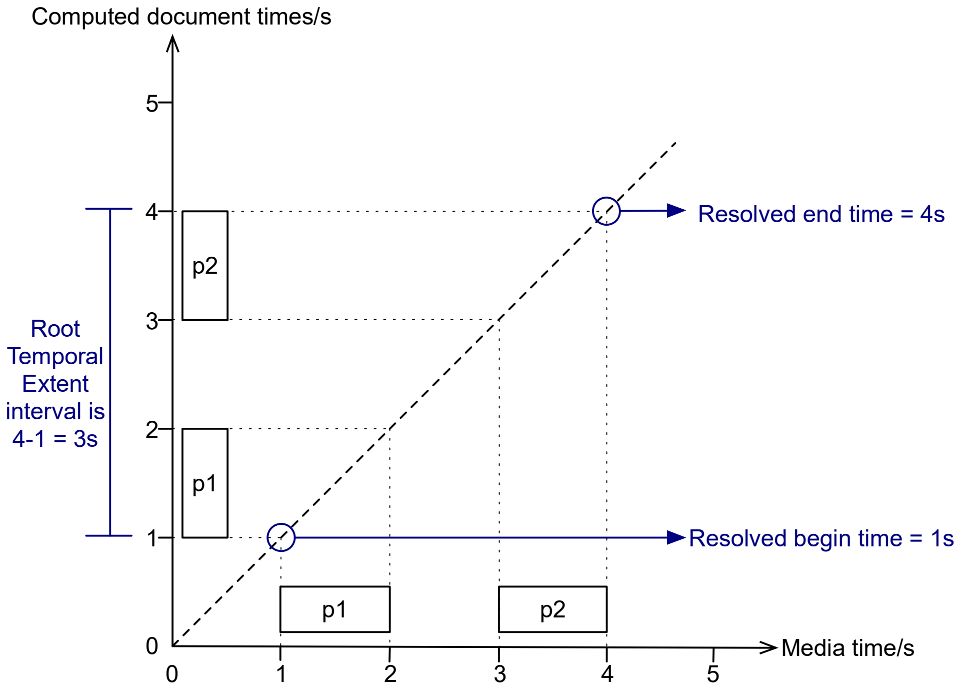
Example Fragment – 1

```
<tt ttp:timeBase="media" ...>
...
<body>
   <div xml:id="d1" begin="1s">
      <p xml:id="p1" begin="0s" end="1s">First paragraph</p>
      <p xml:id="p2" begin="2s" end="3s">Second paragraph</p>
   </div>
</body>
</tt>
```

Example Fragment – 2

```
<tt ttp:timeBase="media" ...>
...
<body>
   <div xml:id="d1">
      <p xml:id="p1" begin="1s" end="2s">First paragraph</p>
      <p xml:id="p2" begin="3s" end="4s">Second paragraph</p>
   </div>
</body>
</tt>
```

Timing diagram showing the mapping of document times to media times in the absence of other external information.

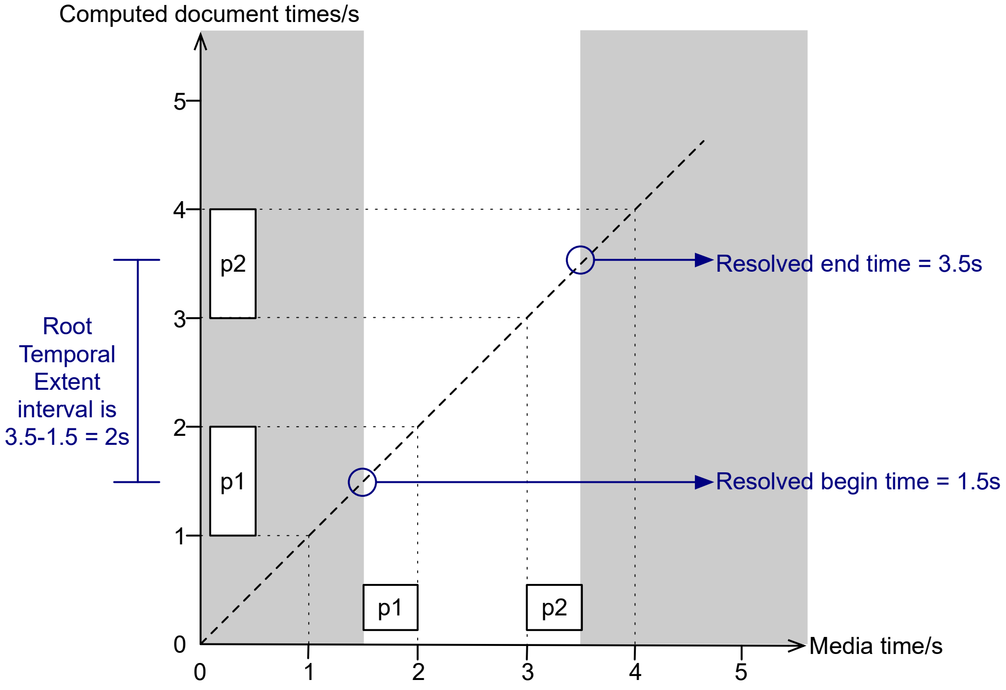


In this example, using the the terminology of [SMIL 2.1], the implicit duration of the body element is identical to its computed duration which is also the *Root Temporal Extent*.

However if the *Document Processing Context* specifies a range of applicable media times, those limit the resolved begin and end times and therefore the *Root Temporal Extent*.

In the following example the processing context defines that the media time range over which the document is active is from 1.5s to 3.5s, and this defines the Root Temporal Extent since timed content is present at both of those times.

Timing diagram showing the mapping of document times to media times when the media time range is constrained by the processing context.



The grey background indicates the periods when the *Document Processing Context* defines the document to be inactive. If the document processing context were to define that the media time range is 0-3s then the *Root Temporal Extent* would be from 1s to 2s as defined by p1 and p2 would never be shown.

N.3 SMPTE Time Base

When operating with the smpte time base, the following semantics should be applied for interpreting time expressions, as defined by <timeExpression>, and their relationship to media time and local real time.

If the computed value of the ttp:markerMode parameter is discontinuous, then there is no direct relationship between time expressions and media time M or local real time R. In this case, time expressions refer to synchronization events (markers) emitted by the *Document Processing Context* when *SMPTE Time Codes* are encountered in the *Related Media Object*.

Otherwise, if the computed value of the ttp:markerMode parameter is continuous, then the relationships between time expressions and local real time and media time are as described below in terms of a *Synthetic SMPTE Document Syncbase*, here referred to as the SMPTE time base **S**.

TTML Semantics – Time Expressions and SMPTE Time Relationship

> `S = (countedFrames - droppedFrames + (subFrames / subFrameRate)) / effectiveFrameRate`
>
> where
>
> `countedFrames = (3600 * hours + 60 * minutes + seconds) * frameRate + frames`
>
> `hours`, `minutes`, `seconds`, `frames`, `subFrames` components are extracted from time expression if present, or zero if not present;
>
> `droppedFrames` is computed as follows:
>
> 1. let *dropMode* be the computed value of the `ttp:dropMode` parameter;
>
> 2. if *dropMode* is `dropNTSC`, let `droppedFrames = (hours * 54 + minutes - floor(minutes/10)) * 2`;
>
> 3. otherwise, if *dropMode* is `dropPAL`, let `droppedFrames = (hours * 27 + floor(minutes/2) - floor(minutes/20)) * 4`;
>
> 4. otherwise, let `droppedFrames = 0`;
>
> `frameRate` is the computed value of the `ttp:frameRate` parameter;
>
> `subFrameRate` is the computed value of the `ttp:subFrameRate` parameter;
>
> and `effectiveFrameRate` (in frames per second) is `frameRate * frameRateMultiplier` where `frameRate` is the computed value of the `ttp:frameRate` parameter and `frameRateMultiplier` is the computed value of the `ttp:frameRateMultiplier` parameter.
>
> Notwithstanding the above, if a time expression contains a frame code that is designated as dropped according to **6.2.3 ttp:dropMode**, then that time expression should be considered to be invalid for purposes of validation assessment.

The SMPTE time base **`S`** is related to the media time base `M` as follows:

TTML Semantics – SMPTE Time and Media Time Relationship

> `M = referenceBegin + S`
>
> where `referenceBegin` is determined according to whether the nearest ancestor time container employs parallel (`par`) or sequential (`seq`) semantics: if parallel or if sequential and no prior sibling timed element exists, then `referenceBegin` is the SMPTE time that corresponds to the beginning of the nearest ancestor time container or zero (0) if this time container is the *Root Temporal Extent*; otherwise, if sequential and a prior sibling timed element exists, then `referenceBegin` is the SMPTE time that corresponds to the active end of the immediate prior sibling timed element;

Given the derived media time base as described above, then media time base **`M`** is related to the local real time **`R`** as described in **N.2 Media Time Base** above.

O Common Caption Style Examples (Non-Normative)

This section provides examples of the following common caption styles using TTML Content to obtain the desired behavior:

- Pop-On Captions
- Roll-Up Captions
- Paint-On Captions

O.1 Pop-On Caption Example

An example of paint-on captions. In this example, two regions are targeted with alternating, paint-on content, where content is timed using explicit sequential time containment rules. Each paragraph is non-overlapping in time and appears in the same single row of its targeted region.

Example – Pop-On Captions

```
<tt ttp:cellResolution="60 20" xml:lang="en" xmlns="http://www.w3.org/ns/ttml"
  xmlns:ttp="http://www.w3.org/ns/ttml#parameter" xmlns:tts="http://www.w3.org/ns/ttml#styling">
  <head>
    <layout>
      <region xml:id="r1" tts:color="white" tts:origin="10c 4c" tts:extent="40c 1c"/>
      <region xml:id="r2" tts:color="yellow" tts:origin="10c 8c" tts:extent="40c 1c"/>
    </layout>
  </head>
  <body>
    <div timeContainer="seq">
      <p region="r1" dur="4s">Lorem ipsum dolor sit</p>
      <p region="r2" dur="4s">Amet consectetur adipiscing elit</p>
      <p region="r1" dur="6s">Sed do eiusmod tempor incididunt labore</p>
      <p region="r2" dur="4s">et dolore magna aliqua</p>
      <p region="r1" dur="7s">Ut enim ad minim veniam quis, nostrud</p>
    </div>
  </body>
</tt>
```

O.2 Roll-Up Caption Example

An example of roll-up captions. Roll-up effects are achieved by using overlapped time intervals, where zero, one, or two paragraphs appear in the region at a given time. Each paragraph consumes a single row (line) of the region since no wrapping occurs. Depending on whether a presentation processor supports smooth scrolling between adjacent synchronic intermediate document instances, the transitions, i.e., moving an old paragraph (line) out and a new paragraph (line) in, will be either smooth or discrete.

Example – Roll-Up Captions

```
<tt ttp:cellResolution="60 20" xml:lang="en" xmlns="http://www.w3.org/ns/ttml"
  xmlns:ttp="http://www.w3.org/ns/ttml#parameter" xmlns:tts="http://www.w3.org/ns/ttml#styling">
  <head>
    <layout>
      <region xml:id="r1" tts:color="white" tts:origin="10c 4c" tts:extent="40c 2c" tts:displayAlign="after"/
    </layout>
  </head>
  <body>
    <div region="r1">
      <p begin="0s" end="8s">Lorem ipsum dolor sit</p>
      <p begin="4s" end="12s">Amet consectetur adipiscing elit</p>
      <p begin="8s" end="18s">Sed do eiusmod tempor incididunt labore</p>
      <p begin="14s" end="25s">et dolore magna aliqua</p>
      <p begin="18s" end="29s">Ut enim ad minim veniam quis, nostrud</p>
    </div>
  </body>
</tt>
```

O.3 Paint-On Caption Example

An example of paint-on captions. Paint-on effects are achieved by using timed `span` elements in order to expose (paint) inline text units, e.g., words, over some time interval. Here, five paragraphs have their individual words sequentially timed in order to paint one new word every second. The end of the active duration of each inline element coincides with the end of the `div` element's parallel time container, so that once a word is painted, it remains in the region (on its rendered line) until the `div` element's active time interval lapses.

Example – Paint-On Captions

```
<tt ttp:cellResolution="60 20" xml:lang="en" xmlns="http://www.w3.org/ns/ttml"
  xmlns:ttp="http://www.w3.org/ns/ttml#parameter" xmlns:tts="http://www.w3.org/ns/ttml#styling">
  <head>
    <layout>
      <region xml:id="r1" tts:color="white" tts:origin="10c 4c" tts:extent="40c 5c" tts:displayAlign="after"/
    </layout>
  </head>
  <body>
    <div region="r1" begin="0s" end="25s">
      <p>
        <span begin="0s">Lorem</span>
        <span begin="1s">ipsum</span>
        <span begin="2s">dolor</span>
        <span begin="3s">sit</span>
      </p>
      <p>
        <span begin="4s">Amet</span>
        <span begin="5s">consectetur</span>
        <span begin="6s">adipiscing</span>
        <span begin="7s">elit</span>
      </p>
      <p>
        <span begin="8s">Sed</span>
        <span begin="9s">do</span>
        <span begin="10s">eiusmod</span>
        <span begin="11s">tempor</span>
        <span begin="12s">incididunt </span>
        <span begin="13s">labore</span>
     </p>
     <p>
        <span begin="14s">et</span>
        <span begin="15s">dolore</span>
        <span begin="16s">magna</span>
        <span begin="17s">aliqua</span>
      </p>
      <p>
        <span begin="18s">Ut</span>
        <span begin="19s">enim</span>
        <span begin="20s">ad</span>
        <span begin="21s">minim</span>
        <span begin="22s">veniam</span>
        <span begin="23s">quis,</span>
        <span begin="24s">nostrud</span>
      </p>
    </div>
  </body>
</tt>
```

P Acknowledgments (Non-Normative)

The editor acknowledges the current and former members of the Timed Text Working Group, the members of other W3C Working Groups, and industry experts in other forums who have contributed directly or indirectly to the process or content of this document as follows:

Glenn Adams, Thomas Bause-Mason, John Birch, Kees Blom, Bert Bos, Brad Botkin, Dick Bulterman, Frans de Jong, Mike

Dolan, Martin Dürst, Donald Evans, Geoff Freed, Al Gilman, Giles Godart-Brown, Markus Gylling, Markku Hakkinen, Sean Hayes, Erik Hodge, Philipp Hoschka, Suzi Hyun, Michael Jordan, Masahiko Kaneko, Courtney Kennedy, George Kerscher, David Kirby, Andrew Kirkpatrick, Philippe Le Hégaret, Pierre-Anthony Lemieux, Chris Lilley, Jason Livingston, Monica Martin, Matthew May, Nigel Megitt, Thierry Michel, Frank Olivier, Soohong Daniel Park, Silvia Pfeiffer, Brian Raymor, David Ronca, Patrick Schmitz, David Singer, Craig Smithpeters, Andreas Tai, and Mohamed Zergaoui.

The editor wishes to especially acknowledge the following contributions by members: Micheal Dolan (SMPTE time codes, streaming; SMPTE liaison), David Kirby (introductory example document; SMPTE time codes, descriptive metadata; EBU/AAF liaison), Geoff Freed (styling and example images of style properties), Sean Hayes (advanced profile concepts, including applicative timing), Eric Hodge (timing), Thierry Michel (metadata), and Dave Singer (animation, scrolling).

The Working Group dedicates this specification to our colleague David Kirby.

↑