

Caplin Trader Client 1.4

Layout Configuration XML Reference

March 2009

Contents

1	Preface.....	1
1.1	What this document contains.....	1
	About Caplin document formats	1
1.2	Who should read this document.....	1
1.3	Related documents.....	1
1.4	Typographical conventions.....	2
1.5	Feedback.....	2
1.6	Acknowledgments.....	2
2	Getting Started.....	3
2.1	Technical assumptions and restrictions.....	3
2.2	Using the XML configuration markup.....	3
	About webcentric	3
	Files using the XML markup	5
	An example configuration file	6
	Ordering and nesting of tags	9
3	XML tag reference.....	11
3.1	<Application>	11
3.2	<Border>.....	13
3.3	<Declarations>.....	15
3.4	<Decorators>	15
3.5	<Frame>	16
3.6	<FrameItems>.....	18
3.7	<GUI>	19
3.8	<Handle>	19
3.9	<Panel>.....	22
3.10	<Stack>.....	25
3.11	<Tabstrip>.....	28
3.12	<Terrace>	30
3.13	<Tower>.....	33
4	Glossary of terms and acronyms.....	36

1 Preface

1.1 What this document contains

This reference document describes the XML-based configuration that defines the layout of Caplin Trader Client and other aspects of its appearance, through webcentric. The information in this document applies to the Caplin Trader Client supplied with Caplin Trader version 1.4.

About Caplin document formats

This document is supplied in three formats:

- ◆ Portable document format (*.PDF* file), which you can read on-line using a suitable PDF reader such as Adobe Reader®. This version of the document is formatted as a printable manual; you can print it from the PDF reader.
- ◆ Web pages (*.HTML* files), which you can read on-line using a web browser. To read the web version of the document navigate to the *HTMLDoc_m_n* folder and open the file *index.html*.
- ◆ Microsoft HTML Help (*.CHM* file), which is an HTML format contained in a single file. To read a *.CHM* file just open it – no web browser is needed.

For the best reading experience

On the machine where your browser or PDF reader runs, install the following Microsoft Windows® fonts: Arial, Courier New, Times New Roman, Tahoma. You must have a suitable Microsoft license to use these fonts.

Restrictions on viewing *.CHM* files

You can only read *.CHM* files from Microsoft Windows.

Microsoft Windows security restrictions may prevent you from viewing the content of *.CHM* files that are located on network drives. To fix this either copy the file to a local hard drive on your PC (for example the Desktop), or ask your System Administrator to grant access to the file across the network. For more information see the Microsoft knowledge base article at <http://support.microsoft.com/kb/896054/>.

1.2 Who should read this document

This document is intended for System Administrators and Software Developers who need to configure Caplin Trader Client.

1.3 Related documents

◆ Caplin Trader Client: Customizing The Appearance

This document describes how to use the XML detailed here to modify the layout of Caplin Trader Client Reference Implementation. It also explains how to change aspects of the look and feel of Caplin Trader Client.

1.4 Typographical conventions

The following typographical conventions are used to identify particular elements within the text.

Type	Uses
<i>/AFolder/Afile.txt</i>	File names, folders and directories
<div>Some code;</div>	Code examples
value	XML tag and attribute names
XYZ Product Overview	Document name
◆	Information bullet point

Note: Important Notes are enclosed within a box like this.
Please pay particular attention to these points to ensure proper configuration and operation of the solution.

Tip: Useful information is enclosed within a box like this.
Use these points to find out where to get more help on a topic.

1.5 Feedback

Customer feedback can only improve the quality of our product documentation, and we would welcome any comments, criticisms or suggestions you may have regarding this document.

Please email your feedback to documentation@caplin.com.

1.6 Acknowledgments

Adobe® Reader is a registered trademark of Adobe Systems Incorporated in the United States and/or other countries.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

2 Getting Started

This section explains briefly how the various XML tags may be combined to define the layout of Caplin Trader Client. For more information on how to use the XML tags and attributes defined in this document please refer to **Caplin Trader Client: Customizing The Appearance**.

2.1 Technical assumptions and restrictions

XML

The XML markup defined in this document conforms to XML version 1.0 and the XML schema version defined at <http://www.w3.org/2001/XMLSchema>.

2.2 Using the XML configuration markup

About webcentric

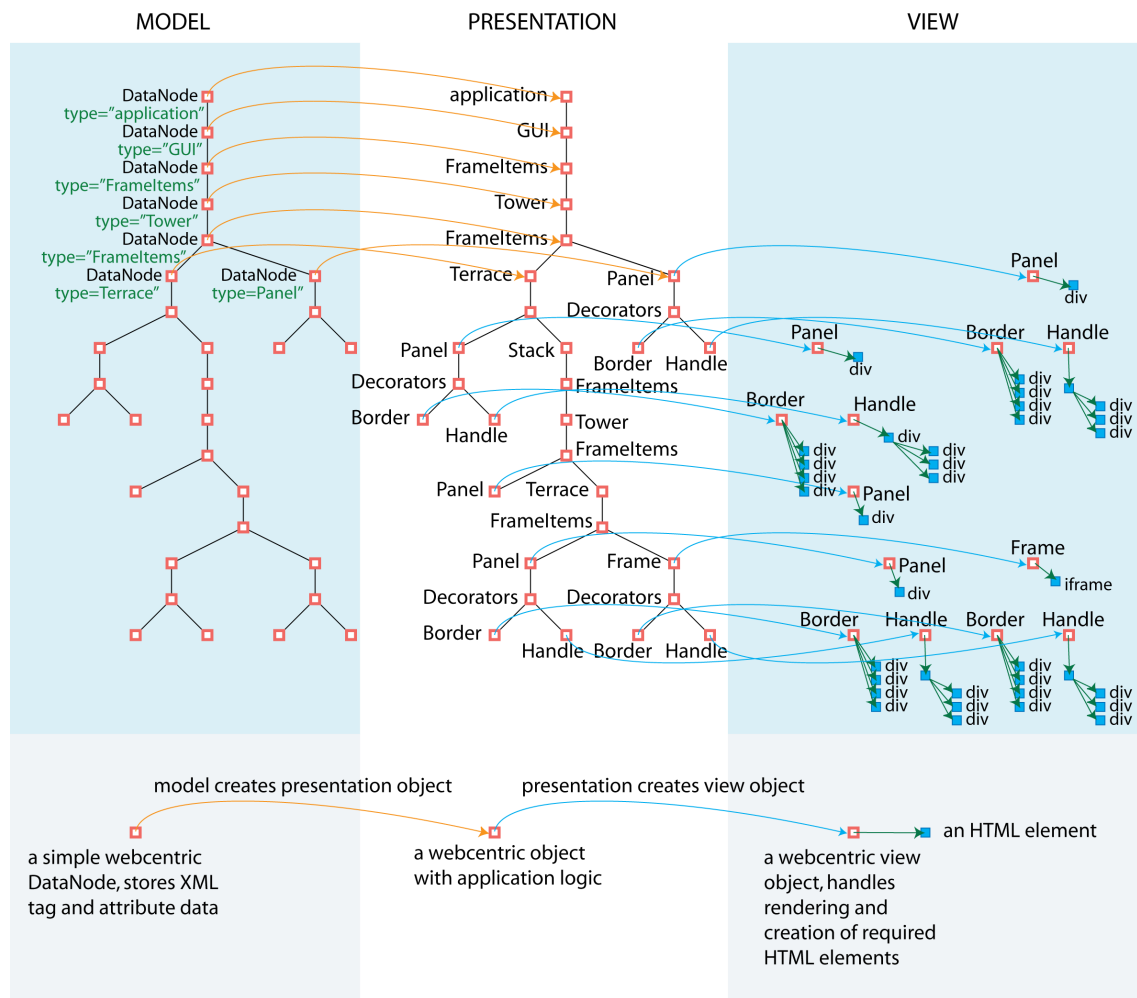
Caplin Trader Client uses a client-side portal framework called webcentric to manage the look and feel of the rendered pages. From the webcentric point of view, Caplin Trader Client is just another webcentric application. Webcentric controls the look and feel of this application according to XML markup that conforms to the definitions in this document.

Model, Presentation and View

The architecture of webcentric is split into three tiers; model, presentation and view. The **model** tier is a hierarchical data structure that contains the fully expanded XML layout data. Each data node in this structure contains the name of the element and the various attribute values, but does not contain the syntax of the XML layout data it was derived from. The logic contained in each node is restricted to generic functionality such as cloning a node, creating a child node, deleting a child node etc. All operations are generic and have no specific logic related to the semantic meaning of the node type. The model tier can be persisted at any time, at which point it is simply translated back into XML.

The **presentation** tier contains analogs of each of the element nodes in the model-tier; each presentation object is created at startup from the information in its corresponding model object. However, the presentation objects also contain the functionality and logic relevant to that object's behaviour in the application; A Tower object has methods related to the behaviour of Towers and a Panel has methods specific to Panel behaviour (for more information on Towers and Panels see **Caplin Trader Client: Customizing The Appearance**).

The **view** tier is responsible for constructing the actual user interface components with which the user will interact. These components are created as HTML. Any interaction with the HTML components is captured in the view object that owns these HTML components, which then relays the event to the presentation layer. If this interaction results in a layout change for that component, then the model is informed of the required change. If need be, changes to the model object's data is then reflected back into the presentation and view.



An example of a how a GUI layout is represented in the webcentric "model - presentation - view" framework.

In the figure above, an XML layout configuration file has been loaded and a hierarchical node structure is created in the model tier. Each node contains information about the XML tag that it represents; specifically a string containing the tag name and string pairs containing the attribute names and values given in the XML source.

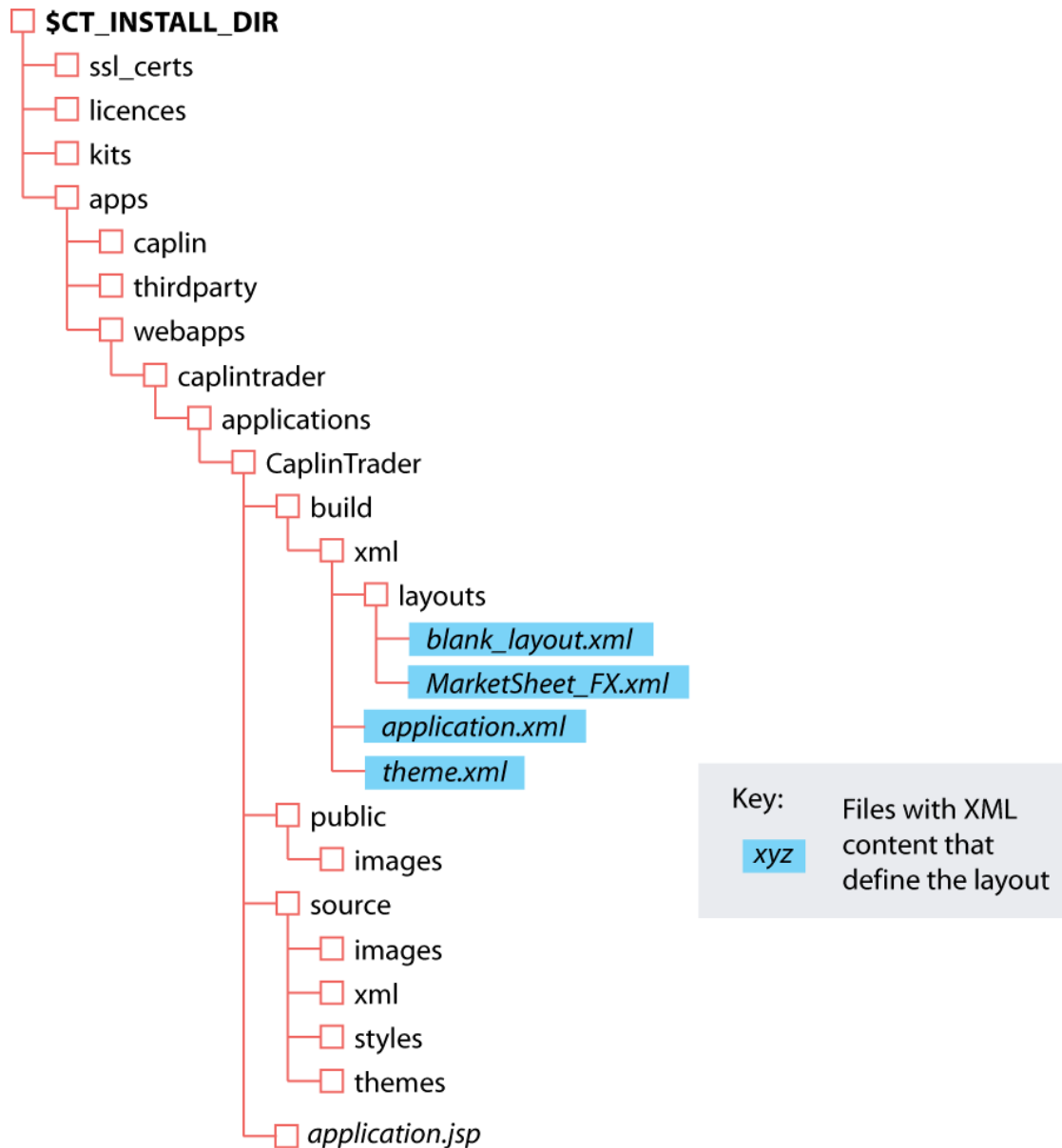
The information the model-tier structure is used to create an associated structure in the presentation tier. This structure has the same number of objects as in the model tier, but in this case the type of each object matches the XML tag name in the corresponding node of the model-tier. The attribute values stored in the model's node are used to set parameters on the object in the presentation tier.

Only the objects that are not frame managers generate objects in the view tier; this includes `Panel`, `Border`, `Handle` and `Frame` amongst others. Each of these view objects is responsible for generating the final HTML elements that make up the interface. For example, a `Border` with four sides will spawn four HTML `div` elements. A `Frame` will spawn an HTML `iframe` element. User interaction with these HTML elements causes events to be fed back to the owning view object, which then passes them on to the presentation object.

If an event causes fundamental changes to the layout of the interface, then the required changes will be fed back to the model object where the changes will be recorded. Webcentric then feeds the changes through the presentation and view tiers and the users page is updated accordingly.

Files using the XML markup

The XML markup can be used in a number of files to determine the layout of an application in Caplin Trader Client. In the figure below, you can see the files that are used to do this.



Tip: For more information on the permitted order and nesting of the XML tags, advanced users may wish to examine the XML schema file *webcentric.xsd* (from which this document is derived)

An example configuration file

An example XML file describing a very simple layout is shown below. For more information on the concepts behind this markup please refer to the document **Caplin Trader Client: Customizing The Appearance**.

The [XML tag reference](#) ¹¹ section defines the XML tags and attributes you can use to define the layouts of the Caplin Trader Client pages and other aspects of the appearance of these pages (such as the colors used).

Note: To have full control over the colors used in the Caplin Trader Client pages, you may also need to modify CSS files and possibly JavaScript code. For further information, see **Caplin Trader Client: Customizing The Appearance**.

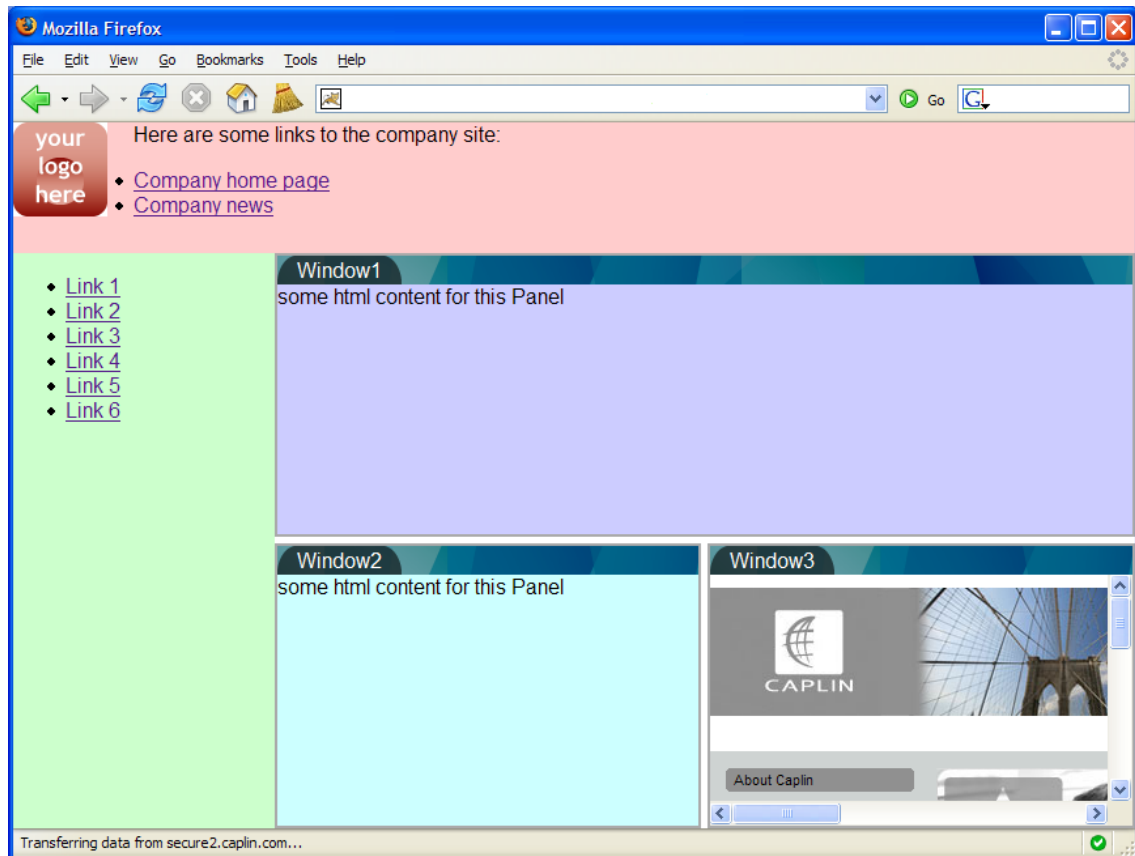
XML for a simple layout

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="file:webcentric.xsd"
  xmlns=""
  xmlns:caplin="http://www.caplin.com"
  drag_container="/*[@id='application-layout']"
  maximize_target="GUI"
  auto_size="true"
  auto_save="false"
  permissioned="false">

  <!-- General purpose global declaration area - put anything at all in here
    and reference it from elsewhere in the document / application. -->
  <Declarations>
    <Decorators id="decorator1">
      <Border style="outer" border_width="2" />
      <Handle handle_height="22" drag_action="SNAP_FRAMEITEM"
        drop_target="SNAP_FRAMEITEM" />
    </Decorators>
  </Declarations>

  <!-- The layout -->
  <GUI>
    <FrameItems>
      <Tower>
        <FrameItems>
          <Panel height="100" background="#fcc" src="titlebar.html" />
          <Terrace>
            <FrameItems>
              <Panel width="200" background="#cfc" src="menu.html" />
              <Stack id="application-layout">
                <FrameItems>
                  <Tower splitters="true">
                    <FrameItems>
                      <Panel background="#ccf" src="content.html"
                        caption="Window1" drop_target="SNAP_FRAMEITEM">
                        <Decorators
                          xref="Declarations/Decorators[@id='decorator1']"/>
                      </Panel>
                      <Terrace splitters="true">
                        <FrameItems>
                          <Panel background="#cff" src="content.html"
                            caption="Window2" drop_target="SNAP_FRAMEITEM">
                            <Decorators ... />
                          </Panel>
                          <Frame src="http://www.caplin.com"
                            caption="Window3" drop_target="SNAP_FRAMEITEM">
                            <Decorators ... />
                          </Frame>
                        </FrameItems>
                      </Terrace>
                    </FrameItems>
                  </Tower>
                </Stack>
              </FrameItems>
            </Terrace>
          </FrameItems>
        </Tower>
      </FrameItems>
    </GUI>
  </Application>
```

The XML markup on the previous page translates into the following Caplin Trader layout (with the exception of the HTML content and graphical details such as the logo and rounded tabs):




Rendering of the simple XML layout

Ordering and nesting of tags

Each top level tag is shown below, together with the child tags that it can typically contain (the children are in no particular order).

Tip: Advanced users may wish to consult the XML Schema for definitive information on the ordering and nesting of tags.

For a description of each tag and its attributes, see the [XML Tag reference](#)  section.

<Application>

This is the outermost tag that contains the layout.

```
<Application>
  <Declarations></Declarations>
  <GUI></GUI>
</Application>
```

<Declarations>

```
<Declarations>
  <Decorators></Decorators>
</Declarations>
```

<GUI>

```
<GUI>
  <Decorators></Decorators>
  <FrameItems></FrameItems>
</GUI>
```

<FrameItems>

```
<FrameItems>
  <Stack></Stack>
  <Tower></Tower>
  <Terrace></Terrace>
  <Panel></Panel>
  <Frame></Frame>
</FrameItems>
```

<Stack>

```
<Stack>
  <Decorators></Decorators>
  <FrameItems></FrameItems>
</Stack>
```

<Tower>

```
<Tower>
  <Decorators></Decorators>
  <FrameItems></FrameItems>
</Tower>
```

<Terrace>

```
<Terrace>
  <Decorators></Decorators>
  <FrameItems></FrameItems>
</Terrace>
```

<Panel>

```
<Panel>
  <Decorators></Decorators>
</Panel>
```

<Frame>

```
<Frame>
  <Decorators></Decorators>
</Frame>
```

<Decorators>

```
<Decorators>
  <Border />
  <Handle />
  <Tabstrip />
</Decorators>
```

<Border>

```
<Border /> (no children)
```

<Handle>

```
<Handle /> (no children)
```

<Tabstrip>

```
<Tabstrip /> (no children)
```

3 XML tag reference

The following sections describe the XML tags. They are arranged in alphabetical order of tag name.

For each tag the attributes you can use within it are listed and described in a table. The “Req?” column indicates whether the attribute is always required (“Y”) or is optional (“N”). Most attributes are optional. If you do not supply an optional attribute within an instance of the tag then webcentric’s behavior depends on whether or not the attribute has a default value. If there is a default, webcentric uses this value. If there is no default (as specified by “(none)” in the Default column), webcentric will not execute the behavior governed by the attribute.

3.1 <Application>

<Application>

This is the root element of the webcentric application document. As well as acting as the container for all other markup elements, the Application element carries attributes which help define the behaviour and presentation of the application. This element has a direct JavaScript counterpart in the presentation-tier Application object. In turn, the JavaScript Application object makes the underlying model accessible through its own 'model' property - this gives a direct reference to the root Application element and through that, to the entire model.

Attributes:

Name	Type	Default	Req?	Description
auto_save	string	true	N	Save user preferences and application state on exit. The application exits when user closes browser or navigates to another url. If auto_save is true, any user preferences that have changed will be saved along with state changes for layouts or components.
auto_size	boolean	true	N	Determines whether the application resizes when the browser window is resized.
drag_container	xpath	(none)	N	Identifies the container element that defines the drag boundaries for a drag operation. It is not possible to drag an element outside this constrained area. Individual draggable elements can define specific drag containers, though it is often be more convenient to define this once for the entire application, on the application node. The value must be an xpath expression that when evaluated identifies a frameset or component.

Name	Type	Default	Req?	Description
layout	string	(none)	N	<p>This attribute specifies the initial layout or layouts to be loaded, if not overridden by a value from the user preferences.</p> <p>Typical use is to specify a default layout to be loaded first time a user launches the application. If preferences are being saved on a per-user basis, this attribute will normally be ignored on subsequent use of the application by the same user - the last loaded layout will be saved as a user preference.</p>
maximize_target	xpath	GUI	N	<p>The default maximize boundary. This identifies the application region to which Frameltems are maximized. It must be an xpath expression identifying a frameset or component.</p> <p>If this attribute is not specified, the default value will be the main content area of the application, excluding any application decorators - heading, menu, toolbars, and so on, but including any components open within the application workspace - navigation panels, consoles and so on. This value will be used to define the maximized size of any Frameltem that is maximized and that does not explicitly declare a maximize_target attribute. In the case of Frameltems that do include a maximize_target attribute declaration, the local declaration takes precedence over the global attribute.</p>
panel_name_max_length	integer	30	N	<p>Defines the maximum length of a Panel name.</p> <p>This value is only used when a Panel can be renamed. For more information, see the "panel_rename" attribute.</p>
panel_rename	boolean	false	N	<p>When "true", an end-user can rename Panels, otherwise they cannot.</p> <p>Individual panels can override this value using the "rename" attribute of the Panel tag.</p>

3.2 <Border>

<Border>

This describes a single border around a frameset or component. The border may completely enclose the target or may be limited to one, two or three sides only. The color and width of each side can be specified individually, if required. Multiple borders can be specified for the same target, if required for visual effect.

Attributes:

Name	Type	Default	Req?	Description
border_bottom_color	string	(none)	N	The hex color value (or color constant value) for the bottom border.
border_color	string	(none)	N	The hex color value (or color constant value) for the whole border.
border_left_color	string	(none)	N	The hex color value (or color constant value) for the left border.
border_right_color	string	(none)	N	The hex color value (or color constant value) for the right border.
border_top_color	string	(none)	N	The hex color value (or color constant value) for the top border.
border_left_width	integer	(none)	N	The width in pixels for the left border.
border_right_width	integer	(none)	N	The width in pixels for the right border.
border_top_width	integer	(none)	N	The width in pixels for the top border.
border_width	integer	(none)	N	The width in pixels for the whole border.

Name	Type	Default	Req?	Description
style	string	(none)	N	<p>This attribute allows for fine-grained control over the visual presentation aspects of components and decorators. The attribute value allows the element to be given a custom appearance or behavior via CSS or JavaScript respectively.</p> <p>In HTMLView, which is the base object for defining the HTML markup of components in the GUI, the value of the style attribute will be incorporated into the CSS class name of the rendered HTML object. In this way, custom visuals can be defined for this element in CSS.</p> <p>For example, using style="outer" in a Border element, results in the rendered HTML element being given the Border_outer CSS class name. In this way CSS visuals can be applied to '.Border_outer' to differentiate from the CSS class of '.Border' in general.</p> <p>Additionally, in JavaScript, a custom view implementation can be provided for a component or decorator. The name of the custom view object is HTML<ElementName>_<style>, where <style> is the value of this attribute and <ElementName> is the name of the element, for example 'Border' or 'Tabstrip' for example. Methods on this view object can provide custom HTML construction mechanisms for the actual HTML components used in the element's display.</p>

3.3 <Declarations>

<Declarations>

This is a general-purpose container intended for content that may be referenced from elsewhere in the document or loaded dynamically by the application at runtime. For example, common sets of decorator declarations, may be declared once here and may then be referenced by multiple components/framesets, avoiding duplication within the markup. Another example is Dialog definitions, which can be declared here and then be referenced by the showDialog action, to be loaded and displayed on demand at runtime. Any content can be hosted here, to be used as seen fit by the application.

Attributes: This tag has no attributes.

3.4 <Decorators>

<Decorators>

The Decorators tag serves to group a number of Decorators to be used on a particular component. Examples of Decorators are Handle, Tabstrip, Border. Any of these may be used in the markup as children of the <Decorators> tag.

Attributes:

Name	Type	Default	Req?	Description
id	unique id	(none)	N	This is a unique identifier. It must be unique within the current application document.
xref	string	(none)	N	This pulls in content from another location within the model. This attribute can contain an IDREF or an xpath expression used to locate the content.

3.5 <Frame>

<Frame>

Frame allows the user to present external content via a URL specified in the src attribute.

Attributes:

Name	Type	Default	Req?	Description
height	integer	(none)	N	<p>This value normally determines the height of the element in pixels. However, if this component is a child of a Tower frameset, its height in pixels is determined by considering the other children of the Tower.</p> <p>Specifically, if the heights of all the children have been defined, then for each child the fraction calculated by dividing its height by the sum total of all the children's heights is multiplied by the vertical space in pixels to be filled by the Tower. This calculated value is the height in pixels for that child. If however, the heights of one or more of the other children have not been defined or, the value of the fixed_size attribute is 'true', then the height of this element will be the specified height in pixels and the other children will fill up the remaining space.</p>
width	integer	(none)	N	<p>This value normally determines the width of the element in pixels. However, if this component is a child of a Terrace frameset, its width in pixels is determined by considering the other children of the Terrace.</p> <p>Specifically, if the widths of all the children have been defined, then for each child the fraction calculated by dividing its width by the sum total of all the children's widths is multiplied by the horizontal space in pixels to be filled by the Terrace. This calculated value is the width in pixels for that child. If however, the widths of one or more of the other children have not been defined or, the value of the fixed_size attribute is 'true', then the width of this element will be the specified width in pixels and the other children will fill up the remaining space.</p>
caption	string	(none)	N	<p>This is a short, descriptive piece of text which may be used by view for handle titles, tab text, and so on.</p> <p>It is best to keep this to just one or two words.</p>

Name	Type	Default	Req?	Description
<code>fixed_size</code>	boolean	false	N	<p>This provides a hint to webcentric that this frameltem should retain its significant size attribute (width or height, depending upon whether the frameset is a Terrace or Tower), even when the containing frameset is resized.</p> <p>If the frameset is a stack, no size attribute is significant, as all dimensions are dictated by the Stack itself. In an MDI frameset both width and height are significant.</p>
<code>img</code>	string	(none)	N	<p>The URL for an image file.</p> <p>This should generally be a small image suitable for display: for example, within the handle or within a tab. A custom JavaScript View object would have to be written to access this stored URL and use it in the tabs, for example. The image cannot be used as a background image for the frame content.</p>
<code>maximized</code>	boolean	(none)	N	Set this value to true if the frameltem should be displayed in the maximized state.
<code>minimized</code>	boolean	(none)	N	Set this value to true if the frameltem should be displayed in the minimized state.
<code>src</code>	string	(none)	Y	<code>src</code> specifies the external URL that is to be loaded into the Frame.

Name	Type	Default	Req?	Description
style	string	(none)	N	<p>This attribute allows for fine-grained control over the visual presentation aspects of components and decorators. The attribute value allows the element to be given a custom appearance or behavior via CSS or JavaScript respectively.</p> <p>In HTMLView, which is the base object for defining the HTML markup of components in the GUI, the value of the style attribute will be incorporated into the CSS class name of the rendered HTML object. In this way, custom visuals can be defined for this element in CSS.</p> <p>For example, using style="outer" in a Border element, results in the rendered HTML element being given the Border_outer CSS class name. In this way CSS visuals can be applied to '.Border_outer' to differentiate from the CSS class of '.Border' in general.</p> <p>Additionally, in JavaScript, a custom view implementation can be provided for a component or decorator. The name of the custom view object is HTML<ElementName>_<style>, where <style> is the value of this attribute and <ElementName> is the name of the element, for example 'Border' or 'Tabstrip' for example. Methods on this view object can provide custom HTML construction mechanisms for the actual HTML components used in the element's display.</p>

3.6 <FrameItems>

<FrameItems>

Container object for child FrameItems within a Frameset.

Attributes: This tag has no attributes.

3.7 <GUI>

<GUI>

This element, with its children, fully describes the graphical interface of the application. The GUI element is actually a specialized form of MDI Frameset, one of several frameset types. The GUI element hosts the main Application Stack and any dialogs.

Attributes:

Name	Type	Default	Req?	Description
id	unique id	(none)	N	This is a unique identifier. It must be unique within the current application document.

3.8 <Handle>

<Handle>

This decorator is used to provide the user with a draggable strip along the top of another component. This Handle also displays the caption of the parent component.

Attributes:

Name	Type	Default	Req?	Description
align	string	top	N	This value defines the alignment of this decorator relative to the main content area of the parent component. 'top' means the Handle will be displayed along the top of the parent; similarly, 'bottom', 'left' and 'right' display accordingly.
handle_height	integer	(none)	N	This defines the 'depth' of this decorator, which may be either the height or the width, depending on whether the alignment of the Handle is a horizontal alignment or vertical one. The default alignment is generally top (a horizontal alignment), but this varies from decorator to decorator (for example the default alignment of StatusBar is bottom). When alignment is top or bottom (that is, horizontal), handle_height defines the height, otherwise it defines the width.

Name	Type	Default	Req?	Description
drag_action	string	(none)	N	This defines the manner in which this decorator and its parent component can be dragged. The value 'SNAP_FRAMEITEM' allows the component to be dragged and dropped on any other component that has the 'drop_target' attribute also set to 'SNAP_FRAMEITEM'. the value 'MDI_DRAG' allows the decorator and its parent component to be dragged and dropped anywhere.
drag_container	xpath	(none)	N	Identifies the container element that defines the drag boundaries for a drag operation. It is not possible to drag an element outside this constrained area. Individual draggable elements can define specific drag containers, though it is often be more convenient to define this once for the entire application, on the application node. The value must be an xpath expression that when evaluated identifies a frameset or component.
drop_target	string	(none)	N	If this attribute is set to 'SNAP_FRAMEITEM' then other components being dragged can be dropped onto this decorator or component.

Name	Type	Default	Req?	Description
style	string	(none)	N	<p>This attribute allows for fine-grained control over the visual presentation aspects of components and decorators. The attribute value allows the element to be given a custom appearance or behavior via CSS or JavaScript respectively.</p> <p>In HTMLView, which is the base object for defining the HTML markup of components in the GUI, the value of the style attribute will be incorporated into the CSS class name of the rendered HTML object. In this way, custom visuals can be defined for this element in CSS.</p> <p>For example, using style="outer" in a Border element, results in the rendered HTML element being given the Border_outer CSS class name. In this way CSS visuals can be applied to '.Border_outer' to differentiate from the CSS class of '.Border' in general.</p> <p>Additionally, in JavaScript, a custom view implementation can be provided for a component or decorator. The name of the custom view object is HTML<ElementName>_<style>, where <style> is the value of this attribute and <ElementName> is the name of the element, for example 'Border' or 'Tabstrip' for example. Methods on this view object can provide custom HTML construction mechanisms for the actual HTML components used in the element's display.</p>

3.9 <Panel>

<Panel>

The Panel is a simple component into which local HTML content can be rendered.

Attributes:

Name	Type	Default	Req?	Description
defer	boolean	(none)	N	This attribute determines whether external content expressed with the 'src' attribute is pulled in on first use as opposed to at loadtime. The src attribute specifies a URL from which further XML layout can be loaded.
background	string	(none)	N	This hex color value defines the component's background color.
height	integer	(none)	N	<p>This value normally determines the height of the element in pixels. However, if this component is a child of a Tower frameset, its height in pixels is determined by considering the other children of the Tower.</p> <p>Specifically, if the heights of all the children have been defined, then for each child the fraction calculated by dividing its height by the sum total of all the children's heights is multiplied by the vertical space in pixels to be filled by the Tower. This calculated value is the height in pixels for that child. If however, the heights of one or more of the other children have not been defined or, the value of the fixed_size attribute is 'true', then the height of this element will be the specified height in pixels and the other children will fill up the remaining space.</p>

Name	Type	Default	Req?	Description
width	integer	(none)	N	<p>This value normally determines the width of the element in pixels. However, if this component is a child of a Terrace frameset, its width in pixels is determined by considering the other children of the Terrace.</p> <p>Specifically, if the widths of all the children have been defined, then for each child the fraction calculated by dividing its width by the sum total of all the children's widths is multiplied by the horizontal space in pixels to be filled by the Terrace. This calculated value is the width in pixels for that child. If however, the widths of one or more of the other children have not been defined or, the value of the fixed_size attribute is 'true', then the width of this element will be the specified width in pixels and the other children will fill up the remaining space.</p>
caption	string	(none)	N	<p>This is a short, descriptive piece of text which may be used by view for handle titles, tab text, and so on.</p> <p>It is best to keep this to just one or two words.</p>
fixed_size	boolean	false	N	<p>This provides a hint to webcentric that this frameItem should retain its significant size attribute (width or height, depending upon whether the frameset is a Terrace or Tower), even when the containing frameset is resized.</p> <p>If the frameset is a stack, no size attribute is significant, as all dimensions are dictated by the Stack itself. In an MDI frameset both width and height are significant.</p>
img	string	(none)	N	<p>The URL for an image file.</p> <p>This should generally be a small image suitable for display: for example, within the handle or within a tab. A custom JavaScript View object would have to be written to access this stored URL and use it in the tabs, for example. The image cannot be used as a background image for the frame content.</p>
maximized	boolean	(none)	N	<p>Set this value to true if the frameItem should be displayed in the maximized state.</p>

Name	Type	Default	Req?	Description
rename	boolean	(none)	N	<p>When "true", an end-user can rename panels, otherwise they cannot.</p> <p>If this attribute is not specified, the value of the "panel_rename" attribute in the Application tag determines whether the panel can be renamed.</p>
script	string	(none)	N	<p>Specifies the URL of a script to be loaded at runtime to be used by the Panel. Note that if HTML content is also specified (via the 'src' attribute), this HTML content is fully loaded into the Panel before the script is run.</p>
src	string	(none)	N	<p>Specifies a file whose contents will be loaded at runtime and written into the content area of the Panel. The intended use is to load static html content into the panel.</p>
style	string	(none)	N	<p>This attribute allows for fine-grained control over the visual presentation aspects of components and decorators. The attribute value allows the element to be given a custom appearance or behavior via CSS or JavaScript respectively.</p> <p>In HTMLView, which is the base object for defining the HTML markup of components in the GUI, the value of the style attribute will be incorporated into the CSS class name of the rendered HTML object. In this way, custom visuals can be defined for this element in CSS.</p> <p>For example, using style="outer" in a Border element, results in the rendered HTML element being given the Border_outer CSS class name. In this way CSS visuals can be applied to '.Border_outer' to differentiate from the CSS class of '.Border' in general.</p> <p>Additionally, in JavaScript, a custom view implementation can be provided for a component or decorator. The name of the custom view object is HTML<ElementName>_<style>, where <style> is the value of this attribute and <ElementName> is the name of the element, for example 'Border' or 'Tabstrip' for example. Methods on this view object can provide custom HTML construction mechanisms for the actual HTML components used in the element's display.</p>

Name	Type	Default	Req?	Description
xml	string	(none)	N	Specifies an xml document to be loaded and stored as a property of the Panel. Usually this attribute is used in conjunction with the 'xsl' attribute. The xslt stylesheet is applied to this xml file and the output from the transformation is written into the content area of the Panel. If no xsl value is given, no processing is performed on the xml file, and nothing is written to the content area of the Panel. However, the xml data remains accessible via a property of the Panel through JavaScript.
xsl	string	(none)	N	Specifies an xslt stylesheet to be applied either to the xml document specified in the src attribute, or to the xml persisted form of the contextNode. The output from the transformation is written into the content area of the Panel.

3.10 <Stack>

<Stack>

The stack is a layout component that arranges child frameItems on top of each other like sheets or pages. Each child fills the same space as the Stack itself, but only one is visible at a time. Typically Stacks are used with Tabstrip decorators that allow the user to switch between child sheets.

Attributes:

Name	Type	Default	Req?	Description
height	integer	(none)	N	This value normally determines the height of the element in pixels. However, if this component is a child of a Tower frameset, its height in pixels is determined by considering the other children of the Tower. Specifically, if the heights of all the children have been defined, then for each child the fraction calculated by dividing its height by the sum total of all the children's heights is multiplied by the vertical space in pixels to be filled by the Tower. This calculated value is the height in pixels for that child. If however, the heights of one or more of the other children have not been defined or, the value of the fixed_size attribute is 'true', then the height of this element will be the specified height in pixels and the other children will fill up the remaining space.

Name	Type	Default	Req?	Description
width	integer	(none)	N	<p>This value normally determines the width of the element in pixels. However, if this component is a child of a Terrace frameset, its width in pixels is determined by considering the other children of the Terrace.</p> <p>Specifically, if the widths of all the children have been defined, then for each child the fraction calculated by dividing its width by the sum total of all the children's widths is multiplied by the horizontal space in pixels to be filled by the Terrace. This calculated value is the width in pixels for that child. If however, the widths of one or more of the other children have not been defined or, the value of the fixed_size attribute is 'true', then the width of this element will be the specified width in pixels and the other children will fill up the remaining space.</p>
maximized	boolean	(none)	N	Set this value to true if the frameltem should be displayed in the maximized state.
minimized	boolean	(none)	N	Set this value to true if the frameltem should be displayed in the minimized state.
padding	integer	(none)	N	<p>The pixel value for the space left between the interior perimeter of a frameset and its children.</p> <p>The same effect can always be achieved with a Border, but padding is more efficient and uses less markup.</p>
selected_ind	integer	(none)	N	<p>The index position of the 'active' or 'selected' child frameltem.</p> <p>All framesets have the concept of a selected child frameltem. For a Stack, it is the visible frameltem (the 'top' item on the Stack). For other framesets the distinction is not so obvious, but it is the selected frameltem that is activated when the frameset is activated. In the case of an MDI frameset, the frameset manages the z-index of the child frameltems; the selected frameltem will appear in front of other frameltems.</p>

Name	Type	Default	Req?	Description
style	string	(none)	N	<p>This attribute allows for fine-grained control over the visual presentation aspects of components and decorators. The attribute value allows the element to be given a custom appearance or behavior via CSS or JavaScript respectively.</p> <p>In HTMLView, which is the base object for defining the HTML markup of components in the GUI, the value of the style attribute will be incorporated into the CSS class name of the rendered HTML object. In this way, custom visuals can be defined for this element in CSS.</p> <p>For example, using style="outer" in a Border element, results in the rendered HTML element being given the Border_outer CSS class name. In this way CSS visuals can be applied to '.Border_outer' to differentiate from the CSS class of '.Border' in general.</p> <p>Additionally, in JavaScript, a custom view implementation can be provided for a component or decorator. The name of the custom view object is HTML<ElementName>_<style>, where <style> is the value of this attribute and <ElementName> is the name of the element, for example 'Border' or 'Tabstrip' for example. Methods on this view object can provide custom HTML construction mechanisms for the actual HTML components used in the element's display.</p>

3.11 <Tabstrip>

<Tabstrip>

The Tabstrip is a Stack decorator that is used to facilitate the selection of child components in the Stack. When one of the tabs is selected the corresponding child component of the Stack is brought to the front and becomes visible.

Attributes:

Name	Type	Default	Req?	Description
handle_height	integer	(none)	N	This defines the 'depth' of this decorator, which may be either the height or the width, depending on whether the alignment of the Handle is a horizontal alignment or vertical one. The default alignment is generally top (a horizontal alignment), but this varies from decorator to decorator (for example the default alignment of StatusBar is bottom). When alignment is top or bottom (that is, horizontal), handle_height defines the height, otherwise it defines the width.
drag_action	string	(none)	N	This defines the manner in which this decorator and its parent component can be dragged. The value 'SNAP_FRAMEITEM' allows the component to be dragged and dropped on any other component that has the 'drop_target' attribute also set to 'SNAP_FRAMEITEM'. the value 'MDI_DRAG' allows the decorator and its parent component to be dragged and dropped anywhere.
drag_container	xpath	(none)	N	Identifies the container element that defines the drag boundaries for a drag operation. It is not possible to drag an element outside this constrained area. Individual draggable elements can define specific drag containers, though it is often be more convenient to define this once for the entire application, on the application node. The value must be an xpath expression that when evaluated identifies a frameset or component.
drop_target	string	(none)	N	If this attribute is set to 'SNAP_FRAMEITEM' then other components being dragged can be dropped onto this decorator or component.

Name	Type	Default	Req?	Description
selection	string	(none)	N	<p>An xpath reference to a target element within the document, which allows the current element to listen for changes in the target, firing a SelectionChanged event when this occurs.</p> <p>This attribute can be used to establish interdependencies between components.</p>
style	string	(none)	N	<p>This attribute allows for fine-grained control over the visual presentation aspects of components and decorators. The attribute value allows the element to be given a custom appearance or behavior via CSS or JavaScript respectively.</p> <p>In HTMLView, which is the base object for defining the HTML markup of components in the GUI, the value of the style attribute will be incorporated into the CSS class name of the rendered HTML object. In this way, custom visuals can be defined for this element in CSS.</p> <p>For example, using style="outer" in a Border element, results in the rendered HTML element being given the Border_outer CSS class name. In this way CSS visuals can be applied to '.Border_outer' to differentiate from the CSS class of '.Border' in general.</p> <p>Additionally, in JavaScript, a custom view implementation can be provided for a component or decorator. The name of the custom view object is HTML<ElementName>_<style>, where <style> is the value of this attribute and <ElementName> is the name of the element, for example 'Border' or 'Tabstrip' for example. Methods on this view object can provide custom HTML construction mechanisms for the actual HTML components used in the element's display.</p>

3.12 <Terrace>

<Terrace>

The Terrace is a layout component that manages the layout of a number of child components. The children of a Terrace are arranged horizontally, from left to right. The relative widths of the children can be controlled via the width attribute.

Attributes:

Name	Type	Default	Req?	Description
height	integer	(none)	N	<p>This value normally determines the height of the element in pixels. However, if this component is a child of a Tower frameset, its height in pixels is determined by considering the other children of the Tower.</p> <p>Specifically, if the heights of all the children have been defined, then for each child the fraction calculated by dividing its height by the sum total of all the children's heights is multiplied by the vertical space in pixels to be filled by the Tower. This calculated value is the height in pixels for that child. If however, the heights of one or more of the other children have not been defined or, the value of the fixed_size attribute is 'true', then the height of this element will be the specified height in pixels and the other children will fill up the remaining space.</p>
width	integer	(none)	N	<p>This value normally determines the width of the element in pixels. However, if this component is a child of a Terrace frameset, its width in pixels is determined by considering the other children of the Terrace.</p> <p>Specifically, if the widths of all the children have been defined, then for each child the fraction calculated by dividing its width by the sum total of all the children's widths is multiplied by the horizontal space in pixels to be filled by the Terrace. This calculated value is the width in pixels for that child. If however, the widths of one or more of the other children have not been defined or, the value of the fixed_size attribute is 'true', then the width of this element will be the specified width in pixels and the other children will fill up the remaining space.</p>

Name	Type	Default	Req?	Description
<code>fixed_size</code>	boolean	false	N	<p>This provides a hint to webcentric that this frameltem should retain its significant size attribute (width or height, depending upon whether the frameset is a Terrace or Tower), even when the containing frameset is resized.</p> <p>If the frameset is a stack, no size attribute is significant, as all dimensions are dictated by the Stack itself. In an MDI frameset both width and height are significant.</p>
<code>maximized</code>	boolean	(none)	N	Set this value to true if the frameltem should be displayed in the maximized state.
<code>minimized</code>	boolean	(none)	N	Set this value to true if the frameltem should be displayed in the minimized state.
<code>padding</code>	integer	(none)	N	<p>The pixel value for the space left between the interior perimeter of a frameset and its children.</p> <p>The same effect can always be achieved with a Border, but padding is more efficient and uses less markup.</p>
<code>selected_ind</code>	integer	(none)	N	<p>The index position of the 'active' or 'selected' child frameltem.</p> <p>All framesets have the concept of a selected child frameltem. For a Stack, it is the visible frameltem (the 'top' item on the Stack). For other framesets the distinction is not so obvious, but it is the selected frameltem that is activated when the frameset is activated. In the case of an MDI frameset, the frameset manages the z-index of the child frameltems; the selected frameltem will appear in front of other frameltems.</p>
<code>splitters</code>	boolean	true	N	If set to true, the frameset will use horizontal or vertical bars to separate the child components. These bars (splitters) can be dragged by the user to resize the height or width of one child component relative to another.

Name	Type	Default	Req?	Description
style	string	(none)	N	<p>This attribute allows for fine-grained control over the visual presentation aspects of components and decorators. The attribute value allows the element to be given a custom appearance or behavior via CSS or JavaScript respectively.</p> <p>In HTMLView, which is the base object for defining the HTML markup of components in the GUI, the value of the style attribute will be incorporated into the CSS class name of the rendered HTML object. In this way, custom visuals can be defined for this element in CSS.</p> <p>For example, using style="outer" in a Border element, results in the rendered HTML element being given the Border_outer CSS class name. In this way CSS visuals can be applied to '.Border_outer' to differentiate from the CSS class of '.Border' in general.</p> <p>Additionally, in JavaScript, a custom view implementation can be provided for a component or decorator. The name of the custom view object is HTML<ElementName>_<style>, where <style> is the value of this attribute and <ElementName> is the name of the element, for example 'Border' or 'Tabstrip' for example. Methods on this view object can provide custom HTML construction mechanisms for the actual HTML components used in the element's display.</p>

3.13 <Tower>

<Tower>

The Tower is a layout component that manages the layout of a number of child components. The children of a Tower are arranged vertically, from top to bottom. The relative heights of the children can be controlled via the height attribute.

Attributes:

Name	Type	Default	Req?	Description
height	integer	(none)	N	<p>This value normally determines the height of the element in pixels. However, if this component is a child of a Tower frameset, its height in pixels is determined by considering the other children of the Tower.</p> <p>Specifically, if the heights of all the children have been defined, then for each child the fraction calculated by dividing its height by the sum total of all the children's heights is multiplied by the vertical space in pixels to be filled by the Tower. This calculated value is the height in pixels for that child. If however, the heights of one or more of the other children have not been defined or, the value of the fixed_size attribute is 'true', then the height of this element will be the specified height in pixels and the other children will fill up the remaining space.</p>
width	integer	(none)	N	<p>This value normally determines the width of the element in pixels. However, if this component is a child of a Terrace frameset, its width in pixels is determined by considering the other children of the Terrace.</p> <p>Specifically, if the widths of all the children have been defined, then for each child the fraction calculated by dividing its width by the sum total of all the children's widths is multiplied by the horizontal space in pixels to be filled by the Terrace. This calculated value is the width in pixels for that child. If however, the widths of one or more of the other children have not been defined or, the value of the fixed_size attribute is 'true', then the width of this element will be the specified width in pixels and the other children will fill up the remaining space.</p>

Name	Type	Default	Req?	Description
<code>caption</code>	string	(none)	N	<p>This is a short, descriptive piece of text which may be used by view for handle titles, tab text, and so on.</p> <p>It is best to keep this to just one or two words.</p>
<code>fixed_size</code>	boolean	false	N	<p>This provides a hint to webcentric that this frameltem should retain its significant size attribute (width or height, depending upon whether the frameset is a Terrace or Tower), even when the containing frameset is resized.</p> <p>If the frameset is a stack, no size attribute is significant, as all dimensions are dictated by the Stack itself. In an MDI frameset both width and height are significant.</p>
<code>maximized</code>	boolean	(none)	N	Set this value to true if the frameltem should be displayed in the maximized state.
<code>minimized</code>	boolean	(none)	N	Set this value to true if the frameltem should be displayed in the minimized state.
<code>padding</code>	integer	(none)	N	<p>The pixel value for the space left between the interior perimeter of a frameset and its children.</p> <p>The same effect can always be achieved with a Border, but padding is more efficient and uses less markup.</p>
<code>selected_ind</code>	integer	(none)	N	<p>The index position of the 'active' or 'selected' child frameltem.</p> <p>All framesets have the concept of a selected child frameltem. For a Stack, it is the visible frameltem (the 'top' item on the Stack). For other framesets the distinction is not so obvious, but it is the selected frameltem that is activated when the frameset is activated. In the case of an MDI frameset, the frameset manages the z-index of the child frameltems; the selected frameltem will appear in front of other frameltems.</p>
<code>splitters</code>	boolean	true	N	If set to true, the frameset will use horizontal or vertical bars to separate the child components. These bars (splitters) can be dragged by the user to resize the height or width of one child component relative to another.

Name	Type	Default	Req?	Description
style	string	(none)	N	<p>This attribute allows for fine-grained control over the visual presentation aspects of components and decorators. The attribute value allows the element to be given a custom appearance or behavior via CSS or JavaScript respectively.</p> <p>In HTMLView, which is the base object for defining the HTML markup of components in the GUI, the value of the style attribute will be incorporated into the CSS class name of the rendered HTML object. In this way, custom visuals can be defined for this element in CSS.</p> <p>For example, using style="outer" in a Border element, results in the rendered HTML element being given the Border_outer CSS class name. In this way CSS visuals can be applied to '.Border_outer' to differentiate from the CSS class of '.Border' in general.</p> <p>Additionally, in JavaScript, a custom view implementation can be provided for a component or decorator. The name of the custom view object is HTML<ElementName>_<style>, where <style> is the value of this attribute and <ElementName> is the name of the element, for example 'Border' or 'Tabstrip' for example. Methods on this view object can provide custom HTML construction mechanisms for the actual HTML components used in the element's display.</p>

4 Glossary of terms and acronyms

This section contains a glossary of terms and acronyms relating to the Caplin Trader Client Layout Configuration XML.

Term	Definition
Caplin Platform	The Caplin Platform is a suite of software products for on-line financial trading and Web delivery of real-time market data.
Caplin Trader	Caplin Trader is a complete platform and toolkit for building multi-product trading portals. It is built on the Caplin Platform and webcentric .
Caplin Trader Client	Caplin Trader Client is a Web application written in Ajax that provides a rich trading workstation in a browser. It is built using webcentric.
webcentric	A client-side portal framework that uses Ajax technology. Caplin Trader Client uses webcentric to manage the look and feel of the rendered pages. From the webcentric point of view, Caplin Trader Client is just another webcentric application. Webcentric controls the look and feel of this application according to XML markup that conforms to the definitions in this document.

Contact Us

Caplin Systems Ltd
Triton Court
14 Finsbury Square
London EC2A 1BR
Telephone: +44 20 7826 9600
Fax: +44 20 7826 9610
www.caplin.com

The information contained in this publication is subject to UK, US and international copyright laws and treaties and all rights are reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the written authorization of an Officer of Caplin Systems Limited.

Various Caplin technologies described in this document are the subject of patent applications. All trademarks, company names, logos and service marks/names ("Marks") displayed in this publication are the property of Caplin or other third parties and may be registered trademarks. You are not permitted to use any Mark without the prior written consent of Caplin or the owner of that Mark.

This publication is provided "as is" without warranty of any kind, either express or implied, including, but not limited to, warranties of merchantability, fitness for a particular purpose, or non-infringement.

This publication could include technical inaccuracies or typographical errors and is subject to change without notice. Changes are periodically added to the information herein; these changes will be incorporated in new editions of this publication.

Caplin Systems Limited may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

This publication may contain links to third-party web sites; Caplin Systems Limited is not responsible for the content of such sites.