CAPLIN

Caplin Trader Client 1.3

Grid Configuration XML Reference

December 2008

CONFIDENTIAL

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 1.3.

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.

  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 thoughts to documentation@caplin.com.

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:

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

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.

3 Getting Started

The following sections explain how the various XML tags may be combined to define grids for Caplin Trader Client.

3.1 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.

3.2 Using the XML configuration markup

An example configuration file

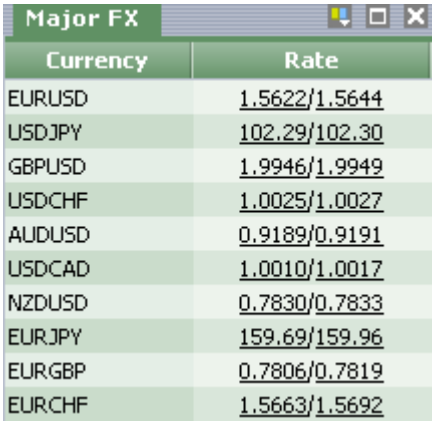
An example XML file describing a simple grid configuration is shown below.

The XML Reference information 16 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 13..

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="textfilter" />
      </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="textfilter"`).

– 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="textfilter" />
      </columnDefinitions>
      <gridRowModel>
         <rttpContainerGridDataProvider container="/CONTAINER/FX/Minor" />
      </gridRowModel>
   </grid>
</grids>
```

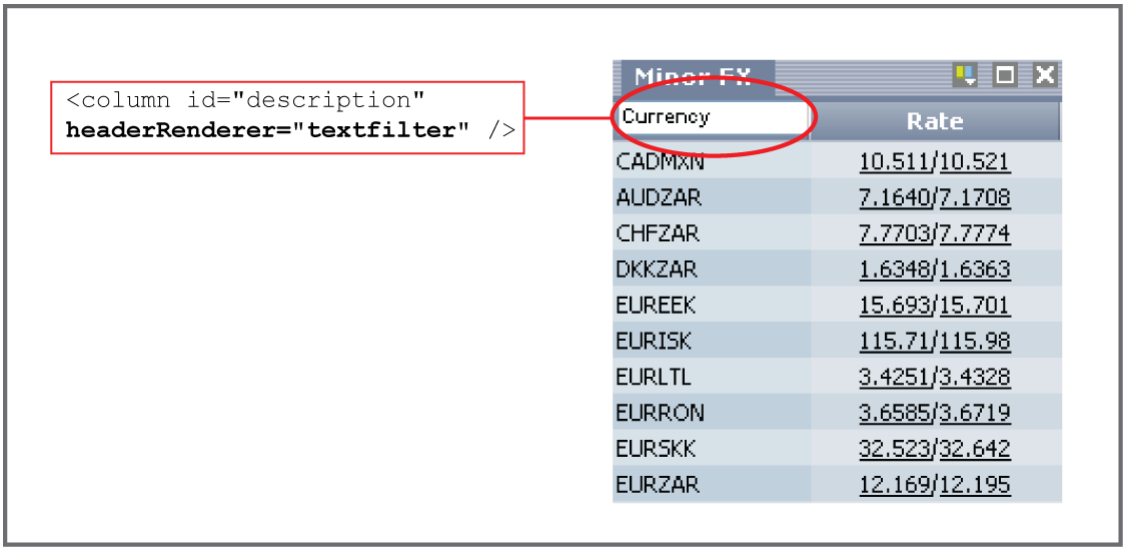
Each grid is defined in a **`<grid>`** tag. In this example there are two grids, "FX.Major" and "FX.Minor", each of which inherits its characteristics from the "FX" grid template. So both grids display description and rate information in columns of the same widths. To change these aspects of the appearance 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="textfilter" />
  </columnDefinitions>
```

The "description" column has a header renderer (`headerRenderer="textfilter"`). The "textfilter" 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).



Effect of applying a textfilter 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.

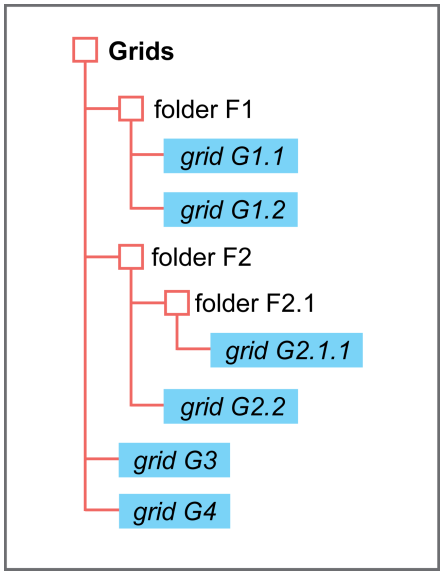
Folders

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

Grid filters

A grid definition 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>`.

Here are some examples.

This XML fragment specifies the filter: BidSize >= 1000

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

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

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

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

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))

```
<grid>
  ...
  <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>
</grid>
```

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"`.

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 the XML Reference information 16 section.

<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>
```

<gridTemplate>

```
<gridTemplate>
   <decorators></decorators> (zero or one)
   <columnDefinitions></columnDefinitions> (zero or one)
   <gridRowModel></gridRowModel> (zero or one)
   <filterExpression></filterExpression> (zero or one)
</gridTemplate>
```

<gridRowModel>

```
<gridRowModel> (Must contain just one of the following tags...)
   <rttpContainerGridDataProvider />
   <someOtherGridDataProvider />
<gridRowModel>
```

<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>
```

<folder>

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

<grid>

```
<grid>
   <decorators></decorators> (zero or one)
   <columnDefinitions></columnDefinitions> (zero or one)
   <gridRowModel></gridRowModel> (zero or one)
   <filterExpression></filterExpression> (zero or one)
</grid>
```

<decorators>

```
<decorators> (Must contain at least one of the following tags...)
   <dragDecorator />
   <dropdecorator />
   <columnMenuDecorator />
</<decorators>
```

<columnDefinitions>

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

<filterExpression>

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

<and>

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

<or>

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

4 XML Reference information

The following sections describe the Grid Configuration 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 the runtime appearance/behavior will be according to the default value of the attribute.

If a table only has a heading row, there are no attributes for that tag.

4.1 <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.

4.2 <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	The name of the JavaScript class that renders the data in the cells of this column.
`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. This is a comma separated list.
`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.

Name	Type	Default	Req?	Description
`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).
`width`	integer	100	N	The default width of the column in pixels.

4.3 <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.

4.4 <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: This tag has no attributes.

4.5 <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.

4.6 <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.

4.7 <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.

4.8 <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.

4.9 <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.

4.10 <dragDecorator>

`<dragDecorator>`

The drag decorator allows instruments to be dragged out of grids that have specified it in their XML configuration. Specify a drag decorator as an empty tag: <dragDecorator />. 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.

4.11 <dropDecorator>

`<dropDecorator>`

The drop decorator allows instruments to be dropped into grids that have specified it in their XML configuration. Specify a drop decorator as an empty tag: <dropDecorator />. 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.

4.12 <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.

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.
`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.

4.13 <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: This tag has no attributes.

4.14 <folder>

`<folder>`

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

Attributes:

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

4.15 <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.

4.16 <gridDefinitions>

`<gridDefinitions>`

The outermost tag of the grid definition XML.

Attributes: This tag has no attributes.

4.17 <gridRowModel>

`<gridRowModel>`

Contains information about the type of row model used in the grid. At present there is only one type of row model (defined by <rttpContainerGridDataProvider>). 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: This tag has no attributes.

4.18 <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.

4.19 <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.

4.20 <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.

4.21 <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.

4.22 <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".

4.23 <someOtherGridDataProvider>

`<someOtherGridDataProvider>`

This is a placeholder for other data providers that can populate the rows of a grid. The XML tag name for a newly defined data provider must be the same as the name of the JavaScript class implementing the data provider. The attributes of the new tag are determined by the functionality of the data provider.

Attributes:

Name	Type	Default	Req?	Description
`foo`	string	(none)	Y	

4.24 <templates>

`<templates>`

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

Attributes: This tag has no attributes.

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