CAPLIN

Caplin Trader 1.5

How To Create And Use Element Renderers

April 2010

Contents

1 Preface

1.1 What this document contains

This document describes how to create and use Element Renderers in a Caplin Trader application (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 software developers who want to create Element Renderers in Caplin Trader.

1.3 Related documents

- **Caplin Trader: Element Renderer Configuration XML Reference**

  Describes the XML-based configuration that defines Element Renderers in a Caplin Trader application.
- **Caplin Trader: Grid XML Configuration Reference**

  Describes the XML-based configuration that defines the layout and functionality of Grids in a Caplin Trader application.

- **Caplin Trader: API Reference**

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

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.

Information about the applicability of a section is enclosed in a box like this.
For example: “This section only applies to version 1.3 of the product.”

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.

Visit our feedback web page at https://support.caplin.com/documentfeedback/.

1.6 Acknowledgments

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

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

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

Java and *JavaScript* are trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

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

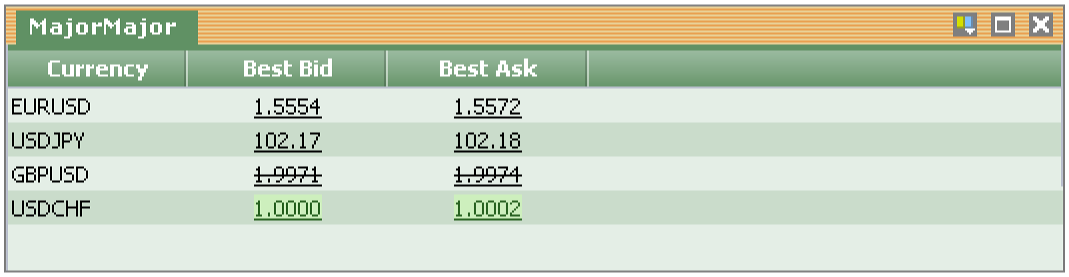
2 Introduction to the Element Renderer Framework

Caplin Trader 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.


MajorMajor

Currency	Best Bid	Best Ask
EURUSD	1.5554	1.5572
USDJPY	102.17	102.18
GBPUSD	~~1.9971~~	~~1.9974~~
USDCHF	1.0000	1.0002

Grid displaying indicative prices for FX currency pairs.

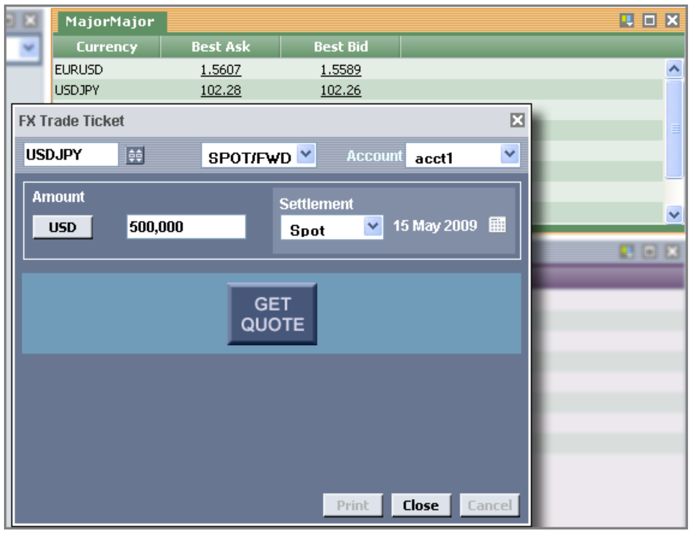
In the example above, three columns are displayed in the grid. The fields of the Currency column are text controls displaying currency pairs, while the fields of the Best Bid and Best Ask columns are text controls displaying indicative "Best Bid" and "Best Ask" prices for these currency pairs.

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.



Currency	Best Ask	Best Bid
EURUSD	1.5607	1.5589
USDJPY	102.28	102.26

USDJPY SPOT/FWD Account acct1

Amount USD 500,000

Settlement Spot 15 May 2009

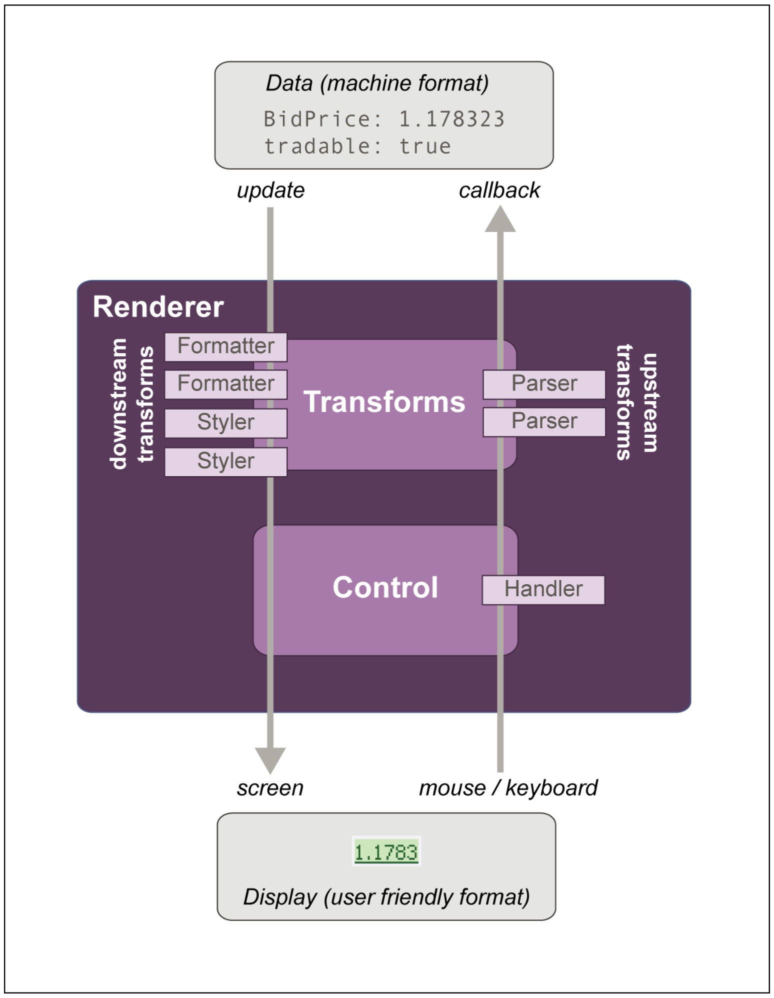
Trade ticket opens when the end user clicks an indicative price

An Element Renderer is always defined in XML, and the Caplin Trader framework 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: 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 24 for a description of these terms).

At present you can only easily apply Element Renderers in Grids, but in future releases of Caplin Trader 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 Defining an Element Renderer in XML

When you define an Element Renderer in XML, you specify the JavaScript classes that the Caplin Trader framework uses to construct each instance of the Element Renderer. You can either write your own custom Element Renderer JavaScript classes, or use one or more of the Element Renderer JavaScript classes provided with the Caplin Trader framework.

The example below shows a typical Element Renderer XML definition.

Tip: The document **Caplin Trader: Element Renderer Configuration XML Reference** fully describes the XML that you can use to define an Element Renderer, and lists the Element Renderer JavaScript classes that are provided with the Caplin Trader framework.

XML that defines an Element Renderer

```
<rendererDefinitions>
  ...
  <renderer type="fx-price">
     <control type="caplin.control.basic.TextControl">
       <handler name="mybank.element.handler.TradeOnClickHandler"/>
     </control>
     <downstream>
       <transform name="caplin.element.formatter.NullValueFormatter">
         <attribute name="nullValue" value=""/>
       </transform>
       <transform name="caplin.element.formatter.DecimalFormatter">
         <attribute name="DP" value="${DP} default="4" />
       </transform>
       <transform name="caplin.element.styler.FlashStyler"
         <attribute name="duration" value="500" />
         <attribute name="color-up" value="#286221" />
         <attribute name="color-down" value="#841819" />
         <attribute name="backgroundColor-up" value="#cdefbd" />
         <attribute name="backgroundColor-down" value="#feb3aa" />
       </transform>
       <transform name="mybank.element.styler.PriceStyler"
         <attribute name="recordStatus" value="${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 decimal 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.

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"`). The defined `type` can be used by your application to render data on the screen (see Using Element Renderers in your application 18). The defined `type` can also be referred to in the definition of a composite renderer (see Composite Element Renderer XML definition) 12.

- **`<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="mybank.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 a custom handler opens a trade ticket (`mybank.element.handler.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.formatter.DecimalFormatter">
        <attribute name="DP" value="${DP} default="4" />
      </transform>
      <transform name="caplin.element.styler.FlashStyler"
        <attribute name="duration" value="500" />
        <attribute name="color-up" value="#286221" />
        <attribute name="color-down" value="#841819" />
        <attribute name="backgroundColor-up" value="#cdefbd" />
        <attribute name="backgroundColor-down" value="#feb3aa" />
      </transform>
      <transform name="mybank.element.styler.PriceStyler"
        <attribute name="recordStatus" value="${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 four transforms are defined.

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

A framework JavaScript class formatter that defines how null values are displayed. The formatter is configured by `name`/`value` pairs 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.DecimalFormatter">`

A framework JavaScript class formatter that formats a value to a specified number of decimal places. The formatter is configured by `name`/`value` pairs as shown in the table below.

Formatter configuration (DecimalFormatter)	**Description**
`<attribute name="DP" value="${DP}" default="4" />`	In this case values are rendered to the number of decimal places specified by the `DP` field. If this field has no value, then the number of decimal places is "4" (`default="4"`). Note that a value is obtained from a field if `value` is defined using the notation `value="${FIELD_NAME}"`.

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

A framework JavaScript class styler that gives the appearance of a flashing value when the price 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.

4. `<transform name="mybank.element.styler.PriceStyler">`

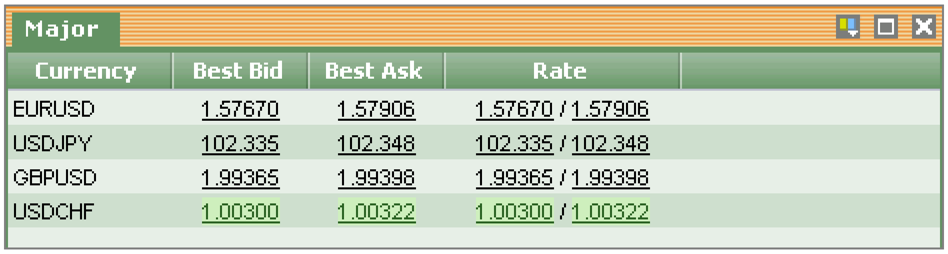
A custom JavaScript class styler that changes the appearance of the displayed value (using CSS classes), depending on the tradable state of the data. The styler is configured by `name`/`value` pairs as shown in the table below.

Styler configuration (PriceStyler)	**Description**
`<attribute name="recordStatus" value="${RTTP.RECORD_STATUS}" />`	The status of the data (stale or not stale). In this case the field "`RTTP.RECORD_STATUS`" holds the status of the data.
`<attribute name="tradableState" value="${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, 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*.

3.1 Composite Element Renderer XML definition

A composite Element Renderer is constructed from HTML and references to other renderer definitions, and is used to render data from multiple fields in the column of a Grid.



Currency	Best Bid	Best Ask	Rate
EURUSD	1.57670	1.57906	1.57670 / 1.57906
USDJPY	102.335	102.348	102.335 / 102.348
GBPUSD	1.99365	1.99398	1.99365 / 1.99398
USDCHF	1.00300	1.00322	1.00300 / 1.00322

Data from two fields being rendered in the 'Rate' column of a Grid

In this case two prices ('Best bid' and 'Best ask') are separated by the '/' character and rendered in the Rate column of a Grid.

The example below shows a typical XML definition for a composite renderer.

Tip: The document **Caplin Trader: Element Renderer Configuration XML Reference** fully describes the XML that you can use to define an Element Renderer, and lists the Element Renderer JavaScript classes that are provided with the Caplin Trader framework.

XML that defines a composite Element Renderer

```
<rendererDefinitions>
  ...
  <renderer type="fx-spread" streams="bid,ask">
    <template>
      <var>
        <renderer type="fx-price" stream="bid"/>
        /
        <renderer type="fx-price" stream="ask"/>
      </var>
    </template>
  </renderer>
  ...
  <renderer type="fx-price">
     <control type="caplin.control.basic.TextControl">
       ...
     </control>
     <downstream>
       ...
     </downstream>
  </renderer>

</rendererDefinitions>
```

In this configuration the composite renderer `type="fx-spread"` renders data from two data streams (`streams="bid,ask"`).

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-spread" streams="bid,ask">
    <template>
    ...
  </renderer>

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

In this case two renderers are defined (`type="fx-spread"` and `type="fx-price"`).

The renderer `type="fx-spread"` is a composite renderer, with components defined by the `<template>` tag. The defined `type` of this composite renderer (`type="fx-spread"`) can be used in your application to render data on the screen (see Using Element Renderers in your application 18).

If the composite renderer is used to render data in the column of a Grid, then the order and number of streams (`streams="bid,ask"`) must match the order and number of fields in the XML definition of the column (see the document "Caplin Trader: Grid Configuration XML Reference").

The renderer `type="fx-price"` consists of a `<control>` and the `<downstream>` transforms that transform the data rendered in the control. This renderer is used by the composite renderer to render the data streams (see the `<template>` tag), and is defined by the XML described in Defining an Element Renderer in XML 7.

- **`<template>`** contains the definition of the composite renderer.

```
    <template>
      <var>
        <renderer type="fx-price" stream="bid"/>
        /
        <renderer type="fx-price" stream="ask"/>
      </var>
    </template>
```

The `<template>` tag contains standard HTML as well as other <renderer> tags. In this way a composite renderer is constructed from HTML and references to other renderer definitions.

The HTML `<var>` tag in this example is used to enclose the components of the composite renderer, and can also be referred to in CSS to style the composite renderer.

Because the `<renderer>` tag is a child of the `<template>` tag, the `type` attribute of the `<renderer>` tag identifies the renderer of a particular data stream. In this case the renderer `type="fx-price"` renders both the 'bid' and 'ask' streams (`stream="bid"` and `stream="ask"`).

In this example the '`/`' character separates the data streams when they are rendered on the screen.

4 Customizing Element Renderers

You can customize an Element Renderer by:

- modifying the XML definition of the Element Renderer,
- writing your own custom JavaScript classes and employing these in your XML definition,
- using a combination of these two customization techniques.

The Reference Implementation of Caplin Trader has several pre-defined Element Renderers that you can copy and use in your own Caplin Trader application. You will find the XML definitions for these Element Renderers in the file *apps/webapps/caplintrader/applications/CaplinTrader/conf/rendererDefinitions.jsp*.

4.1 Modifying the XML definition

The following examples show you how to modify the XML definition of Element Renderers that have been created to render data in the cells of a Grid. In each case a downstream transform is inserted into the XML definition.

Note: Formatter and styler transforms 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.

The document **Caplin Trader: Element Renderer Configuration XML Reference** lists the transforms that are supplied with the Caplin Trader framework, and describes how to set the properties of these transforms.

Adding a Decimal Formatter

The Caplin Trader framework includes a decimal formatter that can be inserted into the XML definition of an Element Renderer, as shown in the following example.

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

If this renderer is applied to the cells of a Grid, then the data in each cell will be formatted to three decimal places. The formatter is applied in addition to any other formatters in the XML definition of this Element Renderer.

Adding a Flash Styler

The Caplin Trader framework includes a flash styler that can be inserted into the XML definition of an Element Renderer, as shown in the following example.

```
<rendererDefinitions>
  ...
  <renderer type= ...
     <downstream>
       <transform type="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>
  ...
</rendererDefinitions>
```

If this renderer is applied to the cells of a Grid, then the price flashes when the displayed value increases or decreases.

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 is applied if the value decreases. Each color is applied for 500 milliseconds, after which the colors return to the default colors.

The formatter is applied in addition to any other formatters in the XML definition of this Element Renderer.

4.2 Writing your own JavaScript Classes

The Caplin Trader framework includes a number of JavaScript classes that you can use in your Element Renderer XML definitions. You may however want to write your own JavaScript classes that transform data or that respond to events in a customized way, and include these in your Element Renderer XML definitions.

The following examples show you how to write customized data transforms and event handlers.

Tip: The **Caplin Trader API Reference** document (`caplin.element` package) defines the interfaces that your code must implement when you write a new transform or event handler JavaScript class.

The document also has a section called "Writing your own classes", linked from the "Overview" section, that provides guidance on writing JavaScript classes using the Caplin Trader framework. The section includes guidance on how to implement an interface.

Creating a new Formatter

When you write a new formatter class, your JavaScript code must implement the `format` method of the `caplin.element.Formatter` interface.

The following is an example of a custom formatter that transforms a value to upper case when the value is a string, and returns the unformatted value otherwise.

```
mybank.element.formatter.UpperCaseFormatter.prototype.format = function(sValue,
                                                                mAttributes) {
    return typeof sValue === "string" ? sValue.toUpperCase() : sValue;
}
```

You insert a custom formatter into the XML definition of an Element Renderer as you would for any formatter, by setting the `type` attribute of a `<transform>` tag to the fully qualified name of the implementing JavaScript class.

```
<rendererDefinitions>
  ...
  <renderer type= ...
     <downstream>
       <transform type="mybank.element.formatter.UpperCaseFormatter" />
  ...
</rendererDefinitions>
```

Because this custom formatter does not have any properties that need to be set, the `<transform>` tag does not have any child `<attribute>` tags.

Creating a new Styler

When you write a new styler class, your JavaScript code must implement the `style` method of the `caplin.element.Styler` interface.

The following is an example of a custom styler that adds a CSS class to a control when the trading state is "READY", and removes the CSS class otherwise.

```
mybank.element.styler.PriceStyler.prototype.style = function(sValue,
                                                             mAttributes,
                                                             oControl) {
  if (mAttributes["tradingState"] === "READY") {
    oControl.addClass("tradable");  // adds a CSS class
  } else {
    oControl.removeClass("tradable");  // removes a CSS class
  }
}
```

In this case the trading state is determined by the value of "`tradingState`", which is set in the XML definition of the Element Renderer.

You insert a custom styler into the XML definition of an Element Renderer as you would for any styler, by setting the `type` attribute of a `<transform>` tag to the fully qualified name of the implementing JavaScript class.

```
<rendererDefinitions>
  ...
  <renderer type= ...
     <downstream>
       <transform type="mybank.element.styler.PriceStyler">
         <attribute name="tradingState" value="${TRADING_STATE}"/>
       </transform>
  ...
</rendererDefinitions>
```

Because the `tradingState` property of the custom styler needs to be set, the `<transform>` tag has a child `<attribute>` tag that sets the value of "`tradingState`" to the value of the `TRADING_STATE` field. The `${}` notation indicates that the value of the property is derived from a field.

Creating a new Handler

When you write a new handler class, your JavaScript code must implement the `onclick` method of the `caplin.element.Handler` interface.

The following is an example of a custom handler that opens a trade ticket if the trading state is "READY" when the end user clicks the value displayed by the Element Renderer.

```
mybank.element.handler.TradeOnClickHandler.prototype.onclick = function(oDomEvent,
                                                                         oRenderer,
                                                                         mAttributes) {
  if (mAttibutes["tradingState"] === "READY") {
      // add code here that opens the trade ticket
  }
}
```

In this case the trading state is determined by the value of "`tradingState`", which is set in the XML definition of the Element Renderer.

You insert a custom handler into the XML definition of an Element Renderer as you would for any handler, by setting the `type` attribute of a `<handler>` tag to the fully qualified name of the implementing JavaScript class.

```
<rendererDefinitions>
  ...
  <renderer type= ...
     <control type="caplin.control.basic.TextControl">
       <handler type="mybank.element.handler.TradeOnClickHandler">
         <attribute name="tradingState" value="${TRADING_STATE}"/>
       </handler>
     </control>
  ...
</rendererDefinitions>
```

Because the `tradingState` property of the custom handler needs to be set, the `<handler>` tag has a child `<attribute>` tag that sets the value of "`tradingState`" to the value of the `TRADING_STATE` field. The `${}` notation indicates that the value of the property is derived from a field.

5 Using Element Renderers in your application

Element Renderers that have been defined in XML can be used in your application in two ways:

- In a Grid, by configuring the XML definition of the Grid.
- Elsewhere in the application, by instantiating the Element Renderer in JavaScript.

5.1 Using an Element Renderer in a Grid

When an Element Renderer has been defined in XML, you can use it to render data in the cells of a Grid. To define the Element Renderer that you want to use, set the `cellRenderer` attribute of the Grid column to the name of the Element Renderer in the XML configuration of the Grid.

In the example below, all Grids that inherit from the FX grid template (`gridTemplate id="FX"`) use the `"fx-price"` Element Renderer to render data in the `"Best Bid"` and `"Best Ask"` columns of the Grid.

```
<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>
```

The Grid framework manages the life cycle of the Grid and the Element Renderers that are used in the Grid, so there is no need to instantiate the Grid or Element Renderers in JavaScript. When a new Grid is created, the Grid framework creates an Element Renderer instance for each cell in the Grid; when the Grid is deleted, the Grid framework destroys these Element Renderer instances.

In the Reference Implementation of Caplin Trader, Grids are defined in the file *apps/webapps/caplintrader/applications/CaplinTrader/conf/gridDefinitions.xml*.

For further information about how to configure Grids in a Caplin Trader application, see the document **Caplin Trader: Grid XML Configuration Reference**. In particular, the description of the `<gridTemplate>` tag describes Grid inheritance, and the description of the `cellRenderer` attribute of the `<column>` tag discusses the Element Renderer.

5.2 Using an Element Renderer elswhere in the application

If you want to use an Element Renderer to render data outside of a Grid, then the life cycle of each Element Renderer instance must be managed in JavaScript by your Caplin Trader application.

This is in contrast to an Element Renderer that is used to render data in the cells of a Grid, where the life cycle of each instance of the Element Renderer is managed by the Grid framework.

To use an Element Renderer to render data outside of a Grid, you must:

1. Create an instance of the Element Renderer 19 in JavaScript.
2. Bind the instance of the Element Renderer 20 to an HTML DOM element.
3. Display values 21 and capture events on the DOM element.

When the Element Renderer instance is no longer needed, your application code must unbind the instance from the HTML DOM element, which frees all resources used by that instance.

The **Caplin Trader API Reference** document (`caplin.element` package) describes the JavaScript classes that you can use to manage an instance of an Element Renderer in your Caplin Trader application.

Creating an instance of the Element Renderer

To create an instance of the Element Renderer you must:

1. Identify the type of Element Renderer that you want to instantiate.
2. Get a reference to the `RendererFactory` JavaScript object.
3. Create an instance of the Element Renderer by invoking `createRenderer()` on the `RendererFactory` object.

The Element Renderer type is defined by the `type` attribute of the `<renderer>` tag in the XML definition of the Element Renderer. In the Reference Implementation of Caplin Trader, Element Renderers are defined in the file *apps/webapps/caplintrader/applications/CaplinTrader/conf/rendererDefinitions.jsp*.

The example code below shows you how to instantiate an instance of the "`fx-price`" Element Renderer.

```
var sRendererType = "fx-price";

// Get a reference to a RendererFactory object
var oRendererFactory = caplin.element.ElementFactory.getRendererFactory();

// Create the Element Renderer instance
var oRenderer = oRendererFactory.createRenderer(sRendererType);
```

When you have completed these steps, the Element Renderer instance is ready to be bound to a DOM element.

Binding an instance of the Element Renderer

Element Renderers created by the Renderer Factory implement the `caplin.element.RendererFramework` interface. This interface has methods that bind and unbind Element Renderer instances to DOM elements (see the **Caplin Trader API Reference** for a description of this interface).

To bind an instance of an Element Renderer you must:

1. Get the identity of the DOM element where you want to render data (the element must already exist).
2. Create the HTML for the Element Renderer instance inside the DOM element.
3. Bind (attach) the Element Renderer instance to the DOM element.
4. Set the name and namespace of the Element Renderer instance.

The example code below shows you how to bind an instance of an Element Renderer to the DOM element `BID_PRICE_HOLDER`.

```
// Get the identity of the DOM element
var ePriceHolder = document.getElementById("BID_PRICE_HOLDER");

// Create the HTML for the renderer instance inside the DOM elemnt
ePriceHolder.innerHTML = oRenderer.createHtml();

// Bind the renderer instance to the DOM element
oRenderer.bind(ePriceHolder);

// Set the name and namespace of the renderer instance
oRenderer.setName("BID_PRICE");
oRenderer.setNamespace("/FX/GBPUSD");
```

You must set the name and namespace of the Element Renderer instance to the name of the field (`BID_PRICE`) and instrument (`/FX/GBPUSD`) that is rendered, so that event handlers can access this information. For example, if you have an event handler that handles `onclick` events, then the event handler can invoke `getname()` and `getnamespace()` on the Element Renderer instance to determine the name of the field and instrument that the event relates to.

When you have completed these steps, the Element Renderer instance is ready to display data and capture events.

Displaying values

Element Renderers created by the Renderer Factory implement the `caplin.element.RendererFramework` interface. This interface has methods that determine the values that are displayed by an Element Renderer instance (see the **Caplin Trader API Reference** for a description of this interface).

Below we look at two different ways in which your code can use the same Element Renderer to display a price in a custom dialog. In each case the Element Renderer will:

- Display a price to a specified number of decimal places.
- Flash according to price movement.
- Apply CSS styling according to the tradable state of the price.

The example XML that defines the Element Renderer is reproduced below, and described in detail in Defining an Element Renderer in XML 7.

XML that defines the example Element Renderer

```
<rendererDefinitions>
  ...
  <renderer type="fx-price">
     <control type="caplin.control.basic.TextControl">
       <handler name="mybank.element.handler.TradeOnClickHandler"/>
     </control>
     <downstream>
       <transform name="caplin.element.formatter.NullValueFormatter">
         <attribute name="nullValue" value=""/>
       </transform>
       <transform name="caplin.element.formatter.DecimalFormatter">
         <attribute name="DP" value="${DP} default="4" />
       </transform>
       <transform name="caplin.element.styler.FlashStyler"
         <attribute name="duration" value="500"/>
         <attribute name="color-up" value="#286221"/>
         <attribute name="color-down" value="#841819"/>
         <attribute name="backgroundColor-up" value="#cdefbd"/>
         <attribute name="backgroundColor-down" value="#feb3aa"/>
       </transform>
       <transform name="mybank.element.styler.PriceStyler"
         <attribute name="recordStatus" value="${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>
```

Option 1: invoke setValue() – set the displayed value

One way to render a value is to invoke `setValue()` on the Element Renderer instance.

First create an instance 19 of the Element Renderer, and then bind that instance 20 to the DOM element of the custom dialog.

```
// create instance of the fx-price renderer
var oRendererFactory = caplin.element.ElementFactory.getRendererFactory();
var oRenderer = oRendererFactory.createRenderer("fx-price");

// bind renderer to custom dialog
var ePriceHolder = document.getElementById("BID_PRICE_HOLDER");
ePriceHolder.innerHTML = oRenderer.createHtml();
oRenderer.bind(ePriceHolder);
oRenderer.setName("BID_PRICE");
oRenderer.setNamespace("/FX/GBPUSD");
```

In this case an instance of the "`fx-price`" Element Renderer is created and bound to the DOM element "`BID_PRICE_HOLDER`".

Now invoke `setValue()` on the Element Renderer instance, passing in the value that you want to render.

```
// set renderer value directly
var sBidPrice = "1.4932";
oRenderer.setValue(sBidPrice);
```

In this case `1.4932` is rendered in the custom dialog. The value is rendered to four decimal places because 4 is the default value of the decimal formatter in the Element Renderer XML definition.

```
<transform name="caplin.element.formatter.DecimalFormatter">
  <attribute name="DP" value="${DP}" default="4" />
```

Option 2: invoke updateFields() – update multiple fields at the same time

Another way to render a value is to invoke `updateFields()` on the Element Renderer instance, passing in name/value pairs for the fields that you want to update.

First create an instance 19 of the Element Renderer, and then bind that instance 20 to the DOM element of the custom dialog.

```
// create instance of the fx-price renderer
var oRendererFactory = caplin.element.ElementFactory.getRendererFactory();
var oRenderer = oRendererFactory.createRenderer("fx-price", ["BidPrice"]);

// bind renderer to custom dialog
var sHtml = oRenderer.createHtml();
var ePriceHolder = document.getElementById("BID_PRICE_HOLDER");
ePriceHolder.innerHTML = oRenderer.createHtml();
oRenderer.bind(ePriceHolder);
oRenderer.setName("BID_PRICE");
oRenderer.setNamespace("/FX/GBPUSD");
```

In this case an instance of the "`fx-price`" Element Renderer is created that renders values from the "`BidPrice`" field. The instance is bound to the DOM element "`BID_PRICE_HOLDER`".

Now invoke `updateFields()` on the Element Renderer instance, passing in name/value pairs for the fields that you want to update. Field names for each transform are specified in the Element Renderer XML definition, while the field that is rendered is specified in the call to `createRenderer()`, as shown in the example code above.

```
// set renderer value indirectly, through field names
var mFields = { "BidPrice": "1.4932",
                "DP": 3,
                "TRADABLE_STATE": true,
                "RECORD_STATUS": 3 };
oRenderer.updateFields(mFields);
```

In this case `1.493` is rendered in the custom dialog. The value is rendered to three decimal places because the `"DP"` field is set to 3 when `updateFields()` is invoked. In the XML definition of the Element Renderer, the `"DP"` field (`value="${DP}"`) is specified as holding the value that sets the number of decimal places property (`name="DP"`) of the decimal formatter.

```
<transform name="caplin.element.formatter.DecimalFormatter">
  <attribute name="DP" value="${DP}" default="4" />
```

6 Glossary of terms and acronyms

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

Term	Definition
Blotter	A **display component** of **Caplin Trader** that displays information about each trade.
Caplin Liberator	Caplin Liberator is a real-time financial internet hub (server) that delivers trade messages and market data to and from subscribers over any network.
Caplin Trader	A **Rich Internet Application** framework for constructing **Caplin Xaqua client** applications for browser-based trading. It includes a comprehensive set of trading **GUI** components. Caplin Trader was formerly called "Caplin Trader Client".
Caplin Xaqua	A framework for building single-dealer platforms that enables banks to deliver multi-product trading direct to client desktops.
Caplin Xaqua client	A client desktop application that interfaces with **Caplin Xaqua** to deliver multi-product trading to end users. The application can be implemented in any technology that is supported by Caplin Xaqua; for example Ajax, Microsoft .NET, Microsoft Silverlight™, Adobe Flex™, and Java™. **Caplin Trader** is a framework for constructing browser-based Caplin Xaqua client applications.
Data provider	A data provider provides data to the **display components** of **Caplin Trader**. An example is the 'rttpContainerGridDataProvider' (see **Caplin Trader: Grid XML Configuration Reference**), which provides data from a web server for displaying in a **grid**.
Display component	A **GUI** component of **Caplin Trader** that can be rendered in a page on the screen. The term also refers to the JavaScript code that generates the component and handles its user interaction. Caplin Trader has a number of pre-defined, customizable display components, such as **Grids**, **Trade Tiles**, and the **Blotter**.
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**.

Term	Definition
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** that renders data in a tabular format.
GUI	Graphical User Interface
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).
Renderer	Another name for an **Element Renderer**.
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).
Trade Tile	A **display component** that allows the user to trade on a product with a single mouse click.
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.

Single-dealer platforms for the capital markets

CAPLIN

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