Text Layout Format 2.0 Specification

1. Contents

1. Introduction

The Text Layout Format (TLF) 2.0 specification defines a uniform way to define rich text and its markup. The top level element in TLF is TextFlow and its contents are the text, as Unicode characters, that can be marked-up with other elements and attributes described in this document.

An object hierarchy in a programming language like ActionScript can be created by reading in the serialized TLF XML called TLF markup. Of course, such an object hierarchy can be used to output serialized XML in TLF markup. Such a serialization is sometimes called a “metafile” for the object hierarchy.

Flash authoring tools allow extracting and saving a single text flow as TLF. Since TLF markup is also compatible with MXML, text can also be compiled into a SWF file directly using the MXML compiler.

Both FXG and Flash have the notion of a container, and “container” is part of the lexicon describing those languages. The concept is also useful for TLF as the root TextFlow element is always contained in something, and that “something” is formatted with properties to which some TLF attributes must refer. For example, a client laying out text will set width and height properties before it interprets formatting contained by the TextFlow element. Certain TLF markup, such as attributes with values of “start”, “end”, etc., references those container width and height properties to correctly layout the text. In the context of this specification then, “container” is simply the client-defined container for the text.

2. Scope

This specification defines only TLF 2.0. For information about TLF 1.0 or 1.1, see the TLF 1.1 Specification.

What’s new in TLF 2.0

This section describes what has been added between TLF 1.0 and TLF 2.0.

A short list of the features added or extended includes:

- Lists
- Floats
- Groups
- Spacing controls
- Box leading model
- Padding
- Version attribute

3. Data types

The common data types for TLF attributes fall into the following categories:

3.1. color

The basic type <color> is assumed to be a specification for a color in the sRGB color space SRGB. A <color> is a numerical RGB specification in hexadecimal notation. It is specified by a '#' immediately followed by six hexadecimal characters, like so: #RRGGBB.

3.2. integer

An <integer> is specified as an optional sign character ('+' or '-') followed by one or more digits "0" to "9". If the sign character is not present, the number is non-negative. Unless stated otherwise for a particular attribute, the range for an <integer> encompasses (at a minimum) -2147483648 to 2147483647.

3.3. boolean

A <boolean> can have one of two values: true or false. These values are used typically for logical operations.

3.4. identifier

A text string for the id property that matches the regular expression [A-Za-z][A-Za-z0-9_]*.

3.5. number (real number value)

A <number> is specified either in decimal notation, or in scientific notation. Decimal notation consists of either an <integer>, or an optional sign character followed by zero or more digits followed by a dot (.) followed by one or more digits. Scientific notation consists of a decimal number immediately followed by the letter "e" or "E" immediately followed by an <integer>.

Unless stated otherwise for a particular attribute, a <number> has the capacity for at least a single-precision floating point number (see ICC32) and has a range (at a minimum) of -3.4e+38F to +3.4e+38F.

3.6. percentage

The format of a percentage value is a <number> immediately followed by a '%'. Percentage values are always relative to another value; for example, a length. Each attribute or property that allows percentages also defines the reference measurement to which the percentage refers.

4. Elements

4.1. TextFlow element

The root element of TLF markup is a <TextFlow> element.

The contents of a <TextFlow> element are a string of Unicode text characters possibly marked up with one or more of the following child elements:

- div
- list
- p
- g
- tcy
- a
- img
- span
- br
- tab
- linkNormalFormat
- linkHoverFormat
- linkActiveFormat
- listMarkerFormat

4.1.1. The version attribute

The TextFlow has a special attribute to indicate what version of the Text Layout markup format is being used. For content in the TLF 2.0 format, the TextFlow element must always have a version attribute, and its value should be "2.0.0".

4.2. div element

The content of the <div> element is a string of Unicode text characters possibly marked up with one or more of the same child elements supported by the TextFlow element.

The <div> element is similar in function to the div tag in HTML. The main purpose of a <div> element is to group together text for applying attributes.

4.3. list element

The <list> element is similar to a <div>, but for the purpose of grouping items together in a numbered or unnumbered list. It is similar to an ol or ul tag in HTML.

A <list> element can have all the same children as a TextFlow, plus an <li> element.

4.4. listItem element

An <li> element is used to contain the content in a single member in a list.

An <li> element can have all the same children as a TextFlow.

4.5. paragraph element

The <p> element is used within <TextFlow> that has multiple paragraphs. A <p> starts a new paragraph. A <p> can be a child of a <TextFlow> element , <div>, <list> or <li>. Every <p> has at least one <span> that can be implied. Any character content of a <p> is the content of an implied <span>.

Elements allowed to markup the content characters of a <p> element are:

- tcy
- g
- a
- img
- span
- br
- tab
- linkNormalFormat
- linkHoverFormat
- linkActiveFormat

4.6. group element

The <g> element is used to group together elements in a paragraph. Its purpose is to allow nesting below the paragraph level. A single <g> element may include multiple nested <g> elements or spans, for example.

The content that may appear in a <g> element is the same as for a paragraph.

4.7. tcy element

The <tcy> element is used for holding text that is tate-chu-yoko; that is, horizontal text within a vertical text flow. Traditionally it is used to make small blocks of non–Japanese text or numbers, such as dates, more readable. <tcy> can be applied to horizontal text but has no effect on drawing style unless and until it is turned vertical.

<tcy> elements may not be nested so the following is illegal:

```
<tcy>
  TCY
    <a>LINK
      <tcy>NESTED TCY</tcy>
    </a>
</tcy>
```

The content of the <tcy> element is a string of Unicode text characters possibly marked up with one or more of the following child elements (Same as <p> except that it cannot contain another <tcy>):

- a
- g
- img
- span
- br
- tab
- linkNormalFormat
- linkHoverFormat
- linkActiveFormat

4.8. a element

The <a> element represents an anchor for a hyperlink used for associating a URI with text in a paragraph. A link element may have a uri (attribute), a set of rollover link formats (child tags), and a set of attributes used for indicating an anchor that has been activated.

Note that anchor elements do not nest, so the following is not valid:

```
<a>My
  <span>My
    <a>NESTEDLINK</a>
  </span>is here.
</a>
```

The content of the <a> element is a string of Unicode text characters possibly marked up with one or more of the following child elements:

- tcy
- g
- img
- span
- br
- tab
- linkNormalFormat
- linkHoverFormat
- linkActiveFormat

The link format tags (linkNormalFormat, linkHoverFormat, and linkActiveFormat as defined below) must be the first children in their containing element. Link formats must be defined before the anchor text spans (link formats must be parents of actual link text). Link formats are inherited. If formats have not been explicitly defined, the defaults are applied.

The <a> element is the only element that may contain the following attributes:

- **href <String>**: The URI associated with the text. The interpretation of the URI is always left to the client and is not automatically associated with the content file. It may be absolute or relative but to what is left up to the client. For example, in a Flash application it may also be a file relative to the SWF file.
- **target <String>**: The location where the target of the link will appear. If specified, target must be one of the following:
    - **_self** Replaces the current html page. If the anchor is in a frame or frameset, it will load the link target within that frame. If the anchor resides in a page opened in a browser, the link target opens to replace the html page it came from.
    - **_blank** Opens a new browser window with no name.
    - **_parent** Replaces the html page containing the anchor with the link target.
    - **_top** Loads in the current browser window by replacing anything within that browser window (such as a frameset).

4.9. img element

An <img> element is used for specifying a graphic that appears in the text. Images appear inline in the text as if they were a special character. An image may appear as the content of the following elements:

- Textflow
- div
- paragraph
- g
- a
- tcy

The <img> element contains no children.

The <img> element may contain the following attributes:

- **width<Number, Percent or "auto">**: The image width. The value specifies how much space is set aside for the image in the text. If a percent, it's a percentage of the measured or default width expressed as a number followed by a percent sign. A value of "auto" sets the width to the source image width.
- **height<Number, Percent or "auto">**: The image height. The value specifies how much space is set aside for the image in the text. If a percent, it's a percentage of the measured or default height. A value of "auto" sets the height to the source image height.
- **source<String>**: The image source. The source is typically a string containing the URI to the

image; however, interpretation of this attribute shall be left to the client.

- **float<String>**. Controls the placement of the graphic. The float property may have these values:
    - **none** Graphic will appear inline with the text.
    - **left** Graphic will appear to the left of the text, and the text will wrap around it.
    - **right** Graphic will appear to the right of the text, and the text will wrap around it.
    - **start** Graphic will appear on the leading edge of the text, and the text will wrap around it. The leading edge is the left if the direction of the paragraph is ltr, and the right if the direction is rtl.
    - **end** Graphic will appear on the trailing edge of the text, and the text will wrap around it. The trailing edge is the right if the direction of the paragraph is ltr, and the left if the direction is rtl.

  If a container has floats, and has verticalAlign="justify", the vertical justification will be ignored.

  A float will appear at the same level as the line it is on if it can while maintaining the anchor in the line. If putting the float at the same level as the line (and thus wrapping the line around the float) would cause the anchor to get pushed out of the line, then the float appears below the line. If the float width exceeds the column width it places the float anyway, and the float may either bleed over other content or be clipped. If the float height exceeds the column height, the float is pushed to the next column or container.

You can control how close the text and the image are by setting the padding properties of the img element. Setting the padding values causes the img element to be treated as slightly larger (or smaller if the padding is negative) than its width and height would indicate. Setting the paddingLeft of a left float will cause the float to be indented from the left edge, and setting the paddingTop causes it to get pushed down.

4.10. span element

Explicit span elements can be used for formatting runs of characters within a paragraph. A <span> may appear in a <p> or implied paragraph and within a <tcy> or <a>. A <span> element can be empty.

Unlike their usage in XHTML, spans in TLF must not be nested.

The content of the <span> element is a string of Unicode text characters possibly marked up with one or more of the following children elements:

- br
- tab

4.11. br element

The <br/> element behaves as a Unicode line separator character. It does not end the paragraph; it merely forces a line break at the position where it appears. <br/> elements are always a child of <span> elements, although the <span> element can be implied. The <br/> element must be an empty tag.

The <br/> element contains no children or attributes.

4.12. tab element

A <tab/> element behaves as a Unicode tab character. Treatment depends on the tab setting that is applied to the paragraph the tab is in. A tab is always a child of <span> elements, though the <span> element may be implied. The <tab/> element is an empty tag and must have no children.

The <tab/> element contains no children or attributes.

5. Link elements

Links are stateful, and within TLF provide functionality similar to CSS. A typical scenario would involve using the <linkNormalFormat> element to format a hyperlink in its non–active, non–hovered state. When the mouse hovers over the hyperlink, the <linkHoverFormat> element is used. When the hyperlink is clicked, the <linkActiveFormat> element is used.

Note that formats set directly on the hyperlink take precedence; that is any attributes on a child element take precedence over the attributes on a parent element. Therefore, span element attributes will take precedence over any link element formats.

The link format tags must be the first children in the containing element. Link formats must be defined before the anchor text spans. Link formats are inherited. If formats have not been explicitly defined, the defaults are applied.

Link format elements contain one and only one <TextLayoutFormat> child element that specifies the formatting attributes that will be applied to any hyperlinks within the parent element, depending on the hyperlink state. For example:

```
<a href="http://www.example.com/someURL/" target="_self">
  <linkHoverFormat><TextLayoutFormat color="#ff0000"/></linkHoverFormat>
  <linkActiveFormat><TextLayoutFormat color="#00ff00"/></linkActiveFormat>
  <linkNormalFormat><TextLayoutFormat color="#0000ff"/></linkNormalFormat>
  <span>
    link text
  </span>
</a>
```

If these link format elements are children of an ancestor element (for instance, the <TextFlow> element) instead of the anchor element itself, they will be inherited by all hyperlinks in the ancestor.

5.1. linkNormalFormat element

The <linkNormalFormat> element defines the appearance of text used in normal state. It has a single child TextLayoutFormat element.

Because the link format elements may be a child of any ancestor of the a element, hyperlink formatting may be may be inherited from link format elements contained by ancestors up the element hierarchy or by using the defaults if no format element is found. By default, this is blue text that is underlined.

5.2. linkHoverFormat element

The <linkHoverFormat> element defines the appearance of text used in hover state. It has a single child TextLayoutFormat element.

Because the link format elements may be a child of any ancestor of the a element, hyperlink formatting may be may be inherited from link format elements contained by ancestors up the element hierarchy or by using the defaults if no format element is found. The default contains no formatting.

5.3. linkActiveFormat element

The <linkActiveFormat> element defines the appearance of text used in active state. It has a single child TextLayoutFormat element.

Because the link format elements may be a child of any ancestor of the a element, hyperlink formatting may be may be inherited from link format elements contained by ancestors up the element hierarchy or by using the defaults if no format element is found. The default contains no formatting.

5.4. TextLayoutFormat Element

The <TextLayoutFormat> element has no children and is used to hold the formatting attributes for the containing link format. A TextLayoutFormat may have any formatting attribute, but only the character-level attributes are applied to link format elements.

The <TextLayoutFormat> element may only be a single child of the following:

- linkNormalFormat
- linkActiveFormat
- linkHoverFormat

6. List Elements

A list is a group of list items, where each list item has some automatically generated content called a *marker*. The marker is what denotes to the reader that a given paragraph is a member of the list. The marker is just some arbitrary text that hangs off the list-item. In an ordered list, it shows the number of the item within the list.

The content of the text is specified by the listStyleType. Depending on the listStyleType value that is applied to a list, it may be either a numbered or unnumbered list. The listStylePosition controls the position of the marker relative to the list item.

6.1. listMarkerFormat

The <listMarkerFormat> element defines the appearance of marker in a list. It has a single child LinkMarkerFormat element.

6.2. ListMarkerFormat

The ListMarkerFormat is used to hold formatting attributes to control the formatting of the marker. By default, the text in the marker will be formatted the same as the text in the list item. It can contain any formatting property plus additional list formatting properties.

The listMarkerFormat can also be used to customize the content of the marker. It can specify text to appear before and after the marker content, as well as a suffix that goes directly after the marker. Each list has an implicit variable called “ordered”. Nested lists create a new “ordered” marker that is in scope only for that list. The variable can be incremented, decremented, or reset. The listMarkerFormat also has a content attribute that allows the numbering of a list marker in a nested list to reflect the numbering of the parent list, as is common in outline numbering.

The ListMarkerFormat may be appear as a child of a TextFlow, div, or list, and be inherited down to a list item.

6.2.1. beforeContent

```
<xs:attribute name="beforeContent" type="xs:string" default=""/>
```

Specifies a string that will appear before the content in the marker. For instance, this could be used for section numbering, with the string “Section “ as beforeContent so the marker renders as “Section 1”.

6.2.2. afterContent

```
<xs:attribute name="afterContent" type="xs:string" default=""/>
```

Specifies a string that will appear after the content in the marker. For example, in some situations it may be convenient to make the afterContent a tab (`&#x9;`) and use that to control the position of the list item text following.

6.2.3. suffix

```
<xs:attribute name="suffix" type="xs:string" default="auto"/>
```

The auto-generated content has a suffix, or default string that is appended to the content and appears before the afterContent. This can be suppressed by setting suffix to “none”. The default value of suffix is “auto”, which resolves to a “.” when the content is a number, and an empty string if not. For instance, the example below would replace the “1.” in a numbered list with “1-“:

```
<ListMarkerFormat suffix="none" afterContent="-"/>
```

Legal values of suffix are:

- **none** No suffix is added.
- **auto** A suffix is added based on the value of the listStyleType property of the list.

6.2.4. counterReset

```
<xs:attribute name="counterReset" type="xs:string" default="none"/>
```

Set the value of the counter. Example:

```
<ListMarkerFormat counterReset="ordered"/>
```

This sets the implicit “ordered” counter value back to 0. To set to a particular value, put the value after the counter name. This sets the counter to 3:

```
<ListMarkerFormat counterReset="ordered 3"/>
```

Legal values of counterReset are:

• **none**	Counter is not changed.
• **ordered**	The “ordered” counter is reset to 0.
• **ordered <integer>**	The “ordered” counter is set to the integer value.

If a ListMarkerFormat has both counterIncrement and counterReset defined, counterReset is applied before counterIncrement.

6.2.5. counterIncrement

```
<xs:attribute name="counterIncrement" type="xs:string" default="ordered 1"/>
```

Adds a value to the counter. To subtract from the counter, add a negative value. For example, to skip a value:

```
<ListMarkerFormat counterIncrement="ordered 2"/>
```

Legal values of counterIncrement are:

• **none**	The counter is not incremented.
• **ordered**	The “ordered” counter is incremented by 1.
• **ordered <integer>**	The “ordered” counter is incremented by the integer value.

6.2.6. content

```
<xs:attribute name="counterReset" type="xs:string" default="counter(ordered)"/>
```

Controls the content of the marker.

Legal values are:

• **none**	Content is the empty string.
• **counter(ordered)**	Display the marker.
• **counter(ordered, ListStyleType)**	Display the marker using the ListStyleType supplied.
• **counters(ordered)**	Starting from the outermost parent list, create a string of values of the ordered counter using the ListStyleType of the associated list, separated by the suffix. This is used for outline numbering – for example, I.1., I.2., etc.
• **counters(ordered, <string>)**	Same as previous, except that the string is used in place of the suffix.
• **counters(ordered, <string>, ListStyleType)**	Same as previous, except that each counter's ListStyleType is replaced with the ListStyleType supplied.

7. Attributes

Case sensitivity

Attribute values may or may not be case sensitive. Values are case sensitive except when interpretation is left to the client.

Data type and "inherit"

For all attributes, the value may be "inherit"—meaning that the value is determined by the parent element. See Inheritance below.

Inheritance

In TLF, attribute values do inherit by default unless the attribute is specifically designated as non–inheriting, or if the value provided is "inherit."

For example, since the fontSize attribute is inheritable, the value 24 (below) is 24 for all the spans in the flow if it is not overridden:

```
<TextFlow fontSize="24"> … </TextFlow>
```

For an attribute like paddingTop which doesn't inherit by default in the example below, the value of 8 would only apply to the <TextFlow> element unless a child element sets paddingTop="inherit":

```
<TextFlow paddingTop="8"> … </TextFlow>
```

7.1. User style attributes

TLF supports user styles in the form of undefined attributes and properties. Even when undefined by TLF, these attributes and their values shall be round–tripped by applications that serialize and deserialize TLF.

For example: `<TextFlow foo="bar"/>` will import and be written into any serialization and imported during deserialization..

Not supported by FXG.

7.2. General attributes

7.2.1. id attribute

```
<xs:attribute name="id" type="xs:string" default=""/>
```

The values of id attributes shall be unique for elements within a TextFlow. Whether or not id attributes are required is left up to the application.

While an id could be used on any element, TLF only round–trips ids for the elements below. Therefore, if you read in a TLF XML containing id attributes on these supported elements, edit the text, and then export back to XML, you will see these same id attributes on those elements.

The id attribute is a string and may be contained by the following elements:

- TextFlow
- div
- list
- li
- p
- g
- tcy
- a
- img
- span

7.2.2. styleName attribute

```
<xs:attribute name="styleName" type="xs:string" default=""/>
```

The styleName attribute is a string and may be contained by the following elements:

- TextFlow
- div
- list
- li
- p

- g
- tcy
- a
- img
- span

Note: styleName is similar to “class” in CSS.

7.2.3. typeName attribute

```
<xs:attribute name="typeName" type="xs:string" default=""/>
```

The typeName attribute is used to preserve the original type of an element. The typeName attribute is a string value and may be contained by the following elements:

- TextFlow
- div
- list
- li
- p
- g
- tcy
- a
- img
- span

Note: typeName may be used to support type selectors.

7.3. Attributes that affect leaf node elements (<span>)

The following attributes can be applied to any element. Their effect is limited to leaf nodes so that they only take affect at the <span> level:

- fontFamily
- fontSize
- fontStyle
- fontWeight
- kerning
- lineHeight

- textDecoration
- lineThrough
- color
- textAlpha
- whiteSpaceCollapse
- backgroundAlpha
- backgroundColor
- baselineShift
- breakOpportunity
- digitCase
- digitWidth
- dominantBaseline
- alignmentBaseline
- ligatureLevel
- locale
- typographicCase
- textRotation
- trackingLeft
- trackingRight
- renderingMode
- cffHinting
- fontLookup

7.3.1. fontFamily attribute

```
<xs:attribute name="fontFamily" type="xs:string" default="Arial"/>
```

The font family name used to render the text. The font family name may also be a comma–delimited list of font families, in which case the client should evaluate them in order. If no font is supplied, the client will choose a variant of the Arial family based on the platform. Which font is used for rendering the text is up to the client and also depends on the glyphs that are being rendered and the fonts that are available.

7.3.2. fontSize attribute

```
<xs:attribute name="fontSize" type="xs:number" default="12"/>
```

The size of the glyphs used to render the text. Number is specified in pixels ranging from a minimum of 1 pixel to a maximum of 720 pixels.

7.3.3. fontStyle attribute

```
<xs:attribute name="fontStyle" type="xs:string" default="normal"/>
```

The style of the glyphs used to render the text.

Valid values include:

- **normal**
- **italic**

7.3.4. fontWeight attribute

```
<xs:attribute name="fontWeight" type="xs:string" default="normal"/>
```

The boldness or lightness of the glyphs used to render the text.

Valid values include:

- **normal**
- **bold**

7.3.5. kerning attribute

```
<xs:attribute name="kerning" type="xs:string" default="auto"/>
```

Specifies whether or not to apply kerning.

Valid values include:

• **auto**	Used to indicate that kerning is enabled except where inappropriate in Asian typography. Kerning is applied between two characters if neither is Kanji, Hiragana, or Katakana.
• **on**	Font–based kerning is applied.
• **off**	No font–based kerning is applied.

7.3.6. lineHeight attribute

```
<xs:attribute name="lineHeight" type="xs:percent | number" default="120%"/>
```

The distance from the baseline of the previous or the next line to the baseline of the current line is equal to the maximum amount of the leading applied to any character in the line.

Whether or not the distance is from the previous or next line depends on the leadingModel. Leading models commonly used for roman text measure up while asian text usually measures down.

Valid values include:

• **number**	in absolute pixels ranging from -720 to 720.
• **percentage**	ranging from -1000% to 1000%.

7.3.7. textDecoration attribute

```
<xs:attribute name="textDecoration" type="xs:string" default="none."/>
```

The text decoration to apply.

Valid values include:

- **none**
- **underline**

7.3.8. lineThrough attribute

```
<xs:attribute name="lineThrough" type="xs:boolean" default="false"/>
```

Specifies whether there should be a line through the text.

Valid values include:

- **true** if the text has strikethrough applied
- **false** if the text has no strikethrough applied

7.3.9. color attribute

```
<xs:attribute name="color" type="xs:string" default="#000000"/>
```

The text color which must be specified in hexadecimal.

7.3.10. textAlpha attribute

```
<xs:attribute name="textAlpha" type="xs:alpha" default="1.0"/>
```

The alpha value applied to the text. Valid values range from 0 (transparent) to 1 (opaque).

7.3.11. whiteSpaceCollapse attribute

```
<xs:attribute name="whiteSpaceCollapse" type="xs:enumeratedString" default="collapse"/>
```

Specifies how white space is handled.

Valid values include:

• **collapse**	Converts line feeds, newlines, and tabs to spaces and collapses adjacent spaces to one.
• **preserve**	Passes whitespace through unchanged, except when the whitespace would result in an implied <p> and <span> that is all whitespace. In such cases, the whitespace is removed.

7.3.12. backgroundAlpha attribute

```
<xs:attribute name="backgroundAlpha" type="xs:alpha" default="1"/>
```

The alpha (transparency) value for the background. A value of 0 is fully transparent, and a value of 1 is fully opaque.

backgroundAlpha is non–inheriting.

7.3.13. backgroundColor attribute

```
<xs:attribute name="textAlpha" type="xs:text" default=" transparent"/>
```

Background color of the text.

backgroundColor is non–inheriting.

Valid values include:

• **transparent**	No color is applied.
• <**hexadecimal value**>	A value that conforms to the color data type.

7.3.14. baselineShift attribute

```
<xs:attribute name="baselineShift" type="xs:number | percent | superscript | subscript“ default="
0"/>
```

Indicates the baseline shift for the element in pixels. The default is 0 (no shift). The element is shifted perpendicular to the baseline by this amount. In horizontal text, a positive baseline shift moves the element up and a negative baseline shift moves the element down.

Valid values include:

- **superscript** Shifts the text up by an amount specified in the font, and applies a transform to the fontSize also based on preferences in the font.
- **subscript** Shifts the text down by an amount specified in the font, and also transforms the fontSize.
- <**Number**> Shifts the text by a number of pixels within a range of -1000 to1000.
- <**Percent**> Shifts the text by a percentage of the fontSize within a range of -1000% to 1000%.

7.3.15. breakOpportunity attribute

```
<xs:attribute name="breakOpportunity" type="xs:enumeratedString" default=" auto"/>
```

Controls where a line can validly break.

Valid values include:

- **auto** Line breaking opportunities are based on standard Unicode character properties, such as breaking between words and on hyphens.
- **any** Indicates that the line may end at any character. This value is typically used when Roman text is embedded in Asian text and it is desirable for breaks to happen in the middle of words.
- **none** No characters in the range are treated as line break opportunities.
- **all** All characters in the range are treated as mandatory line break opportunities, thereby creating one character per line. This value is useful for creating effects like text on a path.

A range is a continuous group of characters that have **none** applied. Therefore, selecting "one of" and applying **none**, results in a range of characters starting with "o" and ending with "f" that have **none** applied and will not get any line breaks inside the range.

7.3.16. digitCase attribute

```
<xs:attribute name="digitCase" type="xs:string" default="default"/>
```

Specifies the digit case.

Valid values include:

- **default** Uses the normal digit case from the font.
- **lining** Uses the lining digit case from the font.
- **oldStyle** Uses the old style digit case from the font.

7.3.17. digitWidth attribute

```
<xs:attribute name="digitWidth" type="xs:string" default="default"/>
```

Specifies how wide digits will be when the text is set.

Valid values include:

• **proportional**	Means that the proportional widths from the font are used, and different digits will have different widths.
• **tabular**	Means that every digit has the same width.
• **default**	Means that the normal width from the font is used.

7.3.18. dominantBaseline attribute

```
<xs:attribute name="dominantBaseline" type="xs:string" default="auto"/>
```

Specifies which of the baselines of the element snaps to the alignmentBaseline to determine the vertical position of the element on the line as follows:

Valid values include:

• **auto**	Resolves based on the textRotation attribute of the span and the locale of the parent paragraph. A textRotation of “rotate270” resolves to ideographicCenter. A locale of Japanese (“ja”) or Chinese (“zh-XX”, “zh_XX”, etc), resolves to ideographicCenter, whereas all others are resolved to roman.
• **roman**	The roman baseline of the element aligns with the alignmentBaseline.
• **ascent**	The ascent baseline of the element aligns with the alignmentBaseline.
• **descent**	The descent baseline of the element aligns with the alignmentBaseline.
• **ideographicTop**	The ideographic top baseline of the element aligns with the alignmentBaseline.
• **ideographicCenter**	The ideographic center baseline of the element aligns with the alignmentBaseline.
• **ideographicBottom**	The ideographic bottom baseline of the element aligns with the alignmentBaseline.
• **useDominantBaseline**	The dominantBaseline aligns with the same baseline of the line.

7.3.19. alignmentBaseline attribute

```
<xs:attribute name="alignmentBaseline" type="xs:string" default="useDominantBaseline “/>
```

Specifies which of the baselines of the line containing the element the dominantBaseline snaps to and thereby determines the vertical position of the element in the line.

Valid values include:

• **roman**	The dominantBaseline aligns with the roman baseline of the line.
• **ascent**	The dominantBaseline aligns with the ascent baseline of the line.
• **descent**	The dominantBaseline aligns with the descent baseline of the line.
• **ideographicTop**	The dominantBaseline aligns with the ideographic top baseline of the line.
• **ideographicCenter**	The dominantBaseline aligns with the ideographic center baseline of the line.
• **ideographicBottom**	The dominantBaseline aligns with the ideographic bottom baseline of the line.
• **useDominantBaseline**	The dominantBaseline aligns with the same baseline of the line.

7.3.20. ligatureLevel attribute

```
<xs:attribute name="alignmentBaseline" type="xs:enumeratedString" default="common “/>
```

The ligature level specifies the font ligatures used for this text. Each successive ligature level includes the ligatures of the preceding level.

• **minimum**	Ligatures required for the language (‘rlig’ table).
• **common**	Commonly used ligatures (‘rlig’ + ‘clig’ + ‘liga’ tables).
• **uncommon**	Discretionary ligatures (‘rlig’ + ‘clig’ + ‘liga’ + ‘dlig’ tables).
• **exotic**	Historic or unusual ligatures (‘rlig’ + ‘clig’ + ‘liga’ + ‘dlig’ + ‘hlig’ tables).

7.3.21. typographicCase attribute

```
<xs:attribute name="typographicCase" type="xs:enumeratedString" default="default"/>
```

Controls the case in which the text will appear.

Valid values include:

• **default**	Specifies default typographic case. The default is “normal” for the font that’s chosen; that is, what you get without applying any features or case changes.
• **lowercaseToSmallCaps**	converts all characters to uppercase, and for those characters which have been converted, specifies the use of small–caps glyphs on output.
• **capsToSmallCaps**	Specifies that all lowercase characters use small–caps glyphs on output.
• **lowercase**	Specifies that all characters use lowercase glyphs on output.
• **uppercase**	Specifies that all characters use uppercase glyphs on output.

7.3.22. textRotation attribute

```
<xs:attribute name="textRotation" type="xs:enumeratedString" default="auto"/>
```

The rotation of the text specified in ninety degree increments.

Valid values include:

• **auto**	Allows the client to decide which characters should be rotated in vertical text.
• **rotate0**	Specifies no rotation.
• **rotate90**	Specifies a 90 degree counter clockwise rotation for full width and wide glyphs only, as determined by the Unicode properties of the glyph.
• **rotate180**	Specifies a 180 degree counter clockwise rotation for full width and wide glyphs only, as determined by the Unicode properties of the glyph.
• **rotate270**	Specifies a 270 degree counter clockwise rotation for full width and wide glyphs only, as determined by the Unicode properties of the glyph.

7.3.23. trackingLeft attribute

```
<xs:attribute name="trackingLeft" type="xs:number | percent" default="0"/>
```

Specifies the space to add to the left of each character.

Valid values include:

• <**number**>	Specifies the number of pixels with an allowable range of -100 to 1000.
• <**percent**>	Specifies a percent of the current fontSize within a range of -100% to 1000%. It may be negative to bring characters closer together.

7.3.24. trackingRight attribute

```
<xs:attribute name="trackingRight" type="xs:number | percent" default="0"/>
```

Specifies the space added to the right of each character.

Valid values include:

• **<number>**	Specifies the number of pixels with an allowable range of -100 to 1000.
• **<percent>**	Specifies a percent of the current fontSize within a range of -100% to 1000%. It may be negative to bring characters closer together.

7.3.25. renderingMode attribute

```
<xs:attribute name="renderingMode" type="xs:enumeratedString" default="cff"/>
```

Defines the rendering mode used for this text. Applies only to embedded fonts; that is, when the fontLookup attribute is set to embeddedCFF.

Valid values include:

• **cff**	Sets the rendering mode to CFF (Compact Font Format).
• **normal**	Sets the rendering mode to the rendering mode that the client specifies as normal.

Not supported by FXG.

7.3.26. cffHinting attribute

```
<xs:attribute name="cffHinting" type="xs:enumeratedString" default="horizontalStem"/>
```

May be used in some clients and not others.

Defines the type of CFF hinting used for this text. CFF hinting determines whether or not to force strong horizontal stems to fit to a sub pixel grid. This attribute applies only if the renderingMode attribute is set to cff and the font is embedded (the fontLookup attribute is set to embeddedCFF). At small screen sizes, hinting produces a clear, legible text for human readers.

Valid values include:

• **none**	No hinting is applied.
• **horizontalStem**	Fits strong horizontal stems to the pixel grid for improved readability.

Not supported by FXG.

7.3.27. fontLookup attribute

```
<xs:attribute name="fontLookup" type="xs:enumeratedString" default="device"/>
```

Specifies whether the font should be matched with a device font or an embedded font.

Valid values include:

none	Specifies device font lookup.
embeddedCFF	Specifies embedded CFF (Compact Font Format) font lookup.

Not supported by FXG.

7.4. Attributes that apply to paragraph elements <p>

Note: Applying these attributes to leaf node elements has no effect.

The following formatting attributes can be applied to any element and affect the formatting of paragraphs, except where noted. Applying these attributes to elements contained within the paragraph (a, tcy, span) is valid but will not affect the text, since the appearance of the paragraphs is driven by the value of these attributes on the paragraph element itself. Note that the values may have been applied via inheritance from a parent element.

The following attributes may be applied at any level and inherited to through children to paragraphs (**p**).

- textAlign
- textAlignLast
- textIndent
- paragraphStartIndent
- paragraphEndIndent
- paragraphSpaceBefore
- paragraphSpaceAfter
- paddingLeft
- paddingRight
- paddingTop
- paddingBottom
- justificationRule
- justificationStyle
- textJustify
- leadingModel
- tabStops
- wordSpacing
- clearFloats

7.4.1. textAlign attribute

```
<xs:attribute name="textAlign" type="xs:string" default="start"/>
```

Specifies the text alignment relative to the text box edges:

Valid values include:

- **start** Specifies start edge alignment. The text is aligned to match the writing order specified by the direction attribute. The left edge if the direction attribute is “ltr” and the right edge if the direction attribute is “rtl”.
- **center** Specifies center alignment within the container.
- **end** Specifies end edge alignment. The text is aligned opposite from the writing order. The right edge if the direction attribute is “ltr” and the left edge if the direction attribute is “rtl”.
- **justify** Specifies that text is justified within the lines so they fill the container space.
- **left** Specifies left edge of the container.
- **right** Specifies right edge of the container.

7.4.2. textAlignLast attribute

```
<xs:attribute name="textAlignLast" type="xs:string" default="start"/>
```

Alignment of the last (or only) line in the paragraph relative to the container in justified text. If textAlign is set to “justify”, textAlignLast specifies how the last line (or only line, if this is a one line block) is aligned. To make a paragraph set all lines justified, set textAlign and textAlignLast to “justify”.

By default, the value is undefined and therefore not set. If undefined, this attribute will inherit its value from an ancestor. If no ancestor has set this attribute, it will have a value of “start”.

Values are identical to textAlign.

7.4.3. textIndent attribute

```
<xs:attribute name="textIndent" type="xs:number" default="0"/>
```

Specifies a paragraph’s first line indentation. The indent is relative to the start edge and is measured in pixels within a range of -1000 to 1000.

7.4.4. paragraphStartIndent attribute

```
<xs:attribute name="paragraphStartIndent" type="xs:number" default="0"/>
```

Specifies a paragraph’s indentation applied to the start edge. It is the left edge if the direction attribute is “ltr.” It is the right edge if the direction attribute is “rtl.” Measured in pixels within a range of 0 to 1000.

7.4.5. paragraphEndIndent attribute

```
<xs:attribute name="paragraphEndIndent" type="xs:number | inherit" default="0"/>
```

Specifies a paragraph's indentation applied to the end edge (right edge if direction is ltr, left edge otherwise). Measured in pixels within a range from 0 to 1000.

7.4.6. paragraphSpaceBefore attribute

```
<xs:attribute name="paragraphSpaceBefore" type="xs:number | inherit" default="0"/>
```

Specifies the space before a paragraph. Adjacent vertical space collapses. For two adjoining paragraphs (A and B), where A has paragraphSpaceAfter 12 and B has paragraphSpaceBefore 24, the total space between the paragraphs will be 24, the maximum of the two, and not 36, the sum. If the paragraph comes at the top of the column, no extra space is left before it. Valid values range from 0 to 1000.

7.4.7. paragraphSpaceAfter attribute

```
<xs:attribute name="paragraphSpaceAfter" type="xs:number | inherit" default="0"/>
```

Specifies the space after a paragraph. Adjacent vertical space collapses. For example, for two adjacent paragraphs one having a space after and the other having a space before, the space will be the maximum of the two spaces, not the sum.

No "space after" is necessary if the paragraph falls at the bottom of the <TextFlow> element. Valid values range from 0 to 1000.

7.4.8. justificationRule attribute

```
<xs:attribute name="justificationRule" type="xs:enumeratedString" default="auto"/>
```

Specifies the justification rule to use.

Valid values include:

• **auto**	Justification is resolved based on the locale of the paragraph. For example, values for Japanese ("ja") and Chinese ("zh-XX", "zh_XX", etc) resolve to eastAsian, while all other locales resolve to space.
• **space**	Specifies justification for Latin and other horizontal scripts that divide words using spaces.
• **eastAsian**	Specifies East Asian justification rules.

7.4.9. justificationStyle attribute

```
<xs:attribute name="justificationStyle" type="xs:enumeratedString" default="auto"/>
```

Specifies the justification style when the justificationRule attribute is set to "eastAsian." The options specify the handling of kinsoku characters, which are Japanese characters that cannot appear at either the beginning or end of a line.

Valid values include:

• **auto**	Resolved based on the locale of the paragraph. If used, all locales resolve to pushInKinsoku. However, this value is only used in conjunction with a justificationRule attribute value of eastAsian and is therefore only applicable to “ja” and all “zh” locales.
• **prioritizeLeastAdjustment**	Bases justification on either expanding or compressing the line, whichever gives a result closest to the desired width.
• **pushInKinsoku**	Bases justification on compressing kinsoku at the end of the line, or expanding it if there is no kinsoku or if that space is insufficient.
• **pushOutOnly**	Bases justification on expanding the line.

7.4.10. textJustify attribute

```
<xs:attribute name="textJustify" type="xs:enumeratedString" default=" interWord"/>
```

Applies when justificationRule is set to space.

Valid values include:

• **interWord**	Spreads justification space out to spaces in the line.
• **distribute**	Spreads it out to letters as well as spaces.

7.4.11. leadingModel attribute

```
<xs:attribute name="leadingModel" type="xs:enumeratedString" default="auto"/>
```

Specifies the leading basis (baseline to which the lineHeight attribute and the leading direction (which determines whether lineHeight attribute refers to the distance of a line’s baseline from that of the line before it or the line after it). When the leading model is up, that means the leading is added above the line; down means it is added below the line.

Valid values include:

• **auto**	Specifies that leading model is chosen automatically based on the paragraph's locale property. Locale values of Japanese ("ja") and Chinese ("zh-XX", "zh_XX", etc) resolve auto to ideographicTopDown and other locales resolve to romanUp.
• **box**	Lines are laid out based on the "natural" bounding box of the line, so line boxes are stacked contiguously. A *line box* is defined as the bounding box around *inline boxes* for all leaf elements on the text line, after they have been aligned using baselineShift, dominantBaseline, alignmentBaseline, etc. For a span, the inline box is obtained by applying leading equally above and below the text content such that its height equals lineHeight. For an inline graphic, lineHeight is ignored; the inline box is derived from its specified dimensions and padding values. The firstBaselineOffset property is ignored with this leading model.
• **ascentDescentUp**	Specifies that leading basis is ASCENT/DESCENT and leading direction is UP.
• **ideographicCenterDown**	Specifies that leading basis is IDEOGRAPHIC_CENTER and leading direction is DOWN.
• **ideographicCenterUp**	Specifies that leading basis is IDEOGRAPHIC_CENTER and leading direction is UP.
• **ideographicTopDown**	Specifies that leading basis is IDEOGRAPHIC_TOP and leading direction is DOWN.
• **ideographicTopUp**	Specifies that leading basis is IDEOGRAPHIC_TOP and leading direction is UP.
• **romanUp**	Specifies that leading basis is ROMAN and leading direction is UP.

7.4.12. tabStops attribute

```
<xs:attribute name="tabStops" type="xs:array" default=""/>
```

Array of tab stops. By default there are no tabs. Each tab stop has the following components:

- position in pixels, relative to the start of the line
- alignment<start, center, end, decimal>
- decimalAlignmentToken<String>: Aligns the text with a particular character or substring within it (for instance, to a decimal point)

tabStops take the following condensed string–based form: A sequence of tab stops, where each tab stop is delimited by one or more spaces. Tab stops may appear in any order and do not need to be sorted by position. A tab stop takes the following string–based form: <alignment type><alignment position>|<alignment token>.

- **alignment type** An optional, non–case sensitive alignment specifier. The alignment type is a single character, and if it is not specified the default is S. Possible values include:
  - S or s for start
  - E or e for end
  - C or c for center
  - D or d for decimal
- **alignment position** A required Number. The vertical bar is used to separate the alignment position from the alignment token, and should only be present if the alignment token is present.
- **alignment token** The alignment token adheres to the following rules:
  - It is optional if the alignment type is D.
  - It should not be present if the alignment type is anything other than D.
  - If the alignment type is D, and the alignment token is not specified, it will take on the default value of '.'.
  - The alignment token may be any sequence of characters terminated by the space that ends the tab stop (for the last tab stop, the terminating space is optional; end of alignment token is implied).
  - A space may be part of the alignment token if it is escaped with a backslash.
  - A backslash may be part of the alignment token if it is escaped with another backslash.

7.4.13. wordSpacing attribute

```
<xs:attribute name="wordSpacing" type="xs:string" default="100%,50%,150%"/>
```

The wordSpacing attribute is used to control the size of the spaces between words for those languages that use spaces to divide words. It is composed of three parts, the optimum, minimum, and maximum spacing. Each part is specified as a percentage of the normal space width as defined in the font. The optimum space is the size to use for spaces in all justification contexts. The minimum and maximum spacing values control how much the spaces can be contracted or expanded during justification.

7.4.14. clearFloats attribute

```
<xs:attribute name="clearFloats" type="xs:stringEnumeration" default="none"/>
```

The clearFloats attribute is to control the placement of paragraphs relative to floats. It allows a paragraph to be placed below a float instead of beside it.

Valid values are:

- **none** The paragraph will wrap around any floats.
- **both** The paragraph placed below any floats that appear to the left or right of it.
- **left** The paragraph placed below any floats that appear to the left of it.
- **right** The paragraph placed below any floats that appear to the right of it.
- **start** The paragraph placed below any floats that appear on its leading edge. This is to the left if the paragraph direction is ltr, to the right otherwise.
- **end** The paragraph placed below any floats that appear on its trailing edge. This is to the right if the paragraph direction is ltr, to the left otherwise.

7.5. Attributes that apply to list elements

The following formatting attributes can be applied to any element and affect the formatting of lists. Applying these attributes to elements contained within the list (a, tcy, span) is valid but will not affect the text, since the appearance of the lists is driven by the value of these attributes on the list element itself. Note that the values may have been applied via inheritance from a parent element.

7.5.1. listStyleType

```
<xs:attribute name="listStyleType" type="xs:stringEnumeration" default="disc"/>
```

Valid values are:

• **none**	No content is generated for the marker.
• **upperAlpha**	Upper case alphabetic “numbering”, A–Z, AA–ZZ, etc.
• **lowerAlpha**	Lower case alphabetic “numbering”, a–z, aa–zz, etc.
• **upperRoman**	Upper case Roman numbering, I, II, III, IV, ...
• **lowerRoman**	Lower case Roman numbering, i, ii, iii, iv, ...
• **disc**	A bullet character marker (filled circle)
• **circle**	A hollow circle character marker
• **square**	A filled square marker.
• **box**	A hollow square marker.
• **check**	A check mark.
• **diamond**	A filled diamond.
• **hyphen**	A dash mark.
• **arabicIndic**	Numbering using Arabic script.
• **bengali**	Numbering using Bengali script
• **decimal**	Numbering using decimal 1, 2, 3, ...
• **decimalLeadingZero**	Numbering using decimal with leading zero 01, 02, 03, ...
• **devanagari**	Numbering using Devangari.
• **gujarati**	Numbering using Gujarati.
• **gurmukhi**	Numbering using Gurmukhi.
• **kannada**	Numbering using Kannada.
• **persian**	Numbering using Persian
• **thai**	Numbering using Thai.
• **urdu**	Numbering using Urdu.
• **cjkEarthlyBranch**	Numbering for CJK.
• **cjkHeavenlyStem**	Numbering for CJK.
• **hangul**	Numbering for Hangul.
• **hangulConstant**	Numbering for Hangul.
• **hiragana**	Numbering for Hiragana.
• **hiraganaIroha**	Numbering for Hiragana.
• **katakana**	Numbering for Katagana.
• **katakanaIroha**	Numbering for Katagana.
• **lowerGreek**	Lower–case Greek alphabetic “numbering”.
• **lowerLatin**	Lower–case alphabetic “numbering”, a–z, aa–zz, etc.
• **upperGreek**	Upper–case Greek alphabetic “numbering”.
• **upperLatin**	Upper–case alphabetic “numbering”, A–Z, AA–ZZ, etc.

7.5.2. listStylePosition

```
<xs:attribute name="listStylePosition" type="xs:stringEnumeration" default="outside"/>
```

Controls the position of the marker relative to the text around it.

Valid values for this property are:

• **inside**	The marker appears inline with the text of the list item.
• **outside**	The marker appears outside the margin of the list item, with the trailing edge of the marker aligned with the leading edge of the list item.

7.5.3. listAutoPadding

```
<xs:attribute name="listAutoPadding" type="xs:number" default="40"/>
```

Specifies an auto indent for the start edge of lists. This value is what is used when the padding value of the list on that side is “auto”.

Valid values include:

• <**number**>	The number of pixels within a range from -1000 to 1000.

7.6. Attributes that apply at the TextFlow level

These attributes can be applied to any text element, but when applied to the <TextFlow> element they affect the entire text flow. They are expected on <TextFlow>, <div>, <list>, and <li> tags. Applying them to other elements has no effect.

Note: A processor MAY ignore attributes applied to other text elements and MAY discard those attributes in round–tripping scenarios.

Note: Applying these attributes to paragraph or leaf node elements has no effect.

The following attributes may be applied at any level but only take effect on TextFlow:

- direction
- blockProgression
- lineBreak
- columnGap
- columnCount
- columnWidth
- firstBaselineOffset

- verticalAlign

7.6.1. blockProgression attribute

```
<xs:attribute name="blockProgression" type="xs:string" default="enumeratedString"/>
```

Controls the direction in which lines are stacked.

Valid values include:

- **tb** In Latin text, lines start at the top and proceed downward.
- **rl** In vertical Chinese or Japanese, lines should start at the right side of the container and proceed leftward.

7.6.2. lineBreak attribute

```
<xs:attribute name="lineBreak" type="xs:enumeratedString" default="toFit"/>
```

Specifies a line break.

lineBreak is non–inheriting.

Valid values include:

- **toFit** Wraps the lines at the edge of the enclosing <TextFlow>.
- **explicit** Breaks the lines only at a Unicode line end character (such as a newline or line separator).

7.6.3. columnGap attribute

```
<xs:attribute name="columnGap" type="xs:number" default="0"/>
```

Specifies the space in pixels between columns within a range of 0 to 1000. It does not include space before the first column or after the last column, as that space is padding.

columnGap is non–inheriting.

7.6.4. columnCount attribute

```
<xs:attribute name="columnCount" type="xs:IntWithEnumeration" default="auto"/>
```

Specifies the number of columns.

columnCount is non–inheriting.

Valid values include:

- **auto** Specifies that the number of columns is determined by the number of columns that would fit in the space specified by the columnWidth attribute.
- **integer** Specifies the number of columns within the valid range of 0 to 50.

The column number overrides the other column settings. If the columnCount attribute is not specified, but columnWidth is, then columnWidth is used to create as many columns as can fit in the container.

7.6.5. columnWidth attribute

```
<xs:attribute name="columnWidth" type="xs:NumberWithEnumeration" default="auto"/>
```

Specifies the column width in pixels.

columnWidth is non–inheriting.

Valid values include:

- **auto** Specifies that the column width is determined by the number of columns specified by the columnCount attribute.
- <**number**> Specifies the width of the columns within the valid range of 0 to 8000.

If columnWidth is specified but not the columnCount attribute, the client creates as many columns of that width as possible given the container width and columnGap settings. Any remainder space is left after the last column.

7.6.6. firstBaselineOffset attribute

```
<xs:attribute name="firstBaselineOffset" type="xs:NumberWithEnumeration" default="auto"/>
```

Specifies the position of the first line of text for each column in the container relative to the top of the container. The first line may appear at the position of the line's ascent, or below by the lineHeight of the first line. The baseline that this attribute refers to is deduced from the container's locale as follows: ideographicBottom for Chinese and Japanese locales, roman otherwise.

Valid values include:

- **auto** Specifies that the line top be aligned to the container top inset.
- **ascent** Specifies an offset equal to the ascent of the line, that is, the ascent of the tallest font in the line, accounting for inline graphics as having the bottom of the graphic on the baseline.
- **lineHeight** Specifies an offset equal to the height of the line.
- <**Number**> The number of pixels to offset in a range from 0 to 1000.

7.6.7. verticalAlign attribute

```
<xs:attribute name="verticalAlign" type="xs:enumeratedString" default="top"/>
```

Specifies the vertical alignment of the lines within the container. The lines may appear at the top of the container, centered within the container, at the bottom, or evenly spread out across the depth of the container.

verticalAlign is non–inheriting.

Valid values include:

top	Specifies alignment with the top edge of the frame.
middle	Specifies alignment with the middle edge of the frame.
bottom	Specifies alignment with the bottom edge of the frame.
justify	Specifies vertical line justification with the frame.
inherit	Specifies alignment should be inherited.

7.7. Attributes that apply to multiple levels

7.7.1. direction attribute

```
<xs:attribute name="direction" type="xs:string" default="ltr"/>
```

Controls the dominant writing direction for the paragraph (left–to–right or right–to–left) and how characters with no implicit writing direction, such as punctuation, are treated. Also controls the direction of the columns, which are set according to the value of the direction attribute of the <TextFlow> element.

Valid values include:

ltr	The writing direction is left–to–right.
rtl	The writing direction is right–to–left.

7.7.2. locale attribute

```
<xs:attribute name="locale" type="xs:string" default="en"/>
```

The locale of the text that controls case transformations and shaping. Standard locale identifiers as described in Unicode Technical Standard #35. For example, en, en_US and en-US are all English, ja is Japanese. Locale applied at the paragraph and higher level impacts resolution of “auto” values for dominantBaseline, justificationRule, justificationStyle and leadingModel. See the individual attributes for resolution values.

7.7.3. paddingLeft attribute

```
<xs:attribute name="paddingLeft" type="xs:string" default="auto"/>
```

Specifies the amount in pixels to inset from left edge to the content area within a range -8000 to 8000. It may be either a number, or “auto”.

A value of “auto” resolves to 0 for all elements except list. Auto padding of a list resolves to the listAutoPadding property if the leading edge of the list-item is on the left, and otherwise will be 0.

paddingLeft is non-inheriting.

paddingLeft applies to TextFlow, div, p, list li and img elements.

7.7.4. paddingRight attribute

```
<xs:attribute name="paddingRight" type=" xs:string “ default="auto"/>
```

Specifies the amount in pixels to inset from right edge to the content area within a range of -8000 to 8000. It may be either a number, or “auto”.

A value of “auto” resolves to 0 for all elements except list. Auto padding of a list resolves to the listAutoPadding property if the leading edge of the list-item is on the right, and otherwise will be 0.

paddingRight is non-inherting.

paddingRight applies to TextFlow, div, p, list li and img elements.

7.7.5. paddingTop attribute

```
<xs:attribute name="paddingTop" type=" xs:string “ default="auto"/>
```

Specifies the amount in pixels to inset from top edge to the content area within a range of -8000 to 8000. It may be either a number, or “auto”.

A value of “auto” resolves to 0 for all elements except list. Auto padding of a list resolves to the listAutoPadding property if the leading edge of the list-item is on the top, and otherwise will be 0.

paddingTop is non-inheriting.

paddingTop applies to TextFlow, div, p, list li and img elements.

7.7.6. paddingBottom attribute

```
<xs:attribute name="paddingBottom" type=" xs:string" default=auto"/>
```

Specifies the amount in pixels to inset from bottom edge to the content area within a range of -8000 to 8000. It may be either a number, or “auto”.

A value of “auto” resolves to 0.

paddingBottom is non-inheriting.

paddingBottom applies to TextFlow, div, p, list li and img elements.
