CAPLIN

Caplin Trader Client 1.5

Element Renderer Configuration XML Reference

October 2009

CONFIDENTIAL

Contents

1 Preface

1.1 What this document contains

This reference document describes the XML-based configuration that defines Element Renderers in Caplin Trader Client.

The information in this document applies to Caplin Trader version 1.5.

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 Element Renderers in Caplin Trader Client.

1.3 Related documents

- **Caplin Trader Client: Grid XML Configuration Reference**

  Describes the XML-based configuration that defines the layout and functionality of the grids displayed in Caplin Trader Client.

- **Caplin Trader Client: API Reference**

  The API reference documentation provided with Caplin Trader Client. The classes and interfaces of this API allow you to extend the capabilities of Caplin Trader Client.

- **Ext JS API reference documentation**

  This document describes the API of the third party "Ext JS" component framework. In particular, it specifies the Ext Date Tokens that configure the caplin.element.formatter.DateFormatter 25.

1.4 Typographical conventions

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

Type	*Uses*
aMethod	Function or method name
aParameter	Parameter or variable name
/AFolder/Afile.txt	File names, folders and directories
`Some code;`	Program output and code examples
The `value=10` attribute is...	Code fragment in line with normal text
Some text in a dialog box	Dialog box output
`Something typed in`	User input – things you type at the computer keyboard
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.

2 Introduction to the Element Renderer Framework

Caplin Trader Client uses Element Renderers to render data within display components such as grids, trees, trade tickets and trade tiles. An Element Renderer instance binds data from a model to a visual control on the screen. This data may be real-time prices or static information from a variety of sources. An Element Renderer configuration is defined using XML definitions, and instances are created by the Element Renderer Framework at runtime, as required by the display components.

Element Renderers allow you to specify the format (for example, number of decimal places) and style (for example, underlined or bold) of the data that is rendered in a control. In this way the visual appearance of the rendered data can reflect the state of the data, so that an end user can readily determine how reliable a price is (for example, whether a price is current or stale, or if a price has just been updated).

Element Renderers also allow you to specify event handlers that respond to events on a control (for example, to open a trade ticket when an end user clicks on an indicative price). Finally, Element Renderers allow you to specify input controls that accept data entered into the control by the end user.

Here is an example of a grid that is displaying indicative prices for four FX currency pairs.

Grid displaying indicative prices for FX currency pairs.

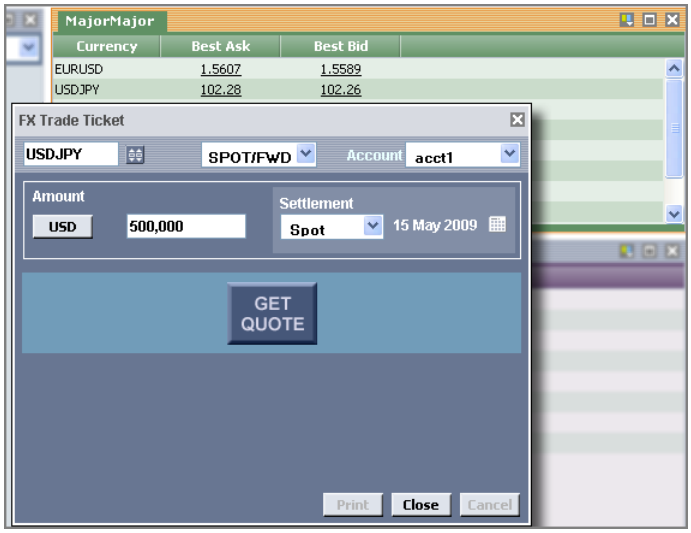
In the example above, three columns are displayed in the grid. The fields of the Currency column are text controls displaying currency pairs, while the fields of the Best Bid and Best Ask columns are text controls displaying indicative "Best Bid" and "Best Ask" prices for these currency pairs.

The Element Renderer for the text controls in the Best Bid and Best Ask columns is configured to:

- Render stale prices with a strike through.
- Flash prices with a green background for half a second when the indicative price increases.
- Flash prices with a red background for half a second when the indicative price decreases.

The prices in the first two rows of this grid are current, the prices in the third row are stale (strike through), and the prices in the fourth row are in the process of being updated (they have a green background because the price has just increased).

The Element Renderer in this example is also configured to open a trade ticket for a currency pair when the end user clicks the indicative price for that currency pair.



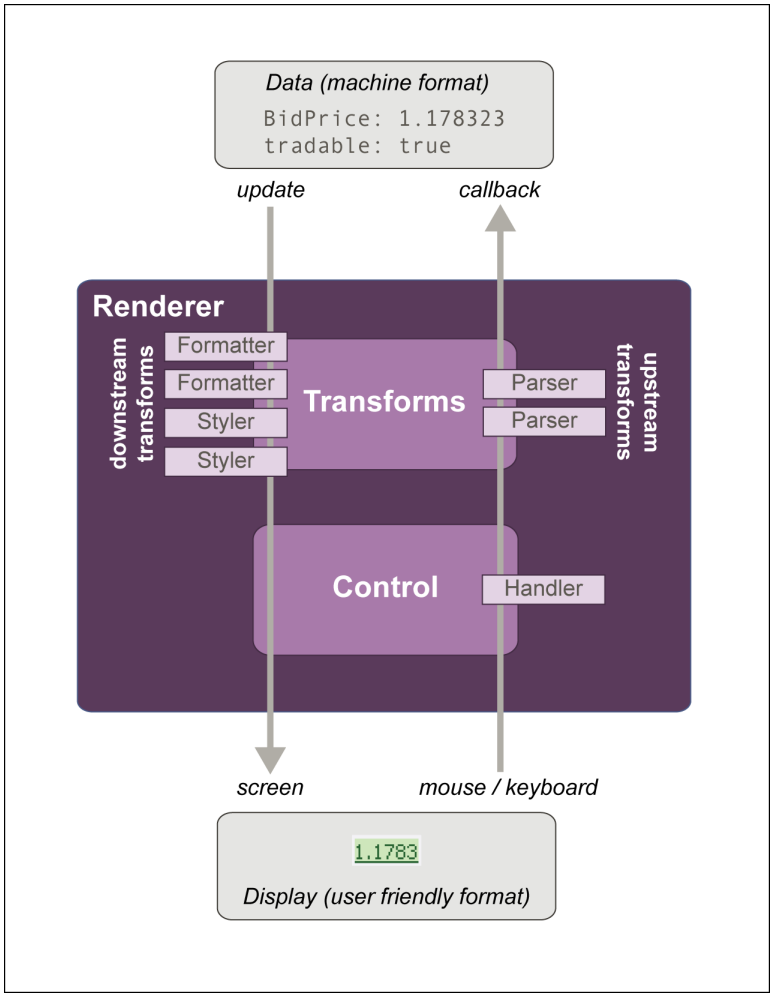
Trade ticket opens when the end user clicks an indicative price

You can define an Element Renderer using the XML configuration defined in this document. The reference implementation of Caplin Trader Client is supplied with ready made formatters, stylers, parsers, and event handlers that you can use in your Element Renderer XML configuration. There is also a well defined JavaScript API (see the **Caplin Trader Client: API Reference**), with suitable extension points, that allow you to create your own formatters, stylers, parsers, and event handlers (see Glossary of terms and acronyms 38 for a description of these terms).

At present you can only easily apply Element Renderers in grids, but in future releases of Caplin Trader Client they will be easier to integrate into trade tickets, trade tiles, and custom display components.

2.1 Schematic view of an Element Renderer

The following diagram shows the component parts of a typical Element Renderer.



A typical Element Renderer

The four downstream transforms in this example (two formatters and two stylers) transform a price in machine format to a format suitable for displaying on the screen. The two upstream transforms (parsers) would typically transform data entered by the end user (such as a date) to a format suited to machine processing. The control handler responds to mouse and keyboard events, such as when the end user clicks a displayed price.

3 Constructing an Element Renderer

The following sections explain how the various XML tags may be combined to define Element Renderers in 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 An example renderer configuration

An example XML file describing an Element Renderer is shown below.

The XML Reference information 14 section defines the XML tags and attributes you can use to define an Element Renderer. Also see the section Ordering and nesting of tags 12.

XML that configures an Element Renderer

```
<rendererDefinitions>
  ...
  <renderer type="fx-price">
     <control type="caplin.control.basic.TextControl">
       <handler name="caplinx.element.handler.TradeOnClickHandler"/>
     </control>
     <downstream>
       <transform name="caplin.element.formatter.NullValueFormatter">
         <attribute name="nullValue" value=""/>
       </transform>
       <transform name="caplin.element.styler.FlashStyler"
         <attribute name="duration" value="500"/>
         <attribute name="color-up" value="#286221"/>
         <attribute name="color-down" value="#841819"/>
         <attribute name="backgroundColor-up" value="#cdefbd"/>
         <attribute name="backgroundColor-down" value="#feb3aa"/>
       </transform>
       <transform name="caplinx.element.styler.PriceStyler"
         <attribute name="recordStatus" value="${RTTP.RECORD_STATUS}" />
         <attribute name="tradableState" value="${TRADABLE}" />
         <attribute name="class-tradable" value="tradablePrices" />
         <attribute name="class-stale" value="stale" />
         <attribute name="class-tradablestale" value="tradablestale" />
       </transform>
     </downstream>
  </renderer>
</rendererDefinitions>
```

In this configuration, the Element Renderer transforms downstream data by applying a null value formatter, a flash styler, and a price styler to the rendered data. Downstream data is data that is entered into a control by the application (such as data from a Liberator server), while upstream data is data that is entered into a control by the end user. This renderer does not need to transform upstream data because the text control (`caplin.control.basic.TextControl`) only displays downstream data, and does not accept data entered by the end user.

Grid displaying 'Best Bid' and 'Best Ask' prices that have been transformed by an element renderer.

Tip: In the reference implementation of Caplin Trader Client, Element Renderers are defined in the file *apps/webapps/caplintrader/applications/CaplinTrader/conf/rendererDefinitions.jsp*.

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:

- **`<rendererDefinitions>`** starts the renderer definitions.

```
<rendererDefinitions>
  ...
  <renderer type="fx-price">
    ...
  </renderer>
</rendererDefinitions>
```

In this case only one renderer is defined (`type="fx-price"`).

- **`<renderer>`** contains the definition of a single renderer.

```
<renderer type="fx-price">
   <control type="caplin.control.basic.TextControl">
     ...
   </control>
   <downstream>
     ...
   </downstream>
</renderer>
```

In this case the renderer consists of a `<control>` and the `<downstream>` transforms that transform the data rendered in the control.

- **`<control>`** identifies the type of control in which data is rendered.

```
   <control type="caplin.control.basic.TextControl">
     <handler name="caplinx.element.handler.TradeOnClickHandler"/>
   </control>
```

The `type` attribute identifies the fully qualified name of the JavaScript class that creates the control. In this case the control is a text control, which simply displays data on the screen (for example, indicative instrument prices in the cells of a grid column).

The `<handler>` tag identifies the fully qualified name of the JavaScript class that responds to events on the text control. In this case the handler opens a trade ticket (`TradeOnClickHandler`) if the end user clicks on a displayed price.

- **`<downstream>`** contains the downstream transforms that are applied to data in the control.

```
   <downstream>
     ...
     <transform ...>
       ...
     </transform>
   </downstream>
```

Downstream transforms are applied to data provided by the application (such as data from a Liberator server).

- **`<transform>`** defines a single transform that can be applied to data in a control.

```
<transform name="caplin.element.formatter.NullValueFormatter">
  <attribute name="nullValue" value=""/>
</transform>
<transform name="caplin.element.styler.FlashStyler"
  <attribute name="duration" value="500"/>
  <attribute name="color-up" value="#286221"/>
  <attribute name="color-down" value="#841819"/>
  <attribute name="backgroundColor-up" value="#cdefbd"/>
  <attribute name="backgroundColor-down" value="#feb3aa"/>
</transform>
<transform name="caplinx.element.styler.PriceStyler"
  <attribute name="recordStatus" value="${RTTP.RECORD_STATUS}" />
  <attribute name="tradableState" value="${TRADABLE}" />
  <attribute name="class-tradable" value="tradablePrices" />
  <attribute name="class-stale" value="stale" />
  <attribute name="class-tradablestale" value="tradablestale" />
</transform>
```

The `name` attribute of the `<transform>` tag identifies the fully qualified name of the JavaScript class that transforms the data in the control. Each `<transform>` also contains one or more child `<attribute>` tags, each containing `name`/`value` pairs that configure the properties of the transform.

In this case three transforms are defined.

1. `<transform name="caplin.element.formatter.NullValueFormatter">`

A formatter that defines how null values are displayed. The formatter is configured by a single `name`/`value` pair as shown in the table below.

Formatter configuration (NullValueFormatter)	Description
`<attribute name="nullValue" value=""/>`	In this case null values are not displayed (`value=""`).

2. `<transform name="caplin.element.styler.FlashStyler">`

A styler that changes the appearance of the displayed value (by giving the appearance of a flashing value) when the data value increases or decreases. The styler is configured by `name`/`value` pairs as shown in the table below.

Styler configuration (FlashStyler)	Description
`<attribute name="duration" value="500"/>`	The number of milliseconds for which the appearance of the displayed value changes.
`<attribute name="color-up" value="#286221"/>`	The applied foreground color when the value increases.
`<attribute name="color-down" value="#841819"/>`	The applied foreground color when the value decreases.
`<attribute name="backgroundColor-up" value="#cdefbd"/>`	The applied background color when the value increases.
`<attribute name="backgroundColor-down" value="#feb3aa"/>`	The applied background color when the value decreases.

3. `name="caplinx.element.styler.PriceStyler"`

A styler that changes the appearance of the displayed value (using CSS classes), depending on the tradable state of the data. The styler is configured by `name`/`value` pairs as shown in the table below.

Styler configuration (PriceStyler)	Description
`<attribute name="recordStatus" value="${RTTP.RECORD_STATUS}" />`	The status of the data (stale or not stale). In this case the field "`RTTP.RECORD_STATUS`" holds the status of the data. Note that a value can be obtained from a field if `value` is defined using the notation `value="${FIELD_NAME}"`.
`<attribute name="tradableState" value="${TRADABLE}" />`	The tradable state of the instrument that the data represents (tradable or not tradable). In this case the "`TRADABLE`" field holds the tradable state of the instrument.
`<attribute name="class-tradable" value="tradablePrices" />`	The CSS class to apply when the data is not stale and the instrument is tradable.
`<attribute name="class-stale" value="stale" />`	The CSS class to apply when the data *is* stale and the instrument is *not* tradable.
`<attribute name="class-tradablestale" value="tradablestale" />`	The CSS class to apply when the data *is* stale but the instrument is tradable.

In the reference implementation of Caplin Trader Client, CSS files are located in the directory *apps/webapps/caplintrader/applications/CaplinTrader/source/styles*. The CSS classes used by the price styler are defined in the file *product-grids.css*.

3.3 Placing a renderer in a grid

To define the renderer that will render data in the cells of a grid column, set the `cellRenderer` attribute of the column to the name of the renderer in the grid XML configuration. In the example below, all grids that inherit from the FX grid template (`gridTemplate id="FX"`) will have `"Best Bid"` and `"Best Ask"` columns that use the `"fx-price"` renderer.

```
<templates>
...
  <gridTemplate id="FX">
    <decorators>
      ...
    </decorators>
    <columnDefinitions>
      ...
      <column id="bestbid"
              cellRenderer="fx-price"
              fields="BestBid"
              displayName="Best Bid"
              width="100"/>
      <column id="bestask"
              cellRenderer="fx-price"
              fields="BestAsk"
              displayName="Best Ask"
              width="100"/>
  </gridTemplate>
  ...
</templates>
```

For further information about grid inheritance, see the description of the `<gridTempalte>` tag in the document **Caplin Trader Client: Grid XML Configuration Reference**.

4 XML Reference information

This is the XML reference information for the Element Renderer.

4.1 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 (*rendererDefinitions.rnc*) for definitive information on the ordering and nesting of tags.

For a description of each tag and its attributes, see XML tag descriptions 14.

<rendererDefinitions>

This is the outermost tag
`<rendererDefinitions>`
`<renderer></renderer>` (one or more)
`</rendererDefinitions>`

<renderer>

`<renderer>`
`<template></template>`
OR
`<control></control>`
`<downstream></downstream>` (one or more)
AND/OR
`<upstream></upstream>` (one or more)
`</renderer>`

<template>

`<template>`
`<anyHTML>` (zero or more)
`<renderer></renderer>` (zero or more)
`</template>`

<anyHTML>

`<anyHTML>`
`<anyHTML>` (zero or more)
`<renderer></renderer>` (zero or more)
`</anyHTML>`

<control>

```
<control>
   <handler></handler> (one or more)
</control>
```

<downstream>

```
<downstream>
   <transform></transform> (one or more)
</downstream>
```

<upstream>

```
<upstream>
   <transform></transform> (one or more)
</upstream>
```

<handler>

```
<handler>
   <attribute /> (zero or more)
</handler>
```

<transform>

```
<transform>
   <attribute /> (zero or more)
</transform>
```

4.2 XML tag descriptions

This is the reference information for the configuration XML that describes the Element Renderer.

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.

<anyHTML>

`<anyHTML>`

A child of <template>, any HTML tag can be used in the definition of a renderer. For example, the HTML tag could contain two <renderer> tags in the definition of a composite renderer.

Attributes: This tag has no attributes.

<attribute>

`<attribute>`

A name/value pair that configures one property of a <control>, <handler>, or <transform>. The JavaScript classes that implement these objects, and the properties that can be configured using name/value pairs, are described in the subsections of "Element Renderer JavaScript classes".

Attributes:

Name	Type	Default	Req?	Description
`default`	string	(none)	N	The default value of the property if the value is derived from a field and the field has no value (see 'value' attribute).
`name`	string	(none)	Y	The name of the property being configured.
`value`	string	(none)	Y	The value of the property. The value can include a named field if the name of the field is placed inside braces preceded by '$'. For example, value="${RTTP.RECORD_STATUS}" sets the value of the property to the value of the field 'RTTP.RECORD_STATUS'.

<control>

`<control>`

Identifies the display control that is rendered on the screen. Typical controls are text controls that display text, image controls that display images, and input controls that allow text to be entered. The controls provided with the reference implementation of Caplin Trader Client are described in "Display Control JavaScript classes".

Attributes:

Name	Type	Default	Req?	Description
`type`	string	(none)	Y	The fully qualified name of the JavaScript class that implements the control (for example, "caplin.control.basic.TextControl").

<downstream>

`<downstream>`

A list of the transforms that are applied to downstream data when the data is displayed in a control. Downstream data is data that is provided by a data provider, such as an 'rttpContainerGridDataProvider' or 'webServiceGridDataProvider' (see the document "Caplin Trader Client: Grid XML Configuration Reference"). Each transform in the list is defined by a <transform> tag.

Attributes:

Name	Type	Default	Req?	Description
`name`	string	(none)	N	A comma separated list of the data streams that are transformed by this set of downstream transforms. For example, name="bid,ask" would transform the 'bid' and 'ask' data streams. This optional attribute is only required if streams are defined using the 'streams' attribute of the <renderer> tag, and allows different transforms to be applied to each data stream.

<handler>

`<handler>`

Identifies an event handler that responds to events on the control, such as when the end user clicks the control or types text into the control. The event handlers provided with the reference implementation of Caplin Trader Client are described in "Event Handler JavaScript classes".

Attributes:

Name	Type	Default	Req?	Description
`name`	string	(none)	Y	The fully qualified name of the JavaScript class that implements the event handler (for example, "caplinx.control.handler. TradeOnClickHandler").

<renderer>

`<renderer>`

A renderer defines a display control and the optional upstream and downstream transforms that transform the data displayed in the control (see <control>, <upstream>, and <downstream>).

Attributes:

Name	Type	Default	Req?	Description
`streams`	string	(none)	N	A comma separated list of stream names (for example, streams="bid,ask"). Any name can be used for a stream, as long as it is unique to the renderer. This optional attribute is only required when a display control can display data from multiple data sources (such as the SpreadControl), and allows different transforms to be applied to different data streams (using the 'name' attribute of the <downstream> and <upstream> tags). The order and number of streams must match the order and number of data sources defined in the XML configuration of the display component. For example, if the renderer is used in a grid column, then the order and number of streams must match the order and number of fields in the XML definition of the column (see the document "Caplin Trader Client: Grid Configuration XML Reference").
`type`	string	(none)	Y	The type uniquely identifies the renderer from all other renderers that you define. The type can be referred to in the XML configuration of a display component, such as in the configuration of a column in a grid.

<rendererDefinitions>

`<rendererDefinitions>`

The outermost tag of the renderer definition XML, containing one or more renderer definitions (see <renderer>).

Attributes: This tag has no attributes.

<template>

`<template>`

A child of the <renderer> tag, the <template> tag can contain standard HTML (see <anyHTML>), as well as other <renderer> tags. In this way a 'composite renderer' can be constructed from HTML and references to other renderer definitions.

Attributes: This tag has no attributes.

<transform>

`<transform>`

Identifies a formatter, styler, or parser that transforms the <upstream> or <downstream> data in a display control. The transforms provided with the reference implementation of Caplin Trader Client are described in the subsections of "Element Renderer JavaScript classes".

Attributes:

Name	Type	Default	Req?	Description
`name`	string	(none)	N	The fully qualified name of the JavaScript class that implements the transform (for example, "caplin.element.styler. FlashStyler").

<upstream>

`<upstream>`

A list of the transforms that are applied to upstream data when the data is displayed in a control. Upstream data is data that is provided by the end user, such as when data is typed into a control in a column header to filter the instruments in a grid. Each transform in the list is defined by a <transform> tag.

Attributes:

Name	Type	Default	Req?	Description
`name`	string	(none)	N	A comma separated list of the data streams that are transformed by this set of upstream transforms. For example, name="bid,ask" would transform the 'bid' and 'ask' data streams. This optional attribute is only required if streams are defined using the 'streams' attribute of the <renderer> tag, and allows different transforms to be applied to each data stream.

5 Element Renderer JavaScript classes

An Element Renderer is a number of JavaScript classes that provide a display control, and the optional event handlers and data transforms (formatters, stylers, and parsers) that transform the data in the control (see the diagram in Schematic view of an element renderer 5).

When you define an Element Renderer in XML you must specify the JavaScript classes that you want to use. You can either specify one of the JavaScript classes provided with the reference implementation of Caplin Trader Client, or provide your own classes that implement the appropriate interfaces (see the **Caplin Trader Client: API Reference** documentation for further information).

This section describes the display control, event handler, and data transform Javascript classes that are provided with the reference implementation of Caplin Trader Client.

Tip: The **Caplin Trader Client: API Reference** documentation describes the interfaces, classes and methods that you can use when you write JavaScript code to extend Caplin Trader Client.

About formatters, stylers, and parsers

Formatters, stylers, and parsers are JavaScript classes that transform the data in a control. The function of each of these transforms is summarized below.

- A formatter converts data from a known input format to a required output format. If the input format is not recognized, then the input and output formats will be identical. A typical use of a formatter is to convert data from a machine friendly format (such as the number of seconds since the beginning of January 1970), to a user friendly format (such as 21-Jun-2009).
- A styler transforms the appearance of data in a control. A typical use is to set the foreground and background color of the displayed text.
- A parser analyses input data and attempts to convert it to a specified output format. A typical use of a parser is to convert a string entered by the end user (such as the date 21-Jun-2009), to a format suited to machine processing (such as the number of seconds since the beginning of January 1970).

Note: Formatters and stylers are applied in the same order in which they are defined in the XML configuration; the output of one transform being the input to the next. Parsers can be defined in any order..

5.1 Display Control JavaScript classes

When you define an Element Renderer in XML, you must specify the JavaScript class that implements the display control. You can write your own JavaScript class to implement the control or use one of the classes provided with the reference implementation of Caplin Trader Client.

This section describes the display controls that are provided with the reference implementation, and shows you how to specify a control in your Element Renderer XML configuration.

About display controls

A display control is a screen element that can display information (such as text or images), or allow the end user to interact with the application (such as by typing text into the control, or clicking on part of the control).

Specifying the JavaScript class that implements the display control

A display control is specified by setting the `type` attribute of the <control> 15 tag to the fully qualified name of the implementing JavaScript class. An example is shown below.

```
<control type="caplin.control.basic.ExampleDisplayControl" />
```

Available display controls

The display controls provided with the reference implementation of Caplin Trader Client are summarized in the following table, and are described in more detail in the sections that follow.

Control	Description
`caplin.control.basic.ImageControl` 21	A display control that can render one part of a multi part image on the screen when the parts of the multi part image are arranged vertically on top of each other. This type of control can display a different part of the image depending on the state of the mouse (mouse out, mouse over, and mouse down).
`caplin.control.basic.TextControl` 22	Simply displays text on the screen. This control can be used to display data in the cells of a grid column, such as indicative prices for an instrument or the name of an instrument.

caplin.control.basic.ImageControl

`<control type="caplin.control.basic.ImageControl">`

Description

An image control is a display control that can render one part of a multi part image on the screen when the parts of the multi part image are arranged vertically on top of each other. This type of control can display a different part of the image depending on the state of the mouse (mouse out, mouse over, and mouse down).

Event Handlers

Image controls can also have event handlers (see Event Handler JavaScript Classes 36) that respond to events on the control (such as when the end user clicks the image).

Properties that can be set using name/value pair attributes of the <attributes> tag:

None.

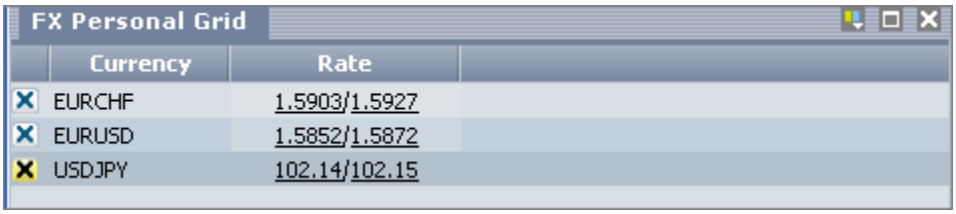
> **Tip:** If you need further information about when and how to use this control, please contact Caplin Support.

Example XML

In the reference implementation of Caplin Trader Client, the image control renders a "remove row" button at the left hand side of each row in a personal grid (one button in each row). The row is removed from the grid when the end user clicks this button (see caplinx.element.handler.RemoveGridRowOnClickHandler 37).

```
<control type="caplin.control.basic.ImageControl" />
```

The following pictures show the multi part image and how the part that is displayed changes when the mouse hovers over the image (in this case, when the mouse hovers over the button in the last row of the grid).



The second top part of the multiple image is displayed when the mouse hovers over a button.

caplin.control.basic.TextControl

```
<control type="caplin.control.basic.TextControl">
```

Description

A text control is a display control that simply displays text on the screen. This type of control can be used to display data in the cells of a grid column, such as the price of an instrument or the name of an instrument.

Event Handlers

Text controls can also have event handlers (see Event Handler JavaScript Classes 36) that respond to events on the control (such as when the end user clicks on a displayed price).

Properties that can be set using name/value pair attributes of the <attributes> tag

None.

Example XML

```
<control type="caplin.control.basic.TextControl" />
```

5.2 Formatter JavaScript classes

When you define an Element Renderer in XML you can specify optional JavaScript classes that format the data in the display control. You can write your own JavaScript class to implement a formatter or use one of the classes provided with the reference implementation of Caplin Trader Client.

This section describes the formatters that are provided with the reference implementation, and shows you how to specify a formatter in your Element Renderer XML configuration.

Specifying the JavaScript class that implements the formatter

A formatter is specified by setting the `name` attribute of the <transform> 17 tag to the fully qualified name of the implementing JavaScript class. An example is shown below.

```
<transform name="caplin.element.formatter.ExampleDataFormatter" />
```

Setting formatter properties

The properties of some formatters can be set using `name`/`value` pair attributes of the `<attribute>` 14 tag. In the following example the property `outputFormat` is set to "`d-M-Y`", which provides a format specification to the formatter.

```
<attribute name="outputFormat" value="d-M-Y"/>
```

You can also set the value of a property to the value of a field using the notation `value="${FIELD_NAME}"`. In the following example the property `outputFormat` is set to the value of the field "`PREFERENCE_DATE_FORMAT`".

```
<attribute name="outputFormat" value="${PREFERENCE_DATE_FORMAT}">
```

You can set the default value that is applied to a property if the value of the property is derived from a field, and the specified field has no value.

```
<attribute name="outputFormat" value="${PREFERENCE_DATE_FORMAT}" default="d-M-Y"/>
```

In this case the specified field is "`PREFERENCE_DATE_FORMAT`" and the default value is "`d-M-Y`". Setting the `default` value is optional.

Available formatters

The formatters provided with the reference implementation of Caplin Trader Client are summarized in the following table, and described in more detail in the sections that follow.

Formatter	Description
caplin.element.formatter.BondNotationFormatter 24	Formats a price to one of two standard bond price notations.
caplin.element.formatter.DateFormatter 25	Converts a date from a known format to another format.
caplin.element.formatter.DecimalFormatter 26	Formats a value to a specified number of decimal places.
caplin.element.formatter.InvalidPriceFormatter 27	Substitutes an invalid price with a specified string.
caplin.element.formatter.NullValueFormatter 27	Substitutes a null value with a specified string.
caplinx.element.formatter.SnapshotMaturityDateFormatter 28	Converts a fractional number of years into a formatted string.
caplin.element.formatter.TruncateDecimalFormatter 28	Truncates a value to a specified number of decimal places.

caplin.element.formatter.BondNotationFormatter

```
<transform name="caplin.element.formatter.BondNotationFormatter">
```

Description

The bond notation formatter converts a decimal number to one of two standard bond market notations (called fraction and decimal). Bond notations are sometimes used to represent US Treasury bond prices.

Each notation represents a bond price as a whole number followed by '-', and then two fractional parts. The most significant fractional part is the number of 32nds, and the least significant fractional part (the remainder) is the number of 8ths of a 32nd.

In fraction bond notation the remainder is expressed as a Unicode fraction (as in '102-05 7/8'), with 4/8ths (1/2) being represented by '+' (as in '102-05+').

In decimal bond notation the remainder is expressed as a whole number of 8ths (as in 102-05 07).

Properties that can be set using name/value pair attributes of the <attributes> tag:

name (property)	value (type)	Description
`fractions`	boolean	An optional attribute that converts the decimal number to fraction bond notation when 'true', and to decimal bond notation when 'false' (the default).
`rounding`	string	An optional attribute that rounds the remainder up to the nearest 8th when 'up', and down to the nearest 8th when 'down'. The default is to round fractions of an 8th less than 0.5 down, otherwise to round fractions of an 8th up.

Example XML

The following example converts a decimal number to fraction bond notation, rounding the remainder up to the nearest 8th.

```
<transform name="caplin.element.formatter.BondNotationFormatter">
  <attribute name="fractions" value="true"
             name="rounding" value="up" />
</transform>
```

In this case, the decimal number 102.183 would be converted to '102-05 7/8' (102 + 5/32+ 7/(8*32)).

The next example converts a decimal number to decimal bond notation, rounding the remainder up to the nearest 8th.

```
<transform name="caplin.element.formatter.BondNotationFormatter">
  <attribute name="fractions" value="false"
             name="rounding" value="up" />
</transform>
```

In this case, the decimal number 102.183 would be converted to '102-05 07' (102 + 5/32+ 7/(8*32)).

caplin.element.formatter.DateFormatter

```
<transform name="caplin.element.formatter.DateFormatter">
```

Description

The date formatter converts a date from an expected input format to a required output format. If the input date is not in the expected format then the input is returned unchanged. The input and output formats are specified using Ext date tokens (http://www.extjs.com/deploy/ext-2.2/docs/?class=Date).

Properties that can be set using name/value pair attributes of the <attributes> tag:

name (property)	value (type)	Description
`inputFormat`	Ext Date Tokens	The expected input format of the date (default is d-M-Y h:m:s).
`outputFormat`	Ext Date Tokens	The required output format of the date (default is d-M-Y h:m:s).

Example XML

The following example converts a date in the U format to the default output format. The U format is the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

```
<transform name="caplin.element.formatter.DateFormatter">
  <attribute name="inputFormat" value="U" />
</transform>
```

If the input date is 1e12 (the number of seconds in scientific notation), then the output date is "09-Sep-2001 01:46:40".

The next example converts a date in the U format to a specified output format (YMd).

```
<transform name="caplin.element.formatter.DateFormatter">
  <attribute name="inputFormat" value="U"
             name="outputFormat" value="YMd" />
</transform>
```

If the input date is 1e12 (the number of seconds in scientific notation), then the output date is "2001Sep09".

caplin.element.formatter.DecimalFormatter

`<transform name="caplin.element.formatter.DecimalFormatter">`

Description

The decimal formatter formats a value to a specified number of decimal places.

Properties that can be set using name/value pair attributes of the <attributes> tag:

name (property)	value (type)	Description
`dp`	nonNegativeInteger	The number of decimal places to which the value is formatted.

Example XML

The following example sets the number of decimal places to the value of the field "`DP`". If this field has no value, then the number of decimal places is "3" (`default="3"`).

```
<transform name="caplin.element.formatter.DecimalFormatter">
  <attribute name="dp" value="${DP}" default="3" />
</transform>
```

The next example sets the number of decimal places to the fixed value "3" (a default value is not required or defined).

```
<transform name="caplin.element.formatter.DecimalFormatter">
  <attribute name="dp" value="3" />
</transform>
```

In this case "1.26" would be formatted as "1.260", and "1.26666" would be formatted as "1.267".

caplin.element.formatter.InvalidPriceFormatter

```
<transform name="caplin.element.formatter.InvalidPriceFormatter">
```

Description

The invalid price formatter formats a price when the price is invalid, by substituting the price with replacement text. A price is invalid when it is not a positive number (or zero).

Properties that can be set using name/value pair attributes of the <attributes> tag:

name (property)	value (type)	Description
`invalidPrice`	string	The replacement text when the price is invalid.

Example XML

The following example formats an invalid price as "-".

```
<transform name="caplin.element.formatter.DateFormatter">
  <attribute name="invalidPrice" value="-" />
</transform>
```

caplin.element.formatter.NullValueFormatter

```
<transform name="caplin.element.formatter.NullValueFormatter">
```

Description

The null value formatter formats a string when the string is void, by substituting the string with replacement text. A string is void when it is null, undefined, or the empty string.

Properties that can be set using name/value pair attributes of the <attributes> tag:

name (property)	value (type)	Description
`nullValue`	string	The replacement text when the string is void.

Example XML

The following example formats a null, undefined, or empty string with the text "N/A".

```
<transform name="caplin.element.formatter.DateFormatter">
  <attribute name="nullValue" value="N/A" />
</transform>
```

caplinx.element.formatter.SnapshotMaturityDateFormatter

```
<transform name="caplinx.element.formatter.SnapshotMaturityDateFormatter">
```

Description

The snapshot maturity date formatter converts a fractional number of years into a formatted string. Values less than 1.0 are formatted as months, and values greater than or equal to 1.0 are formatted as years.

Properties that can be set using name/value pair attributes of the <attributes> tag:

None.

> **Tip:** If you need further information about when and how to use this control, please contact Caplin Support.

Example XML

The following example converts a fractional number of years into a formatted string.

```
<transform name="caplin.element.formatter.SnapshotMaturityDateFormatter" />
```

For example, 0.5 would be formatted as "6 m", and 1.5 would be formatted as "1.5 year".

caplin.element.formatter.TruncateDecimalFormatter

```
<transform name="caplin.element.formatter.TruncateDecimalFormatter">
```

Description

The truncate decimal formatter truncates a value to the specified number of decimal places. If the value already has fewer decimal places than that specified, then the value is not truncated.

Properties that can be set using name/value pair attributes of the <attributes> tag:

name (property)	value (type)	Description
dp	nonNegativeInteger	The number of decimal places to which the value is truncated.

Example XML

The following example sets the number of decimal places to "3", which would truncate "1.20000" to "1.200" and "1.26666" to "1.267", but would leave "1.20" unchanged.

```
<transform name="caplin.element.formatter.TruncateDecimalFormatter">
  <attribute name="dp" value="3" />
</transform>
```

5.3 Styler JavaScript classes

When you define an Element Renderer in XML you can specify optional JavaScript classes that style the data in the display control. You can write your own JavaScript class to implement a styler or use one of the classes provided with the reference implementation of Caplin Trader Client.

This section describes the stylers that are provided with the reference implementation, and shows you how to specify a styler in your Element Renderer XML configuration.

Specifying the JavaScript class that implements the styler

A styler is specified by setting the `name` attribute of the <transform> 17 tag to the fully qualified name of the implementing JavaScript class. An example is shown below.

```
<transform name="caplin.element.styler.ExampleDataStyler" />
```

Setting styler properties

The properties of some stylers can be set using `name`/`value` pair attributes of the <attribute> 14 tag. In the following example the property `color-up` is set to "`#286221`" (dark green).

```
<attribute name="color-up" value="#286221"/>
```

You can also set the value of a property to the value of a field using the notation `value="${FIELD_NAME}"`. In the following example the property `color-up` is set to the value of the field "`THIS_COLOR_UP`".

```
<attribute name="color-up" value="${THIS_COLOR_UP}"/>
```

Available stylers

The stylers provided with the reference implementation of Caplin Trader Client are summarized in the following table, and described in more detail in the sections that follow.

Styler	Description
caplin.element.styler.ClassStyler 30	Applies a CSS class to the HTML elements in a display control if a field has a particular value.
caplin.element.styler.EllipsisStyler 31	Applies an ellipsis (…) to the HTML elements in a control.
caplin.element.styler.FlashStyler 31	Changes the color of a displayed value for a specified period (giving the appearance of a flashing value) when the value increases or decreases.
caplinx.element.styler.PriceStyler 32	Applies a CSS class to a price to reflect the status of the price. The status of a price can be stale or not stale, and tradable or not tradable.
caplin.element.styler. RemoveButtonStyler 33	Disables the "remove row" button in a personal grid, and updates the tooltip for the button, when the record for the row is stale.
caplin.element.styler.TooltipStyler 34	Applies text for a tooltip to the HTML elements in a control.

caplin.element.styler.ClassStyler

```
<transform name="caplin.element.styler.ClassStyler">
```

Description

The class styler applies a CSS class to the HTML elements in a display control when a field has a particular value.

Properties that can be set using name/value pair attributes of the <attributes> tag:

name (property)	value (type)	Description
`state`	string	The field whose value determines whether or not the CSS class is applied. The class is applied if the value of this field is true, otherwise the class is not applied. The value that represents true is determined by the `true` property (see below).
`class`	string	The name of the CSS class that is applied.
`true`	string	The value that represents true.

Example XML

The following example identifies a field, a CSS class, and a value that determines whether or not the CSS class is applied.

```
<transform name="caplin.element.styler.ClassStyler">
  <attribute name="state" value="${RTTP.RECORD_STATUS}" />
  <attribute name="class" value="stale" />
  <attribute name="true" value="3" />
</transform>
```

In this case the CSS class `stale` is applied when the value of the field `${RTTP.RECORD_STATUS}`is `3`.

caplin.element.styler.EllipsisStyler

`<transform name="caplin.element.styler.EllipsisStyler">`

Description

The ellipsis styler displays an ellipsis (…) at the end of visible text when there is not enough room in the control to display all the text.

Note: The ellipses styler only works in Internet Explorer.

Properties that can be set using name/value pair attributes of the <attributes> tag:

None.

Example XML

The following example applies an ellipsis styler.

```
<transform name="caplin.element.styler.EllipsisStyler" />
```

caplin.element.styler.FlashStyler

`<transform name="caplin.element.styler.FlashStyler">`

Description

The flash styler changes the color of a displayed value for a specified period when the value increases or decreases. Changing the color in this way gives the appearance of a flashing value, and is typically used in the columns of a grid if the columns are displaying prices.

Properties that can be set using name/value pair attributes of the <attributes> tag:

name (property)	value (type)	Description
`duration`	positiveInteger	The number of milliseconds for which the color changes.
`color-up`	string	The foreground color to apply when the value increases (optional). The color can be any valid CSS color.
`color-down`	string	The foreground color to apply when the value decreases (optional). The color can be any valid CSS color.

name (property)	value (type)	Description
`backgroundColor-up`	string	The background color to apply when the value increases (optional). The color can be any valid CSS color.
`backgroundColor-down`	string	The background color to apply when the value decreases (optional). The color can be any valid CSS color.

Example XML

The following example sets the foreground and background colors that are applied when the data value increases or decreases.

```
<transform name="caplin.element.styler.FlashStyler">
  <attribute name="duration" value="500" />
  <attribute name="color-up" value="#286221" />
  <attribute name="color-down" value="#841819" />
  <attribute name="backgroundColor-up" value="#cdefbd" />
  <attribute name="backgroundColor-down" value="#feb3aa" />
</transform>
```

In this case a dark green foreground on a light green background is applied if the value increases, and a dark red foreground on a light red background if the value decreases. Each color is applied for 500 milliseconds, after which the colors return to the default colors.

caplinx.element.styler.PriceStyler

```
<transform name="caplinx.element.styler.PriceStyler">
```

Description

The price styler applies a CSS class to a price to reflect the status of the price. The status of a price can be stale or not stale, and tradable or not tradable.

Properties that can be set using name/value pair attributes of the <attributes> tag:

name (property)	value (type)	Description
`recordStatus`	string	The field that determines whether or not the price is stale. The name of the field is specified using the notation `"${FIELD_NAME}"`.
`tradableState`	string	The field that determines whether or not the price is tradable. The name of the field is specified using the notation `"${FIELD_NAME}"`.
`class-tradable`	string	The CSS class that is applied when the price is tradable but not stale.
`class-stale`	string	The CSS class that is applied when the price is stale but not tradable.
`class-tradablestale`	string	The CSS class that is applied when the price is tradable and stale.

Tip: If you need further information about when and how to use this styler, please contact Caplin Support.

Example XML

The following example sets the CSS classes that are applied when the price is tradable and not stale (`tradablePrices`), stale and not tradable (`stale`), and tradable and stale (`tradablestale`). No CSS class is applied when the price is not tradable and not stale.

```
<transform name="caplin.element.styler.PriceStyler">
  <attribute name="recordStatus" value="${RECORD_STATUS}" />
  <attribute name="tradableState" value="${TRADABLE}" />
  <attribute name="class-tradable" value="tradablePrices" />
  <attribute name="class-stale" value="stale" />
  <attribute name="class-tradablestale" value="tradablestale" />
</transform>
```

caplin.element.styler.RemoveButtonStyler

`<transform name="caplin.element.styler.RemoveButtonStyler">`

Description

The remove button styler disables the "remove row" button in a personal grid, and changes the tooltip for the button when the record for the row is stale.

Properties that can be set using name/value pair attributes of the <attributes> tag:

name (property)	value (type)	Description
`recordStatus`	string	The name of the field that determines whether or not the record is stale. The name is specified using the notation `"${FIELD_NAME}"`.
`stale-tooltip`	string	The tooltip that is displayed when the record is stale and the end user hovers over the "remove row" button.
`Non-stale-tooltip`	string	The tooltip that is displayed when the record is not stale and the end user hovers over the "remove row" button.

Example XML

The following example sets the tooltip for the "remove row" button when the record is not stale (`"Click to remove"`), and when the record is stale (`"disabled"`). The name of the field that determines whether or not the record is stale is set to `"RECORD_STATUS"`.

```
<transform name="caplin.element.styler.RemoveButtonStyler">
  <attribute name="recordStatus" value="${RECORD_STATUS}" />
  <attribute name="non-stale-tooltip" value="Click to remove" />
  <attribute name="stale-tooltip" value="disabled" />
</transform>
```

caplin.element.styler.TooltipStyler

```
<transform name="caplin.element.styler.TooltipStyler">
```

Description

The tooltip styler applies text for a tooltip to the HTML elements in a control. The tooltip text pops up when the mouse hovers over the control.

Properties that can be set using name/value pair attributes of the <attributes> tag:

name (property)	value (type)	Description
`tooltip`	string	The text for the tooltip. Field values can be included in tooltip text using the notation "`${FIELD_NAME}`".

Example XML

The following example sets the tooltip text for the HTML elements in a control.

```
<transform name="caplin.element.styler.TooltipStyler">
  <attribute name="tooltip" value="${ISSUER} ${COUPON} ${MATURITY}" />
</transform>
```

In this case the tooltip text is completely defined by field values, and would evaluate to something like "`JPM 5,00 16-Sep-2012`".

5.4 Parser JavaScript classes

When you define an Element Renderer in XML you can specify optional JavaScript classes that parse the data in the display control. Parsers are not currently provided with the reference implementation of Caplin Trader Client. You must write your own JavaScript class to implement a parser.

This section shows you how to specify a parser in your Element Renderer XML configuration.

Specifying the JavaScript class that implements the parser

A parser is specified by setting the `name` attribute of the <transform> 17 tag to the fully qualified name of the implementing JavaScript class. An example is shown below.

```
<transform name="mybank.element.parser.ExampleDataParser" />
```

5.5 Event Handler JavaScript classes

When you define an Element Renderer in XML you can specify optional JavaScript classes that handle mouse and keyboard events on the display control. You can write your own JavaScript class to implement an event handler or use one of the classes provided with the reference implementation of Caplin Trader Client.

This section describes the event handlers that are provided with the reference implementation, and shows you how to specify a handler in your Element Renderer XML configuration.

Specifying the JavaScript class that implements the event handler

An event handler is specified by setting the `name` attribute of the <handler> 16 tag to the fully qualified name of the implementing JavaScript class. An example is shown below.

```
<handler name="caplin.element.handler.ExampleEventHandler" />
```

Setting event handler properties

The properties of some event handlers can be set using `name`/`value` pair attributes of the <attribute> 14 tag. In the following example the property `action` is set to "`onAddLeg`".

```
<attribute name="action" value="onAddLeg"/>
```

You can also set the value of a property to the value of a field using the notation `value="${FIELD_NAME}"`. In the following example the property `action` is set to the value of the field "`LEG`".

```
<attribute name="action" value="${LEG}"/>
```

Available event handlers

The event handlers provided with the reference implementation of Caplin Trader Client are summarized in the following table, and described in more detail in the sections that follow.

Handler	Description
caplinx.element.handler.RemoveGridRowOnClickHandler 37	An example event handler that removes a row from a personal grid when the end user clicks a "remove row" button at the left-hand-side of the row.
caplinx.element.handler.TradeOnClickHandler 37	An example event handler that opens a trade ticket when the end user clicks a price in the display control.

caplinx.element.handler.RemoveGridRowOnClickHandler

```
<handler name="caplinx.element.handler.RemoveGridRowOnClickHandler">
```

Description

The remove-grid-row-on-click handler is an example of an event handler that removes a row from a personal grid when the end user clicks a "remove row" button at the left-hand-side of the row in a personal grid (see caplin.control.basic.ImageControl 21).

Properties that can be set using name/value pair attributes of the <attributes> tag

None.

> **Tip:** If you need further information about when and how to use this handler, please contact Caplin Support.

Example XML

```
<handler name="caplinx.element.handler.RemoveGridRowOnClickHandler" />
```

caplinx.element.handler.TradeOnClickHandler

```
<handler name="caplinx.element.handler.TradeOnClickHandler">
```

Description

The trade-on-click handler is an example of an event handler that opens a trade ticket when the end user clicks a price in the display control.

Properties that can be set using name/value pair attributes of the <attributes> tag

None.

Example XML

```
<handler name="caplinx.element.handler.TradeOnClickHandler" />
```

6 Glossary of terms and acronyms

This section contains a glossary of terms, acronyms, and abbreviations, used in this document.

Term	Definition
Caplin Liberator	A streaming (Comet) web server that streams market data over intranets, extranets and the Internet.
Caplin Trader	A framework for building multi-product financial trading portals.
Caplin Trader Client	The client (browser) side components of **Caplin Trader**.
Data provider	A data provider provides data to the **display components** of **Caplin Trader Client**. An example is the 'rttpContainerGridDataProvider' (see **Caplin Trader Client: Grid XML Configuration Reference**), which provides data from a web server for displaying in a **grid**.
Display component	One of the components of **Caplin Trader Client** that displays information on the screen. An example of a display component is a **grid**.
Display control	A screen element that is rendered by a JavaScript class. A display control can display information (such as text or images), or allow the end user to interact with the application (such as by typing text into the control, or clicking part of the control).
Downstream data	Data provided by a **data provider**, such as indicative prices from a web server.
Element Renderer	A **display control** and the optional transforms that **transform** the data displayed in the control. An Element Renderer can be identified in the XML configuration of a **display component** (such as to render data in the cells of a **grid** column).
Event handler	An event handler is a JavaScript class that handles mouse and keyboard events on a **display control**, such as when the end user clicks on a displayed price.
Field	A named identifier for a data item. An example of a field is a data item from **Caplin Liberator**, such as the price of a financial instrument. Fields supply data to the cells of a **grid**.
Formatter	A formatter converts data from a known input format to a required output format. If the input format is not recognized, then the input and output formats will be identical. A typical use of a formatter is to convert a value suited to machine processing (such as the number of seconds since the beginning of January 1970), to a string formatted for the benefit of the end user (such as 21-Jun-2009).
Grid	A **display component** of **Caplin Trader Client** that renders data in a tabular format.
Parser	A parser analyses input data and attempts to convert it to a specified output format. A typical use of a parser is to convert a string entered by the end user (such as the date 21-Jun-2009), to a format more suited to machine processing (such as the number of seconds since the beginning of January 1970).

Term	Definition
Stream	A stream is a named data source in an **Element Renderer**, in the same way that a **field** is a named data source in a **grid** column. In controls that support multiple data streams (such as the spread control), different transforms can be applied to each data stream.
Styler	A styler is a JavaScript class that changes the appearance of the data in a **display control** (for example, the color of the displayed text).
Transform	A data **styler**, **formatter**, or **parser** that transforms the data in a **display control**. A transform can change the appearance or value of the data (for example the color of the displayed text or the number of decimal places in a number).
Upstream data	Data provided by the end user, such as when data is typed into a control in a column header to filter the instruments in a grid.

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