CAPLIN

Caplin Trader Client 1.4

Grid Configuration XML Reference (DRAFT)

September 2009

Contents

1 Preface

1.1 What this document contains

This reference document describes the XML-based configuration that defines the layout and functionality of grids displayed in Caplin Trader Client.

. Updated D:\Development\DHTML\main\libraries\grid\src\resources\schema\gridDefinitions.rnc with the tag.

The information in this document applies to 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 grids for Caplin Trader Client.

1.3 Related documents

- **Caplin Trader Client: Customizing the Appearance**

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

- **Caplin Trader Client: Composite Component Configuration XML Reference**

  Describes the XML-based configuration that defines the layout and functionality of the composite display component in Caplin Trader Client. Composite display components can contain grids.

1.4 Typographical conventions

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

Type	***Uses***
/AFolder/Afile.txt	File names, folders and directories
`Some code;`	Program output and code examples
`value`	XML tag and attribute names
XYZ Product Overview	Document name
◆	Information bullet point
■	Action bullet point – an action you should perform

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 Technical assumptions and restrictions

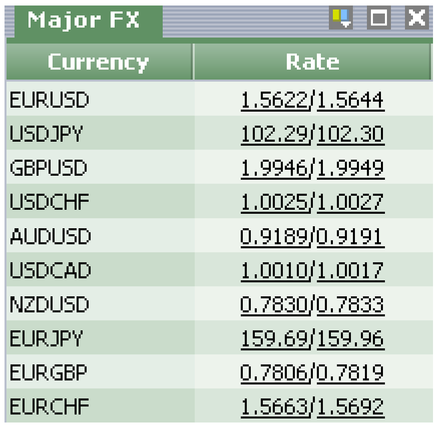
XML

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 Introduction to grids

Caplin Trader Client can display data within panels in a grid format. A grid is a table of data where all of the rows of the table have a similar set of properties (fields), and the format of the data displayed under a particular column is identical for each of the rows.

Here is an example of a grid, as displayed by the reference implementation of Caplin Trader Client:



Currency	Rate
EURUSD	1.5622/1.5644
USDJPY	102.29/102.30
GBPUSD	1.9946/1.9949
USDCHF	1.0025/1.0027
AUDUSD	0.9189/0.9191
USDCAD	1.0010/1.0017
NZDUSD	0.7830/0.7833
EURJPY	159.69/159.96
EURGBP	0.7806/0.7819
EURCHF	1.5663/1.5692

Example grid display

This example is a grid that displays a list of major FX currency pairs. It is displayed within a rectangular display panel. The grid has a name (Major FX), which is displayed in the tab attached to the display panel. It has two columns, with headings Currency and Rate. Each table cell under the Currency column contains the name of a currency pair as simple text. Each cell under the Rate column contains the Bid and Ask prices for the currency, separated by a '/' character.

Grids are an ideal mechanism for displaying large quantities of summary data in tabular format. You can attach renderers to the headings and cells of a grid; these allow you to modify the appearance and behavior of particular cells and column headers. For example rate values can be made into “links” (as shown in the example grid display), where clicking on a link will display a trade ticket for the associated currency pair.

Grids are optimized to handle large tables efficiently. Caplin Trader Client displays a scroll bar that represents all the available data in the grid, but it only registers for streaming data from the rows that are currently visible.

You can create simple grids using the XML configuration defined in this document. There is also well defined API, with suitable extension points, that allows you to implement more complex grids with custom styling and behavior.

2.1 An example configuration file

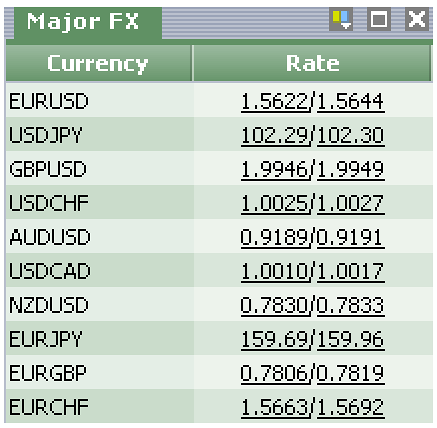
An example XML file describing a simple grid configuration is shown below.

The XML Reference information 12 section defines the XML tags and attributes you can use to define grid layouts in Caplin Trader Client pages. Also see the section Ordering and nesting of tags 12..

XML for a simple grid configuration

```
<?xml version="1.0"?>
<gridDefinitions xmlns="http://www.caplin.com/CaplinTrader/grid">

   <dataProviderMappings>
      <dataProviderMapping id="rttpContainerGridDataProvider"
                           className="caplin.grid.RttpContainerGridDataProvider" />
   </dataProviderMappings>

   <decoratorMappings>
      <decoratorMapping id="dragDecorator"
                        className="caplin.grid.decorator.DragDecorator" />
   </decoratorMappings>

   <templates>
      <gridTemplate id="All">
         <decorators>
            <dragDecorator />
         </decorators>
      </gridTemplate>

      <gridTemplate id="FX" baseTemplate="All" displayedColumns="description,rate">
         <columnDefinitions>
            <column id="description"
                    fields="InstrumentDescription"
                    displayName="Currency" width="70"
                    mandatory="true"/>

            <column id="rate"
                    cellRenderer="caplin.dom.renderer.SpreadElementRenderer"
                    fields="BestBid,BestAsk"
                    displayName="Rate"
                    width="100"/>

            <column id="bestbid"
                    cellRenderer="caplin.dom.renderer.TradableElementRenderer"
                    fields="BestBid"
                    displayName="Best Bid"
                    width="100"/>

            <column id="bestask"
                    cellRenderer="caplin.dom.renderer.TradableElementRenderer"
                    fields="BestAsk"
                    displayName="Best Ask"
                    width="100"/>
         </columnDefinitions>
      </gridTemplate>
   </templates>

   <grids>
      <grid id="FX.Major" displayName="Major FX" baseTemplate="FX">
         <gridRowModel>
            <rttpContainerGridDataProvider container="/CONTAINER/FX/Major" />
         </gridRowModel>
      </grid>

      <grid id="FX.Minor" displayName="Minor FX" baseTemplate="FX">
      <columnDefinitions>
         <column id="description" headerRenderer="inlineTextFilter" />
      </columnDefinitions>
         <gridRowModel>
            <rttpContainerGridDataProvider container="/CONTAINER/FX/Minor" />
         </gridRowModel>
      </grid>
   </grids>

</gridDefinitions>
```

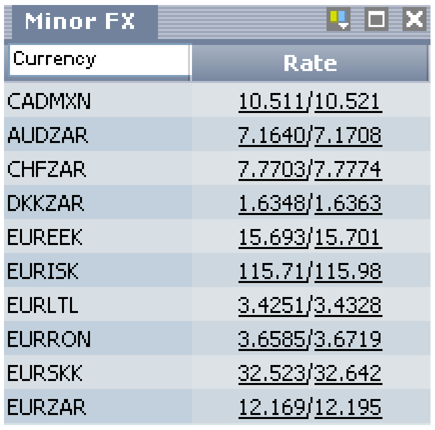
This configuration defines two grids that display FX information:



Major FX

Currency	Rate
EURUSD	1.5622/1.5644
USDJPY	102.29/102.30
GBPUSD	1.9946/1.9949
USDCHF	1.0025/1.0027
AUDUSD	0.9189/0.9191
USDCAD	1.0010/1.0017
NZDUSD	0.7830/0.7833
EURJPY	159.69/159.96
EURGBP	0.7806/0.7819
EURCHF	1.5663/1.5692

<grid id="FX.Major" displayName="Major FX" ...>



Minor FX

Currency	Rate
CADMXN	10.511/10.521
AUDZAR	7.1640/7.1708
CHFZAR	7.7703/7.7774
DKKZAR	1.6348/1.6363
EUREEK	15.693/15.701
EURISK	115.71/115.98
EURLTL	3.4251/3.4328
EURRON	3.6585/3.6719
EURSKK	32.523/32.642
EURZAR	12.169/12.195

<grid id="FX.Minor" displayName="Minor FX"...>

An explanation of the example XML configuration

Here is an explanation of what the example XML configuration contains and how this relates to what the end-user sees on the screen:

- **`<dataProviderMappings>`** contains the definition of a single data provider that supplies the data to a grid:

```
<dataProviderMappings>
   <dataProviderMapping id="rttpContainerGridDataProvider"
                        className="caplin.grid.RttpContainerGridDataProvider" />
</dataProviderMappings>
```

The **`<dataProviderMapping>`** tag defines the JavaScript class that implements the data provider. It maps this class to an `id` (`rttpContainerGridDataProvider`) that the rest of the XML configuration can use as a tag (`<rttpContainerGridDataProvider>`) which refers to this particular data provider.

- **`<decoratorMappings>`** contains the definition of a grid decorator called `dragDecorator`:

```
<decoratorMappings>
   <decoratorMapping id="dragDecorator"
                     className="caplin.grid.decorator.DragDecorator" />
</decoratorMappings>
```

Decorators define various aspects of what the grid looks like and how it behaves. In this particular case the drag decorator is specified. A drag decorator allows an instrument to be dragged out of a grid, so it can be dropped into other screen component, such as a trade panel.

The **`<decoratorMapping>`** tag defines the JavaScript class that implements the drag decorator. It maps this class to an id (`dragDecorator`) that the rest of the XML configuration can use as a tag (`<dragDecorator>`) to refer to this particular decorator.

- **`<templates>`** contains the definitions of two grid templates (`<gridTemplate>`):

```
<templates>
   <gridTemplate id="All">
      <decorators>
         <dragDecorator />
      </decorators>
   </gridTemplate>

   <gridTemplate id="FX" baseTemplate="All" displayedColumns="description,rate">
   ...
   </gridTemplate>
</templates>
```

Grid templates allow grids to be defined in an inheritance hierarchy. In this example there is a basic grid template called “ALL”, which consists just of a drag decorator, so it allows instruments to be dragged out of it.

The grid template called “FX” inherits the characteristics of the “ALL” template (`baseTemplate="All"`), so it too can have instruments dragged out of it. The “FX” grid template additionally defines the columns of an FX grid:

```
<gridTemplate id="FX" baseTemplate="All" displayedColumns="description,rate">
   <columnDefinitions>
       <column id="description"
              fields="InstrumentDescription"
              displayName="Currency"
              width="70"
              mandatory="true"/>

      <column id="rate"
              cellRenderer="caplin.dom.renderer.SpreadElementRenderer"
              fields="BestBid,BestAsk"
              displayName="Rate"
              width="100"/>

      <column id="bestbid" .../>
      <column id="bestask" ... />
   </columnDefinitions>
</gridTemplate>
```

The **`<column>`** tags define four grid columns called “description”, “rate”, “bestbid”, and “bestask”.

Each `<column>` tag defines:

– The pixel width of the column (`width="70"`).

– The data field or fields to be displayed in each cell of that column (`fields="InstrumentDescription"`).

– The text to be displayed in the heading of the column (`displayName="Currency"`).

– An optional renderer that modifies the display format and behavior (if any) of the column heading (`headerRenderer="inlineTextFilter"`).

– An optional renderer that modifies the display format and behavior (if any) of the cells in the column (`cellRenderer="caplin.dom.renderer.RateTextRenderer"`).

The `mandatory` attribute of the “description” column (`mandatory="true"`) specifies that the end user cannot remove this column from a grid; the default is to allow end users to add and remove columns.

The `displayedColumns` attribute of the grid template (`displayedColumns="description, rate"`) specifies that when an FX grid is first displayed, only the description and rate columns are to be shown on the screen.

- **`<grids>`** defines the actual grids that can be added to layouts:

```
<grids>
   <grid id="FX.Major" displayName="Major FX" baseTemplate="FX">
      <gridRowModel>
         <rttpContainerGridDataProvider container="/CONTAINER/FX/Major" />
      </gridRowModel>
   </grid>

   <grid id="FX.Minor" displayName="Minor FX" baseTemplate="FX">
      <columnDefinitions>
         <column id="description" headerRenderer="inlineTextFilter" />
      </columnDefinitions>
      <gridRowModel>
         <rttpContainerGridDataProvider container="/CONTAINER/FX/Minor" />
      </gridRowModel>
   </grid>
</grids>
```

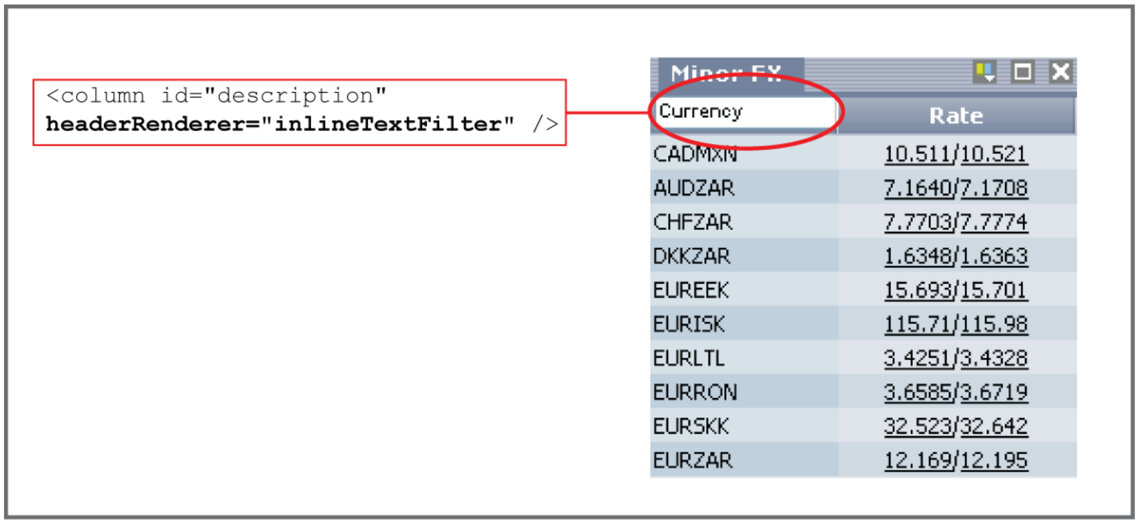
Each grid is defined in a **`<grid>`** tag. In this example there are two grids, “FX.Major” and “FX.Minor”, each of which inherits its characteristics from the “FX” grid template. So both grids display description and rate information in columns of the same widths. To change these aspects of the appearance of both these grids you only need to modify the definition of the parent “FX” grid template. Both grids also allow the end user to drag an instrument out of the grid; they inherit this behaviour from the top level “All” grid template.

The two grids use different row models, as defined in the **`<gridRowModel>`** tags. In this example the row model defines the data provider that fills the grid: a named RTTP container (`<rttpContainerGridDataProvider>`). Each grid obtains its data from a different container (`container="/CONTAINER/FX/Major"`, `container="/CONTAINER/FX/Minor"`), so the two grids display different sets of FX instruments.

The XML for the “FX.Minor” grid also includes an additional column definition:

```
    <columnDefinitions>
         <column id="description" headerRenderer="inlineTextFilter" />
      </columnDefinitions>
```

The “description” column has a header renderer (`headerRenderer="inlineTextFilter"`). The “inlineTextFilter” renderer is supplied with Caplin Trader client. It provides a text box into which the end user can enter search criteria to select the rows displayed in the grid. In this particular case the text filter allows the user to select a subset of the minor currency pairs. For example, entering “EUR” will cause all the minor currency pairs starting with EUR to be displayed (EUREEK, EURISK, EURLTL, and so on).



```
<column id="description"
headerRenderer="inlineTextFilter" />
```

Currency	Rate
CADMXN	10.511/10.521
AUDZAR	7.1640/7.1708
CHFZAR	7.7703/7.7774
DKKZAR	1.6348/1.6363
EUREEK	15.693/15.701
EURISK	115.71/115.98
EURLTL	3.4251/3.4328
EURRON	3.6585/3.6719
EURSKK	32.523/32.642
EURZAR	12.169/12.195

Effect of applying the inlineTextFilter header renderer to the FX.Minor grid

The XML for the “FX.Major” grid does not specify a header renderer, so the default renderer is used; this just displays simple text in the header of every column.

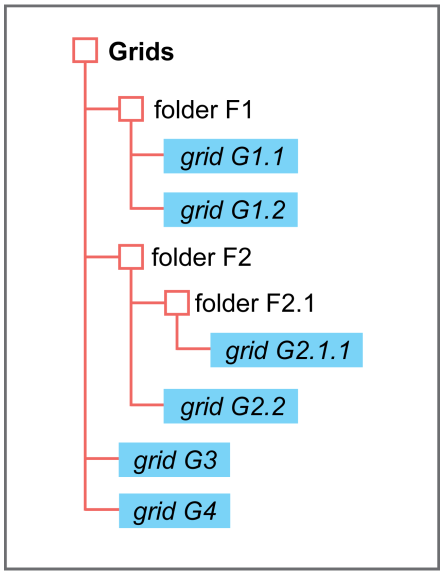
2.2 Folders

Grids can be organized into a nested hierarchy using folders. This is just like organizing files on a disk into a directory structure. The folder hierarchy can be made available to end users, for example when they select the menu option to insert a grid in a layout.

Example of XML that organizes grids into folders

```
<grids>
   <folder displayName="F1">
      <grid id="G1.1">
      </grid>
      <grid id="G1.2">
      </grid>
   </folder>
   <folder displayName="F2">
      <folder displayName="F2.1">
         <grid id="G2.1.1">
         </grid>
      </folder>
      <grid id="G2.2">
      </grid>
   </folder>
   <grid id="G3">
   </grid>
   <grid id="G4">
   </grid>
</grids>
```

The following diagram shows the folder hierarchy defined by this XML fragment.



Grids in a folder hierarchy

2.3 Grid filters

A grid row model can contain a filter expression. The filter expression allows you to restrict the number of rows that are displayed in the grid. For example, the filter expression “BidSize >= 1000” selects only those rows where the BidSize field contains a value greater than or equal to 1,000.

To specify a filter use the `<filterExpression>` tag and its child tags `<fieldFilter>`, `<and>`, and `<or>`. You can also specify a filter on a column using the `<column>` tag and its child `<filter>` tag.

Here are some examples.

This XML fragment specifies the filter: BidSize >= 1000

```
<gridRowModel>
  ...
  <filterExpression>
    <fieldFilter field="BidSize" operator=">=" value="1000" />
  </filterExpression>
</gridRowModel>
```

Filter expressions can specify multiple field criteria using logical OR and logical AND:

Filter: (BidSize >= 1000) AND (AskSize < 1015)

```
<gridRowModel>
  ...
  <filterExpression>
    <and>
    <fieldFilter field="BidSize" operator=">=" value="1000" />
    <fieldFilter field="AskSize" operator="<" value="1015" />
    </and>
  </filterExpression>
</gridRowModel>
```

The following example shows how to specify a filter expression comprising nested OR and AND conditions.

**Filter: (Description = “.*AUSTRIA.*”) AND (CpnRate >= 5)
AND ((BidYield >5.2) OR (AskYield <= 6.4))**

```
<gridRowModel>
  ...
  <filterExpression>
    <and>
      <fieldFilter field="Description" operator="#" value=".*AUSTRIA.*" />
      <fieldFilter field="CpnRate" operator=">=" value="5" />
      <or>
        <fieldFilter field="BidYield" operator=">" value="5.2" />
        <fieldFilter field="AskYield" operator="<=" value="6.4" />
      </or>
    </and>
  </filterExpression>
</gridRowModel>
```

The value specified in a `<fieldFilter>` can be a regular expression. In example above, the filter for the `Description` field uses the regular expression `.*AUSTRIA.*`

This regular expression specifies grid rows where the `Description` field contains the string `"AUSTRIA"` in any position within the field. The `#` operator specifies that the match should be case insensitive, so the filter selects descriptions containing, for example, `"AUSTRIA"`, `"austria"`, `"Austria"`, or `"ausTria"`.

The following example shows how to specify a column filter on a column that is displaying best ask prices.

Filter: BestAsk less than "7.5"

```
<columnDefinitions>
  ...
  <column id="bestask"
          cellRenderer="caplin.dom.renderer.TradableElementRenderer"
          fields="BestAsk"
          displayName="Best Ask"
          width="100">

     <filter type="LESS_THAN" value="7.5">

  </column>
  ...

</columnDefinitions>
```

When specifying a column filter, a filter type is specified rather than a filter operator. If the column is displaying values from more than one field (for example "best bid"/"best ask"), then the column filter is applied to the primary field of the column (the first field of the column).

A filter defined as a column filter is visible in the column header when the grid is displayed in a layout. The end user can change a column filter or completely remove the filter from the grid. Filters that are defined using the `<fieldFilter>` tag are not visible when the grid is displayed in a layout, and cannot be changed or removed by the end user.

3 XML Reference information

This is the XML reference information for the Grid Display Component.

3.1 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 Relax NG Schema (*gridDefinitions.rnc*) for definitive information on the ordering and nesting of tags.

For a description of each tag and its attributes, see XML Tag Descriptions 15.

<gridDefinitions>

This is the outermost tag.

```
<gridDefinitions>
   <dataProviderMappings></dataProviderMappings> (one only)
   <decoratorMappings></decoratorMappings> (zero or one)
   <templates></templates> (zero or one)
   <grids></grids> (one only)
</gridDefinitions>
```

<dataProviderMappings>

```
<dataProviderMappings>
   <dataProviderMapping /> (one or more)
</dataProviderMappings>
```

<decoratorMappings>

```
<decoratorMappings>
   <decoratorMapping /> (one or more)
</decoratorMappings>
```

<templates>

```
<templates>
   <gridTemplate></gridTemplate> (one or more)
</templates>
```

<grids>

```
<grids>  (Must contain at least one of the following tags...)
   <folder></folder>  (zero or more)
   <grid></grid> (zero or more)
   <legacyGrid /> (zero or more)
</grids>
```

<gridTemplate>

```
<gridTemplate>  (The following tags can be in any order...)
   <decorators></decorators> (zero or one)
   <columnDefinitions></columnDefinitions> (zero or one)
   <gridRowModel></gridRowModel> (zero or one)
</gridTemplate>
```

<folder>

```
<folder>
   <folder></folder> (zero or more)
   <grid></grid>  (zero or more)
</folder>
```

<grid>

```
<grid> (The following tags can be in any order...)
   <decorators></decorators> (zero or one)
   <columnDefinitions></columnDefinitions> (zero or one)
   <gridRowModel></gridRowModel> (zero or one)
</grid>
```

<decorators>

```
<decorators> (Must contain at least one of the following tags...)
   <columnMenuDecorator />
   <columnSortDecorator />
   <clearColumnFiltersDecorator />
   <dragDecorator />
   <dropdecorator />
   <groupHeaderDecorator />
   <hoverDecorator />
   <loadingDataDecorator />
   <noDataFoundDecorator />
   <openAjaxFiltersDecorator />
   <removeRowDecorator />
   <selectionDecorator />
   <scrollTipDecorator />
</<decorators>
```

<columnDefinitions>

```
<columnDefinitions>
   <column></column> (one or more)
</columnDefinitions>
```

<gridRowModel>

```
<gridRowModel>
   <rttpContainerGridDataProvider />
   OR
   <personalGridDataProvider></personalGridDataProvider>
   OR
   <webServiceGridDataProvider />
   <sortRule /> (zero or one)
   <filterExpression></filterExpression> (zero or one)
<gridRowModel>
```

<personalGridDataProvider>

```
<personalGridDataProvider>
   <record />  (zero or more)
</personalGridDataProvider>
```

<filterExpression>

```
<filterExpression> (Must contain just one of the following tags...)
   <and></and>
   <or></or>
   <fieldFilter />
</filterExpression>
```

<column>

```
<column>
   <filter /> (zero or more)
</column>
```

<and>

```
<and> (Must contain at least one of the following tags...)
   <fieldFilter />  (one or more)
   <or></or> (one or more)
</and>
```

<or>

```
<or> (Must contain at least one of the following tags...)
   <fieldFilter />  (one or more)
   <and></and> (one or more)
</or>
```

3.2 XML Tag Descriptions

This section describes the XML tags that you can use to configure the Chart Display Component.

Default attribute values

In the tables that follow, if an attribute is not required (Req? = 'N') and there is a default value specified, then not supplying the attribute is equivalent to setting the attribute to this default value. If an attribute is not required and the default is '(none)', then not supplying the attribute can result in one of two behaviors, depending on the particular attribute – either the behavior is as specified in the description column of the table, or there is no effect on the appearance or behavior of the component.

<and>

`<and>`

This tag represents the AND logical operator. It applies a logical AND operation to the tags that are its direct children. The children can be <fieldFilter> tags and <or> tags.

Attributes: This tag has no attributes.

<chartCreationDecorator>

`<chartCreationDecorator>`

Listens to events published by the rightClickDecorator, and adds in the requested chart as a floating panel.

Attributes: This tag has no attributes.

<column>

`<column>`

Defines a single column of a grid or grid template.

Attributes:

Name	Type	Default	Req?	Description
`cellRenderer`	string	'caplin.dom. renderer. TextElement Renderer'	N	Identifies the element renderer that renders data in the cells of this column. This attribute can either be set to the fully qualified name of a JavaScript class or the logical name of an element renderer. The logical name of an element renderer is defined by the <renderer> tag, which belongs to the XML schema described in Caplin Trader Client: Element Renderer XML Configuration Reference.

Name	Type	Default	Req?	Description
`clear`	boolean	false	N	Flag to indicate whether a column should clear the values it was displaying when the containing grid is re-shown after being hidden. When a grid is hidden (for example, when the tab for another grid in a panel is selected), the columns of the grid are not updated by the data provider until the grid is re-shown. Valid settings are "true" (clear previous values) and "false" (display previous values until they are updated by the data provider).
`cssClass`	string	(none)	N	Additional css class name for the column header. The header cell will have this css class in addition to the column id and "cell"
`displayName`	string	Value of id attribute	N	The text to be displayed in the column header.
`fields`	string	Value of id attribute	N	The names of the fields to be sent to the control renderer for display in this column. This is a comma separated list. The first field in the list is known as the "primary field", and is the field that is used to sort or filter the column (either in the XML configuration or when the end user sorts or filters the column). Also see the 'primaryFieldType' and 'sortOrder' attributes.
`fixed`	boolean	false	N	Flag to indicate whether or not a column is fixed in place. Fixed columns cannot be reordered, other columns cannot be reordered past a fixed column. Valid values are "true" (fixed-in-place) and "false" (reorderable).
`headerRenderer`	string	'default'	N	The logical name of the renderer for the header of the column. This is defined in a separate set of XML configuration for the renderers.
`id`	string	(none)	Y	An identifier for this column. This should be unique within the inheritance hierarchy of the grid, with the following exception. A child grid can use an id for one of its columns that is the same as an id used by one of it parents. In this situation, if the parent has defined a attribute of the column that the child has not defined, the child will inherit the attribute value from the parent.
`mandatory`	boolean	false	N	Flag to indicate whether or not a column must be present in a grid. Mandatory columns cannot be removed from a grid. Valid values are "true" (mandatory) and "false" (not mandatory).

Name	Type	Default	Req?	Description
`primaryFieldType`	string	number	N	The type of the primary field displayed by this column. A column can display more than one field, and the primary field is the first field in the comma separated list of fields defined by the 'fields' attribute. Valid values are "number", "text" and "tenor". If set to "number", then the column can be sorted and filtered by the numerical value of the primary field. If set to "text", then the column can be sorted and filtered by the alphabetical value of the primary field. If set to "tenor" then the column can be sorted and filtered by the numerical initial value of the primary field, falling back to an alphabetical sort for the remainder of the value (or if the value does not start with a number). This type is useful for sorting tenors as it correctly interprets M and Y suffixes (e.g. 3M, 2Y).
`resizable`	boolean	true	N	Flag to remove or add the resize functionality to a column. If set to true or omitted (fallback to default) the user is permitted to resize the column.
`sortable`	boolean	true	N	Flag to remove or add the sortable functionality to a column. If set to true or omitted (fallback to default) the user is permitted to sort the column.
`sortOrder`	string	SORT_NONE	N	When the grid is initially rendered, sort the grid by the primary field of this column (see 'fields' attribute). Valid values are: SORT_ASC (sort ascending), SORT_DESC (sort descending), SORT_NONE (do not sort using this column). A grid can only be sorted by one column, so only one column in the grid can have this attribute set to "SORT_ASC" or "SORT_DESC". If the grid is sorted using this attribute, do not sort the grid using the <sortRule> tag of <gridRowModel>.
`width`	integer	100	N	The default width of the column in pixels.

<columnDefinitions>

`<columnDefinitions>`

Contains the column definitions (<column>) for either a grid or a grid template. A grid or template inherits all the column definitions from its inheritance hierarchy. When the grid is first displayed some of these columns may not be shown; the displayedColumns attribute of <gridTemplate> or <grid> defines which of the columns are displayed in the grid by default, and the order they should initially appear in.

Attributes: This tag has no attributes.

<columnMenuDecorator>

`<columnMenuDecorator>`

The column menu decorator adds a drop down menu to each column that allows new columns to be inserted to the right hand side of the selected column, or existing columns to be removed. Specify a column menu decorator as an empty tag: <columnMenuDecorator />. This decorator is supplied with Caplin Trader Client.

Attributes:

Name	Type	Default	Req?	Description
`columnSorting`	boolean	false	N	A flag that determines whether the user is given the option to sort columns. Valid values are "true" and "false".

<columnReorderingDecorator>

`<columnReorderingDecorator>`

The column reordering decorator allows columns to be reordered by dragging their header to the new location. Specify the decorator as an empty tag: <columnReorderingDecorator />. This decorator is supplied with Caplin Trader Client.

Attributes: This tag has no attributes.

<columnResizingDecorator>

`<columnResizingDecorator>`

A decorator which allows the resizing of columns by adding a drag zone at the leftmost edge of columns.

Attributes: This tag has no attributes.

<columnSortDecorator>

`<columnSortDecorator>`

The column sort decorator cycles the grid through three sort orders each time a column header is double clicked. The sort cycle is: sort ascending, sort descending, and remove sorting. When the sort order is ascending or descending, the grid is sorted by the primary field of the column that is double clicked. The primary field is the first field in the list of fields defined by the 'fields' attribute of the <column> tag. The sort cycle restarts each time a different column header is double clicked. Specify this decorator as an empty tag: <columnSortDecorator />. This decorator is supplied with Caplin Trader Client.

Attributes: This tag has no attributes.

<dataProviderMapping>

`<dataProviderMapping>`

For each data provider (see <dataProviderMappings>), this maps the concrete name of the data provider class to a logical name. Only the logical names of data providers are serialized, not the concrete class names. This allows the class implementing a data provider to be swapped for a different one, without affecting the ability to restore previously saved grids using that particular data provider.

Attributes:

Name	Type	Default	Req?	Description
`className`	string	(none)	Y	The class that will be constructed each time the corresponding logical name is referred to.
`id`	string	(none)	Y	The logical name for the data provider. This is the name that is used for the data provider tag embedded in <gridRowModel> tags.

<dataProviderMappings>

`<dataProviderMappings>`

The list of data providers available within the application. Data providers allow disparate data sources to be made available within grids through a common JavaScript interface (caplin.grid.GridDataProvider).

Attributes: This tag has no attributes.

<decoratorMapping>

`<decoratorMapping>`

For each grid decorator (see <decoratorMappings>), this maps the concrete name of the grid decorator class to a logical name. Only the logical names of grid decorators are serialized, not the concrete class names. This allows the class implementing a grid decorator to be swapped for a different one, without affecting the ability to restore previously saved grids that use this decorator.

Attributes:

Name	Type	Default	Req?	Description
`className`	string	(none)	Y	The class that will be constructed each time the corresponding logical name is referred to.
`id`	string	(none)	Y	The logical name of the grid decorator. This is the name that is used for the decorator mapping tag embedded in the <decorators> tag.

<decoratorMappings>

`<decoratorMappings>`

The list of grid decorators available within the application. Grid decorators allow the functionality and/or look and feel of the grid to be augmented in some way. Use grid decorators to apply changes to appearance or behavior relating to the grid as a whole. You can also use element renderers to modify the appearance and behavior of particular cells and column headers within a grid; these are specified using the cellRenderer and headerRenderer attributes the of the <column> tag.

Attributes: This tag has no attributes.

<decorators>

`<decorators>`

Specifies the decorators that will be applied to a particular template (<gridTemplate>) or grid (<grid>). For example, a grid or template may include a <dragDecorator>, a <dropDecorator> and a <columnMenuDecorator>. The name of a decorator tag (such as <dragDecorator>) is the logical name of the decorator as specified in the id attribute of a <decoratorMapping>.

Attributes: This tag has no attributes.

<disconnectedDecorator>

`<disconnectedDecorator>`

The disconnected decorator displays a message to the end user when no connection to the server is present. This decorator is supplied with Caplin Trader Client.

Attributes:

Name	Type	Default	Req?	Description
`message`	string	Data unavailable at this time.	N	The message that will be displayed when the grid contains no data.

<dragDecorator>

`<dragDecorator>`

The drag decorator allows instruments to be dragged out of grids that have specified it in their XML configuration. This decorator is supplied with Caplin Trader Client.

Attributes:

Name	Type	Default	Req?	Description
`ddGroup`	string	(none)	Y	Identifies the drag-drop group that instruments in the grid belong to. Instruments in the grid can only be dropped into components that accept this identifier.

<dropDecorator>

`<dropDecorator>`

The drop decorator allows instruments to be dropped into grids that have specified it in their XML configuration. This decorator is supplied with Caplin Trader Client.

Attributes:

Name	Type	Default	Req?	Description
`ddGroup`	string	(none)	Y	Only instruments that belong to this drag-drop group can be dropped into the grid.

<exportGridDataToExcelDecorator>

`<exportGridDataToExcelDecorator>`

Use this decorator if you wish to add the functionality of exporting grid data to an MS Excel workbook. The result is that rows are removed and/or added to the underlying grid.

Attributes:

Name	Type	Default	Req?	Description
`eventName`	string	(none)	Y	The name of the event being subscribed to.
`eventNamespace`	string	(none)	Y	The namespace of the event being subscribed to. Events have the following format "namespace.channel.event".
`url`	string	(none)	Y	The web service that this decorator will call.

<fieldFilter>

`<fieldFilter>`

Defines an expression for filtering a single field; for example, "BidSize >= 1000", which selects only data where the value of the BidSize field is greater than or equal to 1000. Several <fieldFilter> tags can be combined with <or> and <and> tags to form filters containing logical expressions using "OR" and "AND"; see the <filterExpression> tag, the <and> tag, and the <or> tag. This type of filter is applied to the grid when the grid is initially rendered, and filters the grid in addition to any column filter defined using the <filter> tag.

Attributes:

Name	Type	Default	Req?	Description
`field`	string	(none)	Y	The name of the underlying field (within the grid row model) to which the field filter expression applies. When the grid row model is defined to use the data provider 'caplin.grid.RttpContainerGridDataProvider', the field is the name of an RTTP field.

Name	Type	Default	Req?	Description
`operator`	string	(none)	Y	The logical operator to apply to this field filter expression. The operators that can be used are: = (equals), != (not equal to), > (greater than), >= (greater than or equal to), < (less than), <= (less than or equal to), ~ (marks a regular expression as case sensitive), # (marks a regular expression as case insensitive).
`value`	string	(none)	Y	The filter value applying to this field filter expression.

<filter>

`<filter>`

Filters the grid by the primary field of the column when the grid is initially rendered (see the 'fields' attribute of the <column> tag for a description of the primary field). The filter is defined by specifying a filter type and value. Only rows that match this filter are displayed in the grid. This type of filter is applied in addition to any filters that are defined using the <fieldFilter> tag.

Attributes:

Name	Type	Default	Req?	Description
`type`	string	(none)	Y	The type of filter to apply. Valid values are: "LESS_THAN" (less than), "GREATER_THAN" (greater than), "EXACT_MATCH" (exact match), "PARTIAL_MATCH" (partial match), "WILDCARD" (match any case), "WILDCARD_CASE_SENSITIVE" (case sensitive), "REGULAR_EXPRESSION" (case sensitive), "REGULAR_EXPRESSION_CASE_INSENSITIVE" (match any case).
`value`	string	(none)	Y	The value of the primary field to be filtered. For example, to display only rows where the primary field of the column is greater than 5, set type to "GREATER_THAN" and value to "5".

<filterExpression>

`<filterExpression>`

Contains a logical filter expression (as a combination of <fieldFilter>, <and>, and <or> tags) that is applied to the grid row model. A filter expression allows you to restrict the number of rows that are displayed in the grid. For example, the filter expression "(BidSize >= 1000) AND (AskSize < 1015)" selects only those rows where the BidSize field has a value of 1000 or more and the AskSize field has a value below 1015.

The <filterExpression> tag can only contain one direct child tag; filter expressions containing multiple <fieldFilter> tags are defined by nesting the <fieldFilter> tags within <and> and <or> tags. See the examples shown earlier in this document.

If a filter expression is defined in a parent <grid> or <gridTemplate>, any grids that inherit from this parent also inherit the filter expression. However, a filter expression that is defined in a child grid overrides any filter expressions in grids and grid templates that it inherits from.

The data provider that populates a grid with data can be an RTTP container; that is, an instance of the JavaScript class 'caplin.grid.RttpContainerGridDataProvider' defined in a <dataProviderMapping>. In this case, the filter expression is converted to a request for a filtered RTTP container and is sent to Liberator. Liberator returns a container with the filtered data in it and the Client displays this data in the grid.

Attributes:

Name	Type	Default	Req?	Description
`name`	string	default	N	This attribute is required if more than one <filterExpression> is defined in XML, and must uniquely identify the filter expression from all other filter expressions that you define. The name is not used in any XML configuration, but can be used in JavaScript code to identify the filter expression. The way that JavaScript code refers to a filter expression is discussed in the Caplin Trader Client: API Reference documentation ('caplin.grid' package).

<folder>

`<folder>`

A folder contains a collection of grids (<grid>) and/or more internal folders.

Attributes:

Name	Type	Default	Req?	Description
`name`	string	(none)	Y	The name of the folder as displayed on the screen (for example, when the folder is shown in a dialog).

<grid>

`<grid>`

Defines a grid that can be included in a layout. Also see <gridTemplate>.

Attributes:

Name	Type	Default	Req?	Description
`baseGrid`	string	(none)	N	The id of a grid, if there is one, that this grid inherits from. If a baseTemplate attribute is also present it will override the baseGrid setting.
`baseTemplate`	string	(none)	N	The id of a template, if there is one, that this grid inherits from. If a baseGrid attribute is also present the baseTemplate attribute will override it.
`displayName`	string	Value of id attribute	N	The name for the grid that appears in the Insert dialog when the end user chooses to insert the grid in a layout.
`displayedColumns`	string	Inherited from parent	N	The columns to display in the grid by default (for example, when the grid is first displayed), and the order they should initially appear in. The columns defined within the <columnDefinitions> tag specify all the available columns within a grid, but not the default set of columns initially presented to the end user. If a grid does not specify a displayedColumns attribute, it will inherit the value from its parent. A value must be specified somewhere in the inheritance hierarchy of the grid.
`id`	string	(none)	Y	The unique identifier for this grid. Grid identifiers must be unique across all grids and grid templates (the namespace of a grid is not distinguished by the folder it lives in). This allows grids to be moved, or folders to be renamed without breaking pre-existing serialization.

<gridDefinitions>

`<gridDefinitions>`

The outermost tag of the grid definition XML.

Attributes: This tag has no attributes.

<gridRowModel>

`<gridRowModel>`

Contains information about the type of row model and data provider that populates the grid. The grid data provider is specified using one of the <rttpContainerGridDataProvider>, <personalGridDataProvider>, or <webServiceDataProvider> tags. A grid must have a <gridRowModel> defined for it, either in its <grid> tag, or in a <grid> or <gridTemplate> from which it inherits.

Attributes:

Name	Type	Default	Req?	Description
`groupBy`	string	(none)	N	To group rows of the grid when the grid is initially displayed, specify the field that you want to group the rows by. For row grouping to render correctly, the rows must also be ordered by the same field that they will be grouped on. Rows can either be ordered at the DataSource, or by specifying a <sortRule> tag in the grid XML configuration. Rows in the grid will have a blank row at the head of each group unless a group header decorator is included in the XML configuration. A group header is specified using the <groupHeaderDecorator> tag. If the value of this attribute is set to 'none' then no group-by atribute will be set, including groupBy attributes that might otherwise have been inherited.
`serializeFilters`	string	true	N	A flag that determines whether filters applied by the end user are serialized when the grid is serialized. A grid is serialized when the end user saves the layout containing the grid. Valid values are "true" (serialize user applied filters) and "false" (do not serialize user applied filters).
`transformMode`	string	(none)	N	When a transform (grouping, filter, sort, or combination of the three) is applied to a grid the transform mode controls how updates to the data are pushed out from the DataSource. Valid values are: snapshot - a snapshot of the data is taken at the point that the transform is applied no updates to the contents of the grid are reflected in the client, e.g. rows that should move in/out of the filtered view do not do so, sorting order does not change. live - changes to the contents of the grid are pushed to the client continuously, with the correct transforms applied, so row order will update live to match values, and rows can pop in and out of a filter. Note that this attribute is not required - if it is not present then 'snapshot' is assumed.

<grids>

`<grids>`

Contains the definitions of the grids users can add to their layouts. Grids can be defined within a hierarchical folder structure so users can more easily find them (see <folder>).

Attributes: This tag has no attributes.

<gridTemplate>

`<gridTemplate>`

A grid template is an abstract grid definition. Multiple physical grids (<grid>) can be defined using the same template; the physical grids inherit the characteristics of the template. This allows various aspects of the physical grids, such as the default displayed columns, column names, and column widths, to be reconfigured simply by modifying the template(s) associated with the grid. A grid template can be based on (inherit from) another grid template. Grid templates do not appear in the list of available grids when the user chooses to insert a new grid into their layout.

Attributes:

Name	Type	Default	Req?	Description
`baseTemplate`	string	(none)	N	The <gridTemplate>, if there is one, that this template inherits from.
`displayedColumns`	string	Inherited from parent	N	The columns to display in the grid by default (for example, when the grid is first displayed), and the order they should initially appear in. The columns defined within the <columnDefinitions> tag specify all the available columns within a grid, but not the default set of columns initially presented to the end user. If a <gridTemplate> does not specify a displayedColumns attribute, it will inherit the value from its parent, unless the template has no parent, in which case the attribute is undefined. Note that a value for displayedColumns must be explicitly specified somewhere in the inheritance hierarchy of a grid.
`id`	string	(none)	Y	The identifier of this grid template. The identifier must be unique across all grids (<grid>) and grid Templates.

<groupHeaderDecorator>

`<groupHeaderDecorator>`

The group header decorator inserts a header row before each set of grouped rows when the rows in a grid are grouped by field name (see the 'groupBy' attribute of the <gridRowModel> tag). Only use this decorator if the rows in a grid are grouped. This decorator is supplied with Caplin Trader Client.

Attributes:

Name	Type	Default	Req?	Description
`headerTextField`	string	(none)	Y	The name of the field containing the group header text.

<hoverDecorator>

`<hoverDecorator>`

The hover decorator changes the appearance of a row when the end user hovers over the row with their mouse. The styling that is applied by the hover decorator is defined in CSS (not discussed in this document). Specify a hover decorator as an empty tag: <hoverDecorator />. This decorator is supplied with Caplin Trader Client.

Attributes: This tag has no attributes.

<legacyGrid>

`<legacyGrid>`

Identifies a legacy grid that can be inserted in a layout. Legacy grids are defined using a serialized format rather than XML.

Attributes:

Name	Type	Default	Req?	Description
`displayName`	string	(none)	Y	The name of the grid that appears in the Insert dialog when the end user chooses to insert the grid in a layout.
`src`	string	(none)	Y	The URL that returns the serialized grid definition.

<loadingDataDecorator>

`<loadingDataDecorator>`

The loading data decorator displays a spinning wait icon while the grid waits for data to display from the grid data provider. This decorator is supplied with Caplin Trader Client.

Attributes:

Name	Type	Default	Req?	Description
`showInitialLoad`	boolean	true	N	A flag that determines whether the spinning icon will be displayed the first time the grid is rendered. Valid values are "true" and "false". Displaying the spinning icon will slow down the time it takes to initially display pages that contain grids.

<noDataFoundDecorator>

`<noDataFoundDecorator>`

The no data found decorator displays a message to the end user when a grid contains no data. This decorator is supplied with Caplin Trader Client.

Attributes:

Name	Type	Default	Req?	Description
`message`	string	'No data available'	N	The message that will be displayed when the grid contains no data.

<openAjaxFiltersDecorator>

`<openAjaxFiltersDecorator>`

Listens to subjectAdded and subjectRemoved events published over the OpenAjax Hub. The result is that rows are removed and/or added to the underlying grid.

Attributes: This tag has no attributes.

<or>

`<or>`

This tag represents the OR logical operator. It applies a logical OR operation to the tags that are its direct children. The children can be <fieldFilter> tags and <and> tags.

Attributes: This tag has no attributes.

<personalGridDataProvider>

`<personalGridDataProvider>`

Defines a data provider that populates a personal grid. A personal grid is initially unpopulated, but can be populated by the end user with instruments of their choice (for example, by dragging and dropping instruments from other grids). Instruments that are added to a personal grid can also be deleted from the grid. The personal grid data provider creates an RTTP container to hold the instruments that are currently in the grid. This data provider uses the 'rttpContainerGridDataProvider' to communicate with a Liberator server (see < rttpContainerGridDataProvider>).

Attributes: This tag has no attributes.

<record>

`<record>`

A pre-populated instrument within the personal grid, that user's can add to if they decide.

Attributes:

Name	Type	Default	Req?	Description
`instrument`	string	(none)	Y	The RTTP subject name of the instrument.

<removeRowDecorator>

`<removeRowDecorator>`

The remove row decorator displays the message "deleting..." to indicate that a row is being deleted from a personal grid. The message replaces the row data. When the end user deletes a row from a personal grid, the grid data provider sends a request to the Liberator server to remove the row from the RTTP container. The message "deleting..." is only displayed until the row has been removed from this container. The remove row decorator does not need to be specified in order to enable the deletion of rows; it merely improves the responsiveness of the grid during the removal operation. Specify a remove row decorator as an empty tag: <removeRowDecorator />. This decorator is supplied with Caplin Trader Client.

Attributes: This tag has no attributes.

<requestDataOnCurrencyChangeDecorator>

`<requestDataOnCurrencyChangeDecorator>`

Listens to currencyChanged event published over the OpenAjax Hub. The result is that the existing data is cleared and replaced with data for the new currency.

Attributes:

Name	Type	Default	Req?	Description
`subject`	string	(none)	Y	Subject that will be used to make an initial request.

<resetColumnsDecorator>

`<resetColumnsDecorator>`

The column reset decorator places a button at the right hand side of the rightmost column header. When this button is clicked, every column filter currently applied to the grid is removed. Specify this decorator as an empty tag: <resetColumnsDecorator />. This decorator is supplied with Caplin Trader Client.

Attributes: This tag has no attributes.

<rightClickDecorator>

`<rightClickDecorator>`

Adds a right click context menu to the grid. This decorator is defined in caplinx.grid.decorator. RightClickMenuDecorator and currently includes two options to add in a benchmark or a timeseries chart.

Attributes: This tag has no attributes.

<rowLoadingDecorator>

`<rowLoadingDecorator>`

The row loading decorator displays a loading image while rows are waiting for their data to come back from the grid data provider. This decorator is supplied with Caplin Trader Client.

Attributes:

Name	Type	Default	Req?	Description
`imagePath`	string	(none)	Y	The path to the image to be shown while loading

<rttpContainerGridDataProvider>

`<rttpContainerGridDataProvider>`

Defines a data provider that fills a grid with the data from a named RTTP container. (The RTTP container receives the data from a Liberator server, via a streaming RTTP connection established through StreamLink.)

Attributes:

Name	Type	Default	Req?	Description
`container`	string	(none)	Y	The RTTP container that will be requested. For example, container="/CONTAINER/FX/Major".

<scrollTipDecorator>

`<scrollTipDecorator>`

The scroll tip decorator provides a visual indication about the number of rows that a user has scrolled through. When scrolling through a large grid, only some of the available rows can be displayed. The scroll indicator is placed at the bottom right hand side of the grid and shows the number of the row that is currently at the bottom of the grid, followed by the total number of available rows. Specify a scroll tip decorator as an empty tag: <scrollTipDecorator />. This decorator is supplied with Caplin Trader Client.

Attributes: This tag has no attributes.

<selectionDecorator>

`<selectionDecorator>`

The selection decorator highlights a row when the end user clicks the row. Specify a selection decorator as an empty tag: <selectionDecorator />. This decorator is supplied with Caplin Trader Client.

Attributes:

Name	Type	Default	Req?	Description
`persistSelection`	boolean	false	N	If the row selection is not to be removed until another row in the same grid is clicked set this to true

<sortRule>

`<sortRule>`

Sorts the grid in ascending or descending order when the grid is initially rendered. A grid can be sorted when it is initially rendered, either by defining this tag in the grid XML configuration or by setting the 'sortOrder' attribute of the <column> tag. Only use one of these XML configuration options to sort a grid, not both.

Attributes:

Name	Type	Default	Req?	Description
`field`	string	(none)	Y	The name of the field that will be used to sort the grid. The field can be any field returned by the data provider of the grid row model (see <gridRowModel>), and does not need to be a field in a column (see the 'fields' attribute of the <column> tag).
`order`	string	ascending	N	The order in which the grid will be sorted. Valid values are "ascending" and "descending".

Name	Type	Default	Req?	Description
`type`	string	number	N	The type of field to be sorted. Valid values are "number", "text" and "tenor". Fields of type "number" are sorted numerically, and fields of type "text" are sorted alphabetically. Fields of type "tenor" are sorted numerically if the field value starts with a number or tenor, alphabetically otherwise. Note that tenors are sorted as expected (e.g. 12M = 1Y) Fields containing prices or ISO formatted dates must be set to "number".

<templates>

`<templates>`

Contains a list of grid templates (<gridTemplate>).

Attributes: This tag has no attributes.

<webServiceGridDataProvider>

`<webServiceGridDataProvider>`

Defines a data provider that fills a grid with data from a web service.

Attributes:

Name	Type	Default	Req?	Description
`format`	string	(none)	Y	The format of the data delivered by the web service. At present, this must be set to "json".
`url`	string	(none)	Y	The URL of the web service. For example, url="/services/static-data/ historicalSnapshotJson.jsp".

CAPLIN THE REAL TIME WEB COMPANY

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