

CAPLIN

# Caplin Trader 2.0

---

**Overview**

August 2010

**CONFIDENTIAL**

# Contents

<b>1</b>	<b>Preface.....</b>	<b>1</b>
1.1	What this document contains.....	1
	About Caplin document formats .....	1
1.2	Who should read this document.....	1
1.3	Related documents.....	2
1.4	Typographical conventions.....	2
1.5	Feedback.....	2
1.6	Acknowledgments.....	2
1.7	Open Source Software .....	3
<b>2</b>	<b>What is Caplin Trader?.....</b>	<b>4</b>
2.1	What are the benefits of using Caplin Trader?.....	7
2.2	What does Caplin Trader consist of?.....	8
<b>3</b>	<b>The User Experience.....</b>	<b>9</b>
<b>4</b>	<b>Architecture.....</b>	<b>10</b>
4.1	Caplin Trader and Caplin Xaqua.....	11
4.2	Caplin Trader application structure.....	12
	Application Code .....	13
	Caplin Xaqua Client APIs and Liberator .....	13
	Application server .....	14
	Persistence database .....	14
4.3	Caplin Trader's internal architecture.....	15
	Caplin Trader API .....	16
	Display components .....	17
	Internal modules .....	22
	JavaScript Preprocessor .....	23
	Third-party modules .....	23
<b>5</b>	<b>Building a Caplin Trader application.....</b>	<b>24</b>
5.1	Getting started.....	24
5.2	The Caplin Trader development framework.....	26
5.3	Defining the layout and overall appearance.....	27
	Layout concepts .....	27
	Changing the appearance of the application .....	30
	Layout Manager features .....	30

---

	Customizing layouts .....	31
5.4	Customizing display components.....	34
	Customizing display components with XML (Grids) .....	35
	Element renderers .....	38
	Decorators .....	40
5.5	Adding your own component types.....	42
5.6	Using the Form Builder.....	44
5.7	Defining a Trade Model.....	47
<b>6</b>	<b>Appendix A: Recommended browsers.....</b>	<b>49</b>
<b>7</b>	<b>Appendix B: Customization status.....</b>	<b>50</b>
<b>8</b>	<b>Glossary of terms and acronyms.....</b>	<b>52</b>

# 1 Preface

## 1.1 What this document contains

This document contains a business and technical overview of Caplin Trader version 2.0

### 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 anyone who requires an overview of what Caplin Trader is and how it can be used to build financial trading client applications.

Its readership includes:

- ◆ Business Managers
- ◆ Technical Managers
- ◆ System Architects
- ◆ System Administrators
- ◆ Software Developers

## 1.3 Related documents

### ◆ Caplin Xaqua Overview

A business and technical overview of Caplin Xaqua.

## 1.4 Typographical conventions

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

<i>Type</i>	<i>Uses</i>
<b>aMethod</b>	Function or method name
<i>aParameter</i>	Parameter or variable name
<i>/AFolder/Afile.txt</i>	File names, folders and directories
<code>Some code;</code>	Program output and code examples
The <code>value=10</code> 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
<b>XYZ Product Overview</b>	Document name

## 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*®, *Adobe Reader*, *Adobe Flash*, and *Adobe Flex* are either registered trademarks or trademarks 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.

*Emprise JavaScript Charts* is a product of Emprise Corporation.

*ExtJS* is a JavaScript library produced by Sencha Inc.

*JBoss* and *Hibernate* are registered trademarks of Red Hat, Inc. in the United States and other countries.

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

*MySQL* is a registered trademark of MySQL AB in the United States, the European Union and other countries.

*Oracle* is a registered trademark of Oracle and/or its affiliates.

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

*Sun, Solaris, Java, and JSP* are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. or other countries.

*Sybase and Sybase SQL Server* are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

## 1.7 Open Source Software

This Caplin product incorporates the following Open Source software:

Open Source item	Use in Caplin Trader	Further information
Apache ANT	Build and dependency management framework.	<a href="http://ant.apache.org/">http://ant.apache.org/</a>
Apache Maven	Build and dependency management framework.	<a href="http://maven.apache.org/">http://maven.apache.org/</a>
gzip	The gzip compression algorithm is used (via standard Java utility classes) to compress Caplin Trader application resource files on the application server before they are downloaded to client browsers.	<a href="http://www.gzip.org">http://www.gzip.org</a> <a href="http://java.sun.com/javase/6/docs/api/java/util/zip/package-summary.html">http://java.sun.com/javase/6/docs/api/java/util/zip/package-summary.html</a>
JBoss Hibernate	Communication between the application server and database.	<a href="http://www.hibernate.org">http://www.hibernate.org</a>
JUnit	Unit testing framework.	<a href="http://www.jsunit.net/">http://www.jsunit.net/</a>
Mock4JS	Unit testing framework.	<a href="http://mock4js.sourceforge.net/">http://mock4js.sourceforge.net/</a>
OpenAjax 1.0	Caplin Trader's Event Manager uses the publish and subscribe event management features of the OpenAjax Hub.	OpenAjax Alliance <a href="http://www.openajax.org">http://www.openajax.org</a>
Selenium	Acceptance testing framework	SeleniumHQ <a href="http://seleniumhq.org">http://seleniumhq.org</a>
WebDriver	Performance and stress testing framework.	SeleniumHQ <a href="http://seleniumhq.org">http://seleniumhq.org</a>

## 2 What is Caplin Trader?

Caplin Trader is a web application framework for financial trading. It enables you to build advanced trading applications with unlimited functionality.

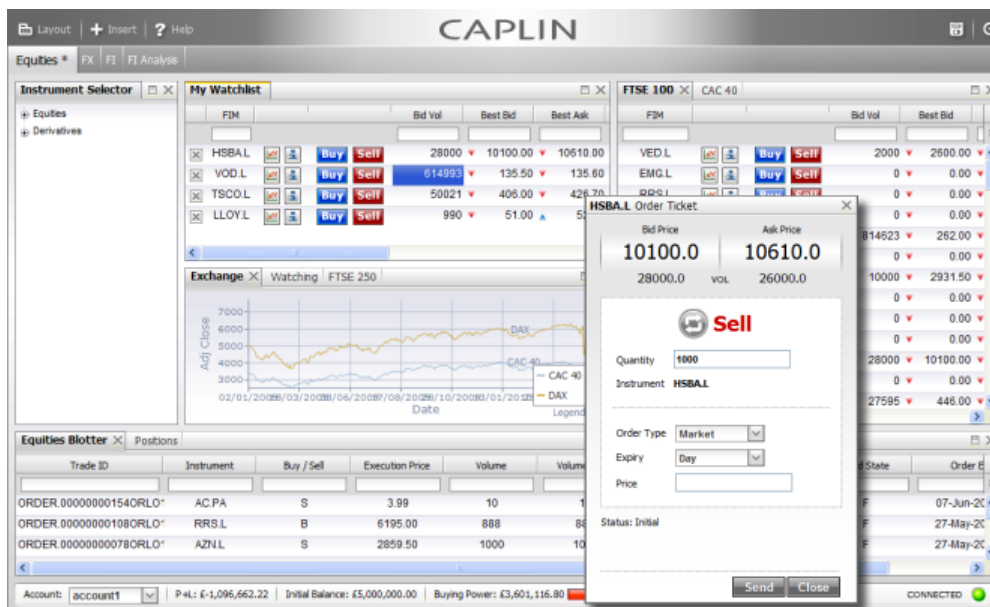
Here are some examples of typical trading screens that you can implement with Caplin Trader:



Example Caplin Trader application – Foreign Exchange (FX) trading screen

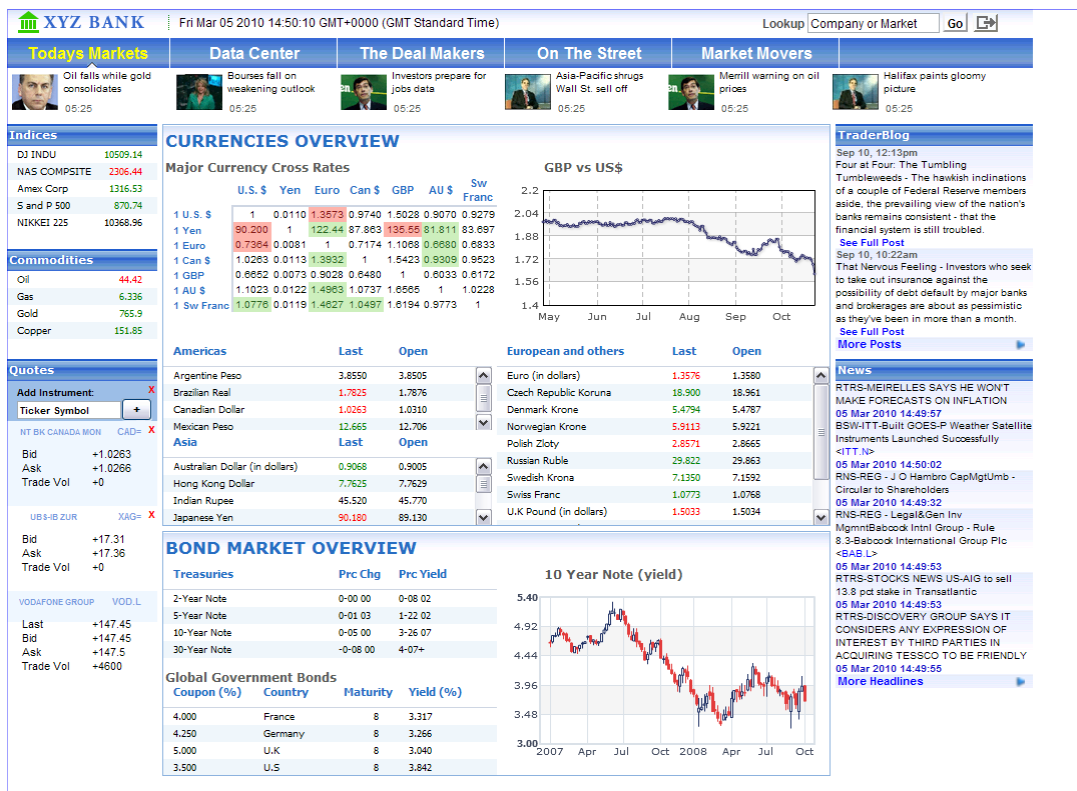


Example Caplin Trader application – Fixed Income (FI) trading screen



Example Caplin Trader application – Equities trading screen





Example Caplin Trader application – Retail screen

## 2.1 What are the benefits of using Caplin Trader?

### **Caplin Trader is built on web standards, which means:**

- ◆ Universal deployment
  - Applications built with Caplin Trader work with all common browsers
  - They automatically handle the non-standard features of some older browser versions
  - There is no need to change firewalls or security settings
- ◆ You can on-board users simply and painlessly
  - End-users do not have to locally install applications or browser plug-ins; access is as easy as viewing a web page
- ◆ It is easy to integrate a Caplin Trader application with your existing web assets and content, such as research, historic data, news, and video
- ◆ Applications built with Caplin Trader are easy to maintain
- ◆ There is a huge pool of programmers with relevant skills

### **Caplin Trader is specifically designed for financial trading, which means:**

- ◆ You can concentrate on the added-value aspects of your trading application, and getting it quickly to market, because:
  - Caplin Trader contains customizable financial components
  - It provides domain specific abstraction and supports all common financial data structures
  - It makes it easy to add new asset classes, products, and trading models to your application
- ◆ You can create a unique and differentiated product offering with your own content, styling, and branding
- ◆ Caplin Trader has been proven in mission critical financial applications
- ◆ It has a multitude of features designed to deliver consistently low latency and high performance

### **Caplin Trader comes with its own environments for development, integration, and test, which means:**

- ◆ You can rapidly start development
- ◆ Your developers are more productive, as it is easy to build high quality, high performance code
- ◆ You can quickly and easily add new features to your trading application at any time

## 2.2 What does Caplin Trader consist of?

Caplin Trader consists of a run-time framework, a development framework, and documentation.

### Run-time framework

The run-time framework includes:

- ◆ An extensive and easily customizable set of display components
- ◆ A Layout Manager
- ◆ A set of Caplin Xaqua APIs (JavaScript libraries) for pricing, trading, and permissioning
- ◆ A JavaScript Preprocessor that ensures faster application start up and a smaller application payload in the browser
- ◆ A persistence database
- ◆ Full licenses for ExtJS and Emprise JavaScript Charts

### Development framework

The development framework helps you build and test a Caplin Trader application. It is built on open source technologies, and consists of the following:

- ◆ A build and dependency management framework (built on Apache ANT and Apache Maven)
- ◆ A unit testing framework (built on JUnit and Mock4JS)
- ◆ An acceptance testing framework (built on Selenium) with a business Domain Specific Language (DSL) for rapidly creating functional acceptance tests
- ◆ A performance and stress testing framework (built on WebDriver)
- ◆ Tools to support deployment for both test and live operation

### Documentation

- ◆ Comprehensive developer documentation, including “how to” guides and reference information for APIs and XML-based configuration

## 3 The User Experience

Trading applications built with Caplin Trader deliver a truly superior end-user experience. They have the rich set of features, functionality, and performance usually associated with real-time trading applications that are installed on the desktop.

Because a Caplin Trader application runs in a standard Web browser, once end-users have registered and been permissioned for trading, they can start trading straight away. There is no additional client software to be installed or configured on the desktop machine.

The user interface provides “windowing” style interaction. For example:

- ◆ There are many panels within the Web page
- ◆ Users can switch between layered panels via “tabs”
- ◆ Users can resize panels
- ◆ Users can drag and drop panels to other parts of the application
- ◆ Users can create new panels and populate them with tradable instruments

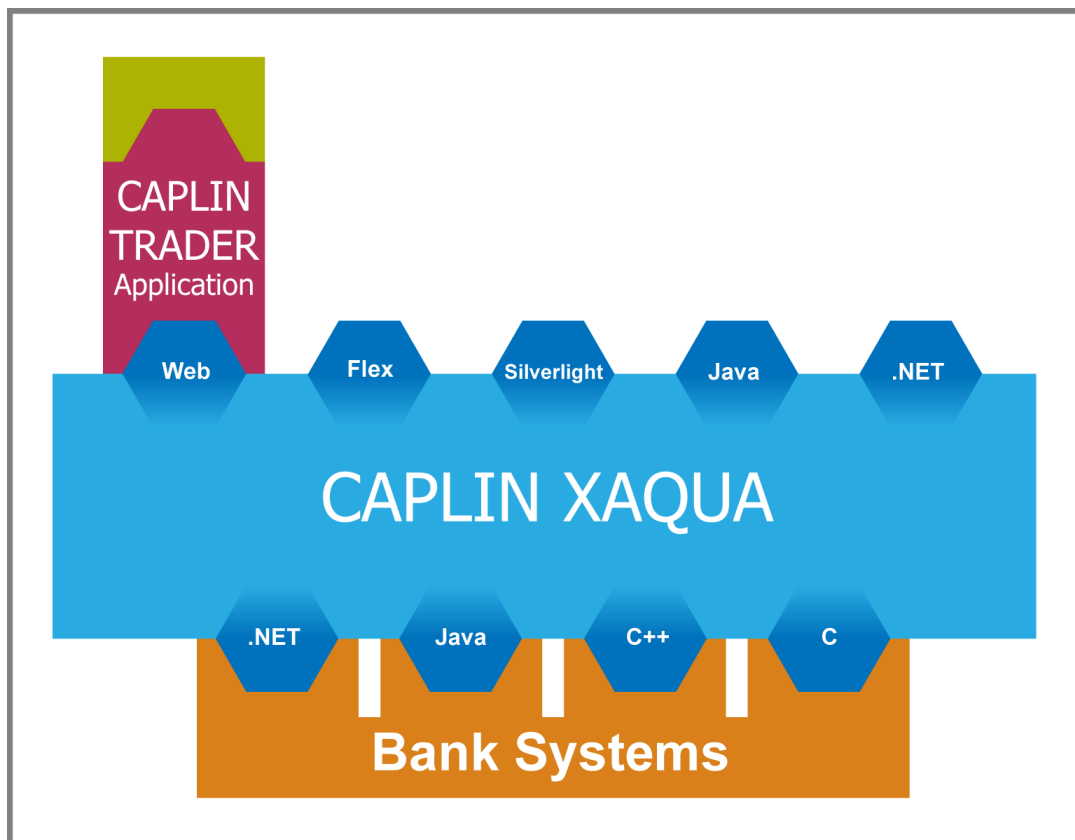
The appearance and behavior (“look and feel”) of a Caplin Trader application are highly customizable, allowing you to tailor the user experience to suit the needs of your business and your customers. It is easy to create optimized work flows that are tailored for your users.

For more information about customizing Caplin Trader applications, see [Building a Caplin Trader application](#)<sup>[24]</sup>, and in particular:

- ◆ [Defining the layout and overall appearance](#)<sup>[27]</sup>
- ◆ [Customizing display components](#)<sup>[34]</sup>
- ◆ [Adding your own component types](#)<sup>[42]</sup>

## 4 Architecture

Caplin Trader is a web application framework for constructing browser-based financial trading applications. The resulting **Caplin Trader application** is a **Caplin Xaqua client**, as shown in the following diagram.

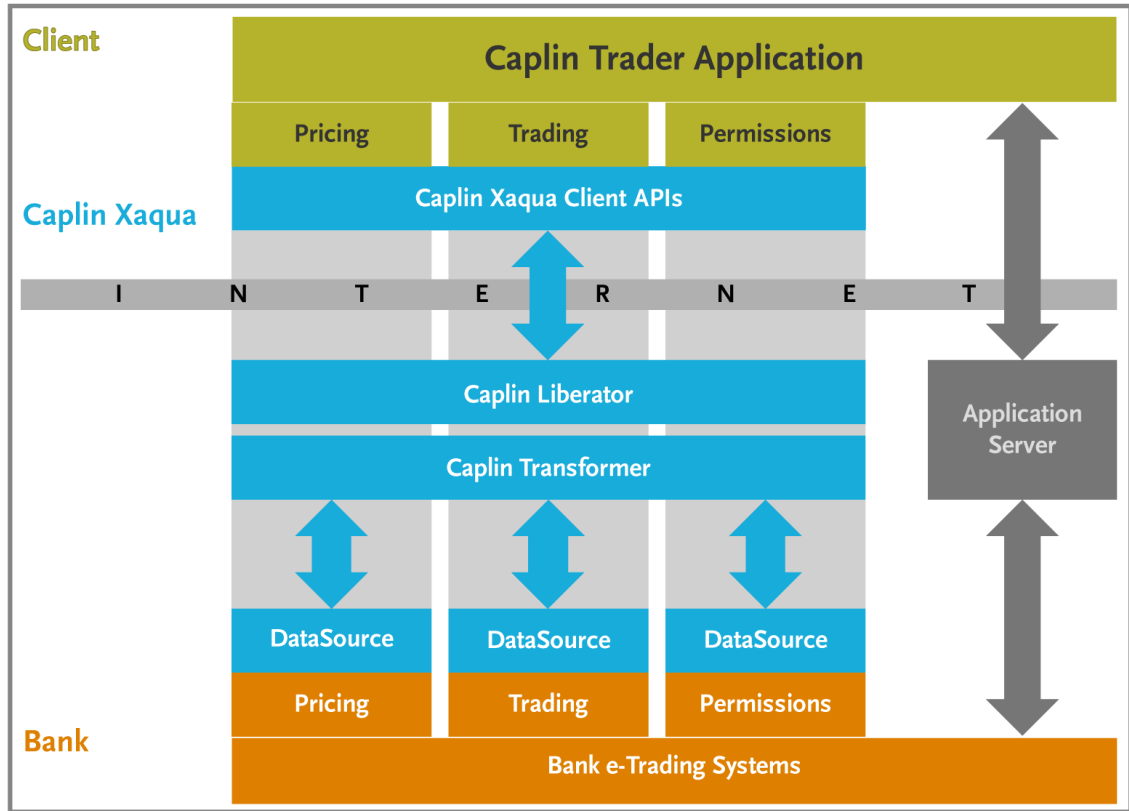


**An application built with Caplin Trader  
is a Caplin Xaqua client**

A Caplin Trader application is implemented using a combination of web technologies, such as HTML, CSS, JavaScript, and XML, that interact with the Caplin Trader framework. Caplin Trader communicates with Caplin Xaqua components across the Internet, and hence with the Bank's systems, allowing end-users to receive price updates in real time and trade a wide variety of financial instruments.

## 4.1 Caplin Trader and Caplin Xaqua

The following diagram shows in more detail, how a Caplin Trader application interacts with Caplin Xaqua (and hence the Bank's e-Trading systems), and with third party components.



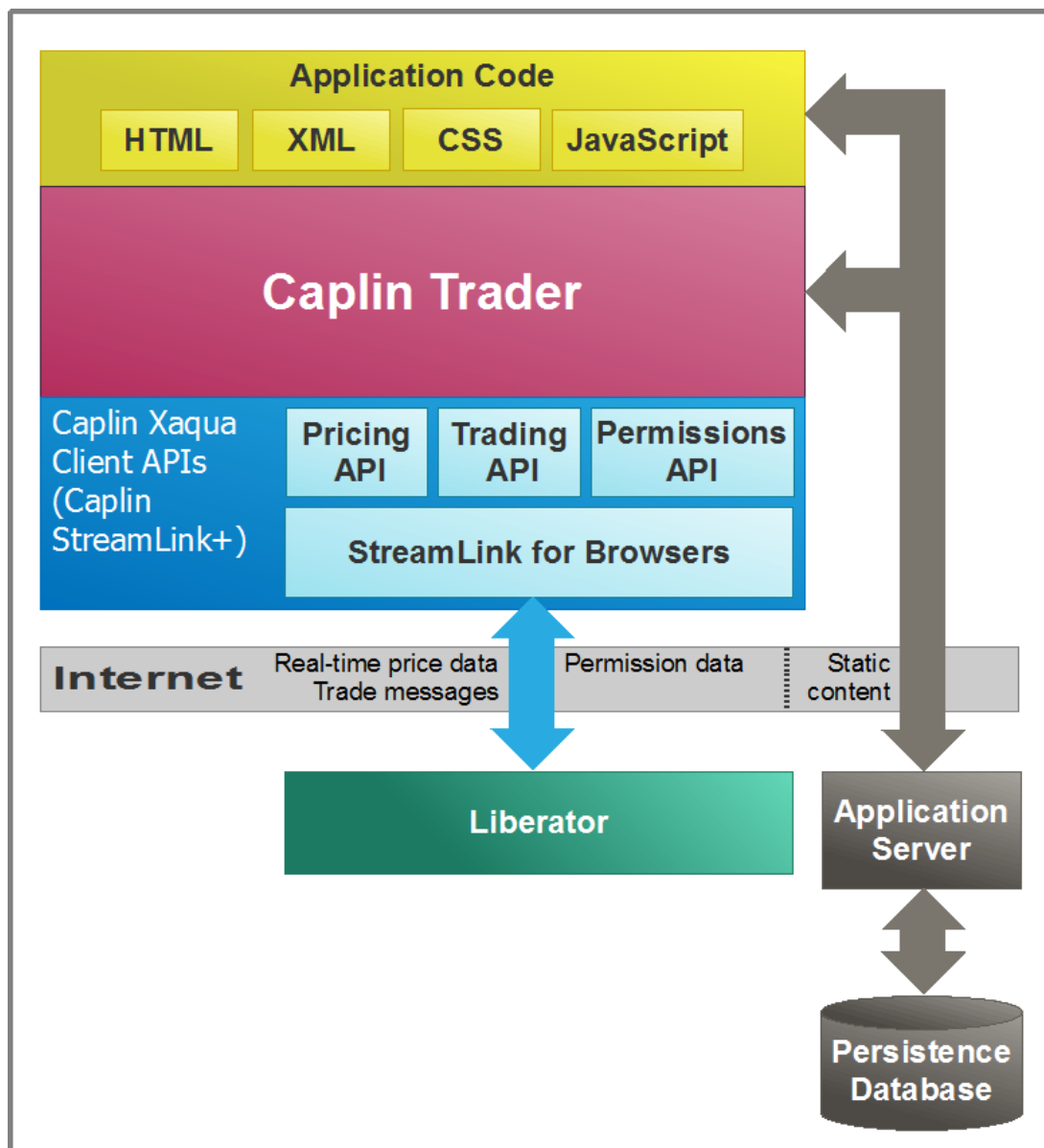
Caplin Trader's interaction with Caplin Xaqua

## 4.2 Caplin Trader application structure

The following diagram shows the structure of a Caplin Trader application.

The complete application consists of the **Application Code** written by the Bank's developers (yellow box), supported by **Caplin Trader** (pink box), which in turn uses the **Caplin Xaqua Client API** library (light blue boxes).

The diagram also shows the external components the application depends on: the **Liberator** server (turquoise box), and the **application server** and **persistence database** (gray boxes). Detailed interactions between components are not shown.



Caplin Trader application structure

The sections that follow describe in more detail the Caplin Trader application structure and the external components.

## Application Code

### Application Code

A Caplin Trader application is written in a mixture of HTML, CSS, XML, and JavaScript.

- ◆ HTML and CSS are used to define and style the visual aspects of the application.
- ◆ XML is used for customization (see [Caplin Trader API](#)<sup>[16]</sup>).
- ◆ JavaScript is used for customization and extension, through calls to the [Caplin Trader API](#)<sup>[16]</sup>, and by extending Caplin Trader [display components](#)<sup>[17]</sup>.

Existing web content and components written in other web technologies, such as Adobe Flex™ and Microsoft Silverlight™, can easily be embedded in a Caplin Trader application.

## Caplin Xaqua Client APIs and Liberator

### Caplin Xaqua Client APIs

### Liberator

A Caplin Trader application communicates with Caplin Xaqua across the Internet or a LAN by means of the **Caplin Xaqua Client APIs** that are provided with Caplin Trader. These APIs are also known collectively as **Caplin StreamLink+**. They include:

- ◆ A **Pricing API** that allows Caplin Trader to manage the real-time price data received from the **Liberator**.
- ◆ A **Trading API** that allows Caplin Trader to manage the flow of trade messages between Caplin Trader's [Trade Engine](#)<sup>[22]</sup> and the Trading DataSource(s) in Caplin Xaqua
- ◆ A **Permissions API** that allows the Caplin Trader application to obtain permissioning information from the Caplin Xaqua Permissions Subsystem, and hence make display components behave according to the permissions of the logged-in user
- ◆ **StreamLink for Browsers (SL4B)** This is an object oriented JavaScript API that gives access to Caplin Xaqua's RTTP functionality. This allows the Caplin Trader application to:
  - Connect to a Liberator
  - Subscribe to, and unsubscribe from, data
  - Receive streaming real-time price data, and other data, from the Liberator
  - Publish data to the Liberator
  - Access the transport mechanism for the exchange of trading and permissioning messages between Caplin Trader and the Liberator



## Application server

### Application Server

The **application server** is a Java application server. It can be any application server that runs on Java version 1.5 or higher and supports servlets and JavaServer Pages (JSP™).



It hosts the Caplin Trader application, the [JavaScript Preprocessor](#)<sup>23</sup>, and the application's resources, including images, HTML and CSS files, and configuration data. Some of these resources, such as saved user layouts, reside on the [persistence database](#)<sup>14</sup>.

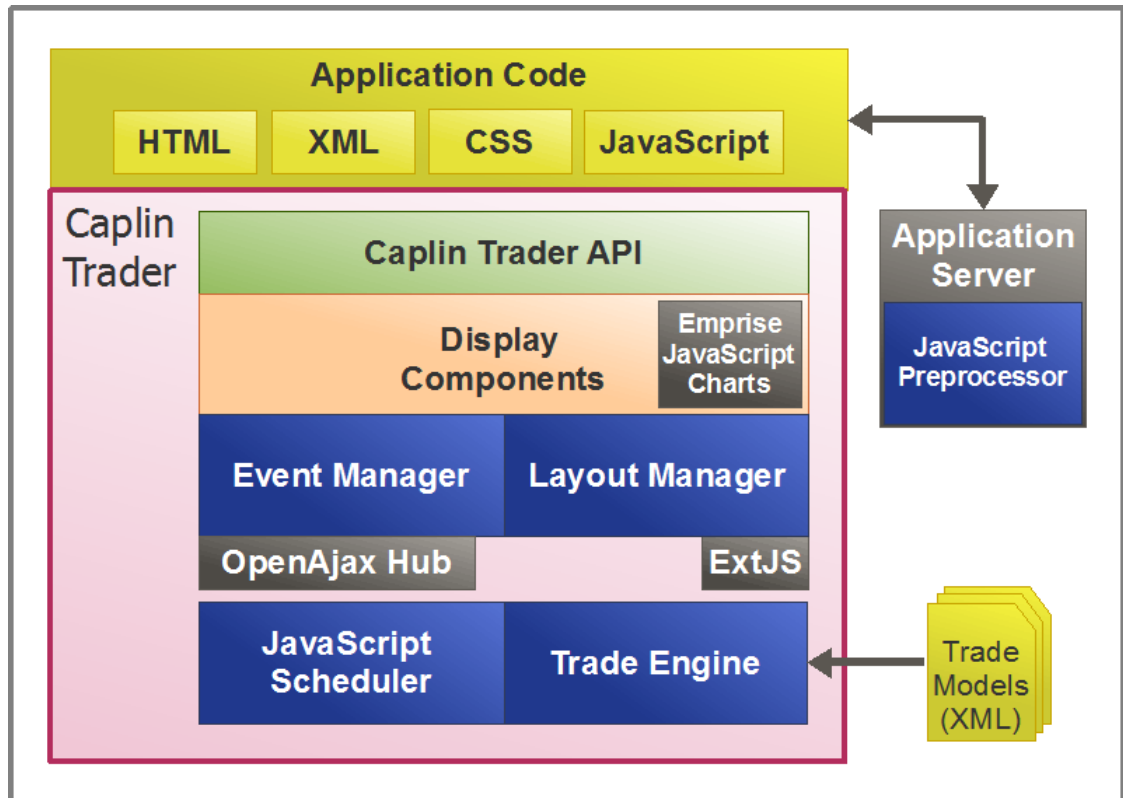
## Persistence database

### Persistence Database

The persistence database is a relational database, accessible from the application server, which stores user settings, preferences, and screen layouts. Caplin Trader does not use any advanced or proprietary database features or extensions, so the database server can be any RDBMS (for example, MySQL®, Oracle®, Sybase®), as long as it is supported by JBoss® Hibernate®.






### 4.3 Caplin Trader's internal architecture

The following diagram shows the internal architecture of Caplin Trader (inside the pink box ) and the relationship between Caplin Trader and the application code written by the Bank's developers (yellow box ). It also shows the JavaScript Preprocessor, which is part of Caplin Trader but runs outside the framework code.



Caplin Trader's internal architecture

The sections that follow describe Caplin Trader's internal modules in more detail.

- ◆ [Caplin Trader API](#) <sup>[16]</sup> 
- ◆ [Display components](#) <sup>[17]</sup> 
- ◆ [Internal modules](#) <sup>[22]</sup> , including
  - Layout Manager
  - Event Manager
  - JavaScript Scheduler
  - Trade Models and the Trade Engine
- ◆ [JavaScript Preprocessor](#) <sup>[23]</sup> 
- ◆ [Third-party modules](#) <sup>[23]</sup> 
  - Open Ajax Hub
  - ExtJS
  - Emprise Javascript Charts
  - Persistence database

## Caplin Trader API

### Caplin Trader API

The Caplin Trader API is the entry point into the Caplin trader framework that allows a bank's developers to implement a fully-functioned trading application. It uses two main technologies: XML and object-oriented JavaScript.

XML is used to customize Caplin Trader display components, such as Grids, Trees, and Charts, and to configure aspects of the overall page layout, sometimes in conjunction with CSS. You can implement much of a Caplin Trader application's appearance and functionality by writing XML to configure Caplin-supplied display components; for an example, see [Customizing display components with XML \(Grids\)](#) <sup>[35]</sup>.

Additionally, you can use Caplin Trader's object-oriented JavaScript API libraries to implement and modify the functionality of a Caplin Trader application by

- ◆ modifying and augmenting the behavior of Caplin-supplied display components,
- ◆ implementing brand new display components and integrating them into the Caplin Trader application.

The API libraries include utility classes for completing common tasks, classes for creating GUI controls, and the interfaces that your code must implement when you want to modify or augment a core component.

For more detail about using the JavaScript API, see [Adding your own component types](#) <sup>[42]</sup>.

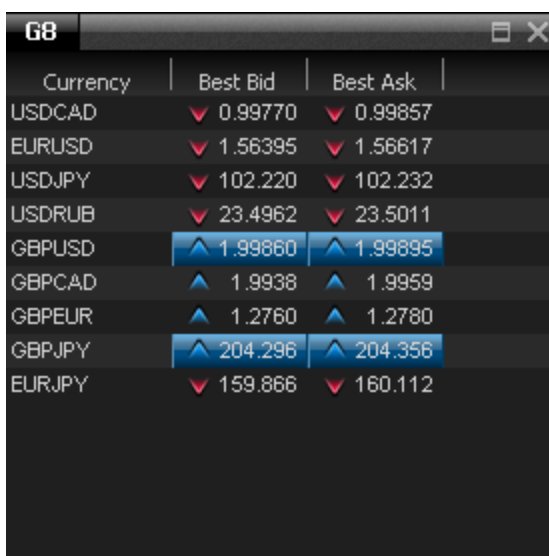
## Display components

### Display Components

Caplin Trader is supplied with a set of highly configurable and extensible display components that provide the essential features of a fully-functioned trading application. Most display components are defined as JavaScript classes that are instantiated as JavaScript objects at run time. They include the following components.

The screen images shown here are examples of how these display components could appear on the screen. You can customize the display components according to the look and feel you require for your Caplin Trader application (see [Customizing display components](#))<sup>34</sup>.

- ◆ **Grids** display information about financial instruments of all kinds, updated in real time.

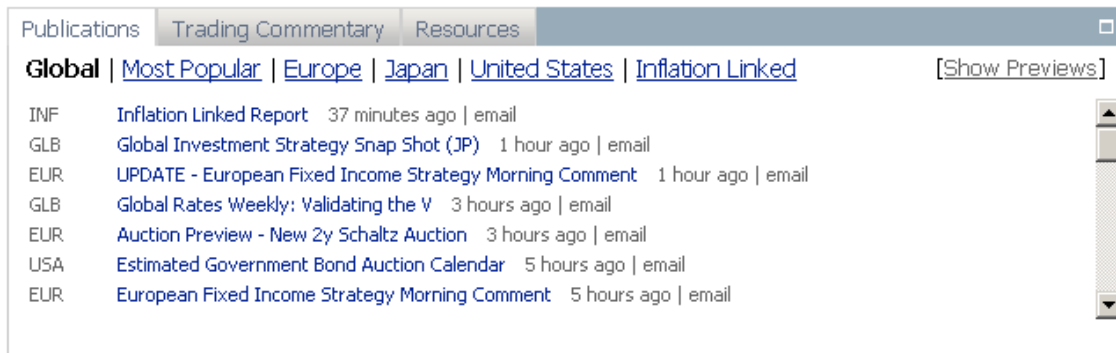


Currency	Best Bid	Best Ask
USDCAD	▼ 0.99770	▼ 0.99857
EURUSD	▼ 1.56395	▼ 1.56617
USDJPY	▼ 102.220	▼ 102.232
USDRUB	▼ 23.4962	▼ 23.5011
GBPUSD	▲ 1.99860	▲ 1.99895
GBPCAD	▲ 1.9938	▲ 1.9959
GBPEUR	▲ 1.2760	▲ 1.2780
GBPJPY	▲ 204.296	▲ 204.356
EURJPY	▼ 159.866	▼ 160.112

**A Grid displaying FX instruments**

Grids are an important part of any Caplin Trader application. They are very flexible and are optimized for displaying real-time data. They have a range of configurable capabilities, such as column resizing, column reordering, the ability to hide and reveal columns, and sorting and filtering of rows. Many of these capabilities are provided by standard decorators, and are easily configured using XML.

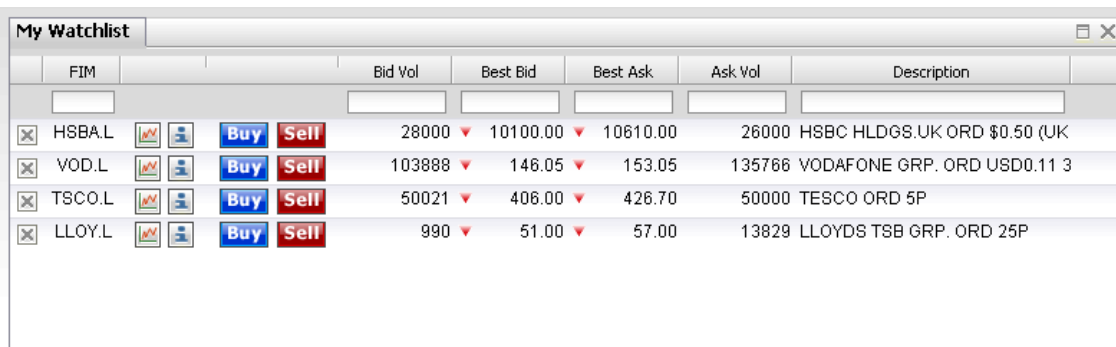
Grids can also display static data from virtually any source by specifying a custom data provider. For example, here is a grid that has been configured and styled to display research information:



### Grid displaying research information

For more information about using grids, see [Customizing display components with XML \(Grids\)](#) <sup>35</sup>.

- ◆ **Watchlists** are dynamically configurable Grids. An end-user can create a Watchlist, drag instruments into as desired, and save it for retrieval in subsequent sessions.



### A Watchlist of Equities

- ◆ **Trade Tiles** are typically used to display executable streaming prices for financial instruments, allowing one-click trading based on these prices.



### An FX Trade Tile

- ◆ **Trade Tickets** allow end-users to execute trades according to more complex Trade Models, such as Request For Quote (RFQ).

The screenshot shows an 'FX Trade Ticket' window with ID: 1277372969160. It contains the following fields and controls:

- Account: acct1 (dropdown)
- Currency Pair: EURUSD (dropdown)
- Trade Type: SPOT/FWD (dropdown)
- Amount: 500,000 (input field) with a EUR button
- Settlement: SPOT (dropdown)
- Date: 28 Jun 2010 (input field with calendar icon)
- SELL EUR button with rate 1.33540
- Spot Rate: 28 (displayed with a circular graphic)
- BUY EUR button with rate 1.33565
- Close and Cancel buttons at the bottom.

**An FX Trade Ticket  
(quote being obtained)**

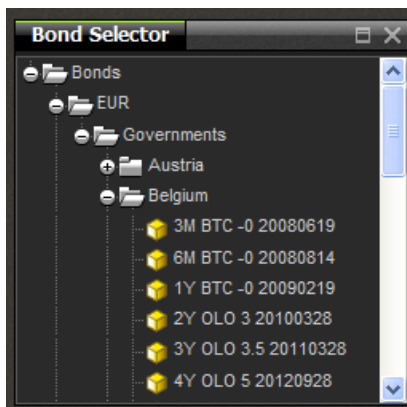
At the time of publication, Trade Tickets are built manually using HTML and CSS. Future releases of Caplin Trader will use the Form Builder framework to simplify development and customization of Trade Tickets (see [Using the Form Builder](#)<sup>[44]</sup>).

- ◆ **Trade Blotters** record and display information about trades in-progress and recently completed trades, and allow the end-user to search for, and display, trade history. They are a specialization of the Grid.

FX Blotter						
Trade ID	Currency Pair	Dealt Currency	Status	B/S	Amount	Rate
1277197643344	GBPCAD	GBP	Opened		500,000.00	
1277197643343	USDRUB	USD	Done	SELL	500,000.00	23.5592
1277197643339	EURGBP	EUR	Done	SELL	500,000.00	0.78302

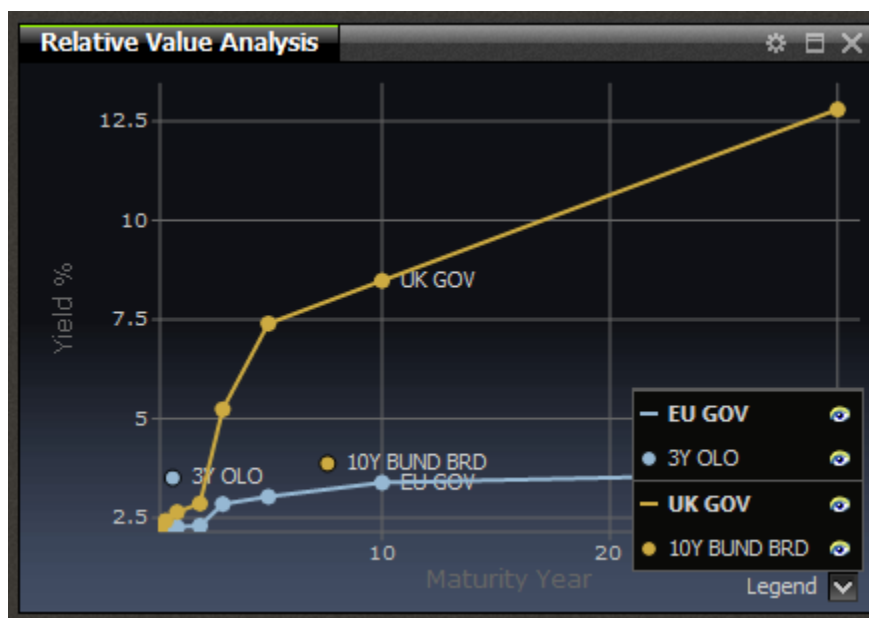
**Trade Blotter**

- ◆ **Tree Views** group large numbers of financial instruments into hierarchies, allowing the end-user to navigate through the hierarchy and select particular instruments. Caplin Trader renders trees using the **ExtJS** JavaScript Library (see [Third-party modules](#) <sup>[23]</sup>).



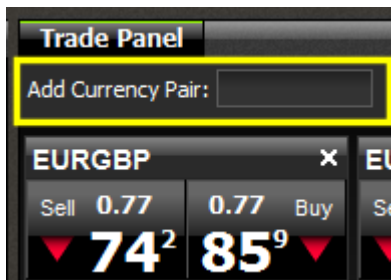
**Tree Display Component  
(FI Bond Selector)**

- ◆ **Chart Display Components** render one or more series of data in various chart formats. Chart components are based on the Emprise JavaScript Charts library (see [Third-party modules](#) <sup>[23]</sup>).



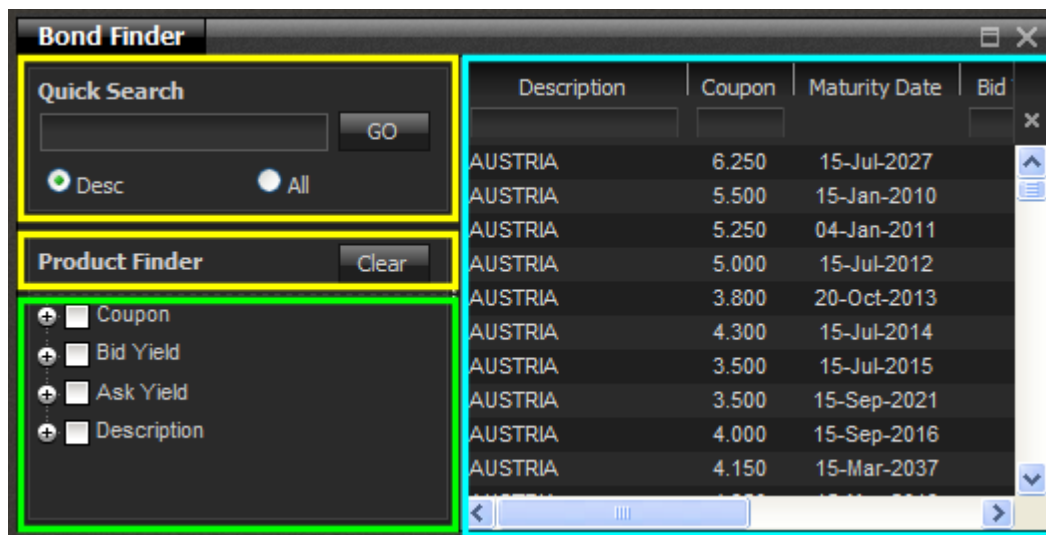
**Chart Display Component  
(FI Bonds – Relative Value Analysis)**

- ◆ **Simple Form Components** allow end-users of a Caplin Trader application to enter information into a displayed page. For example, an end-user can enter search criteria to find and display a particular set of financial instruments in a grid.



**Simple Form Component  
(outlined in yellow)**

- ◆ **Composite Components** group together other display components, and control and coordinate the behavior of these components in response to actions by the end-user.



**Example Composite Display Component  
(FI Bond Finder)**

The FI Bond Finder shown above, consists of two Simple Forms (outlined in **yellow**), a Tree (outlined in **green**), and a Grid (outlined in **blue**)

- ◆ **Element Renderers** render data within display components such as grids, trees, trade tickets, and trade tiles. For more information, see the section on [Element Renderers](#) <sup>38</sup>.



## Internal modules

Caplin Trader has a number of internal modules that provide the basic run-time capabilities of the application.

### Layout Manager

The Layout Manager controls the overall look and feel of the application. Its features include:

- ◆ Many panels within the Web page
- ◆ Layered panels, switchable via “tabs”
- ◆ Panel resize
- ◆ Drag and drop
- ◆ Menu definition and management

It manages the display components, and is highly configurable so you can customize the look and feel of your Caplin Trader application as much or as little as you want. Screen layouts are stored on, and retrieved from, the [persistence database](#) <sup>[14]</sup>.

Also see [Defining the layout and overall appearance](#) <sup>[27]</sup>.

### Event Manager

The Event Manager allows Caplin Trader components to communicate information (events or data) between each other and with external components, using a publish and subscribe model. A publishing component uses the Event Manager API to send out information that is tagged with an identifier. The publisher does not need to be aware of whether there are any consumers for the information. Any subscribers listening for that identifier, through the event Manager API, are notified of the new information immediately. The Event manager is based on the OpenAjax Hub; see [Third-party modules](#) <sup>[23]</sup>.

### JavaScript Scheduler

Different web browsers and browser versions vary in the efficiency and efficacy of their JavaScript scheduling engines. Caplin Trader includes a highly efficient browser-independent scheduler that ensures price updates are displayed promptly and trade messages are processed in a timely fashion, whatever browser type or version the user chooses from the set recommended by Caplin Systems (see [Appendix A: Recommended browsers](#) <sup>[49]</sup>).

### Trade Models

A Trade Model represents a type of financial trade, for example a Request for Quote (RFQ) or Executable Streaming Price (ESP). It consists of a number of states and transitions, and is defined in XML. The Trade Model controls the flow of a Trade by defining all the possible states the Trade can be in and the trade messages that cause transitions from one state to another. Any number of Trade Models can be configured.

A Trade model executes at both ends of the trade – that is, in the Caplin Trader application and in the Trading DataSource within Caplin Xaqua – so that the flow of trade messages is coordinated between the client and the bank's trading systems.

### Trade Engine

Caplin Trader has a built-in Trade Engine. This is a finite state machine that manages the state transitions and messaging for the client end of a trade, according to the Trade Model it is executing.

## JavaScript Preprocessor

### JavaScript Preprocessor

The JavaScript Preprocessor resides on the application server, and runs when the server receives a request from a client to load a Caplin Trader application. Its function is to ensure faster application start up and reduce the application payload in the client browser.

To achieve this, the Preprocessor parses the dependency tree of the JavaScript classes that comprise the application, and selects just those classes that are actually used. A particular application will typically not use all the Caplin Trader framework classes, so the Preprocessor ensures that only the classes required are actually sent to the client. It also concatenates the JavaScript class files held on the server into a single file, with the classes arranged in dependency order.

## Third-party modules

### OpenAjax Hub

The publish and subscribe event management features of the OpenAjax Hub are used to exchange event messages between Caplin Trader Display Components, and between Display Components and third-party components. See the Event Manager in [Internal Modules](#) <sup>[22]</sup>.

### ExtJS

Caplin Trader renders trees using the **ExtJS** JavaScript Library (<http://www.sencha.com>). The Caplin Trader license includes a license to use ExtJS in applications that are based on Caplin Trader.

### Emprise JavaScript Charts

Emprise JavaScript Charts is a JavaScript library for creating fully-featured interactive charts. It is the core technology used in Caplin Trader's Chart Display Component. The Caplin Trader license includes a license to use this library in applications that are based on Caplin Trader.

For more information on Emprise JavaScript Charts, see <http://www.ejschart.com>.

## 5 Building a Caplin Trader application

Caplin Trader provides a comprehensive set of building blocks for implementing your trading application:

- ◆ It has a set of customizable and extensible [display components](#)<sup>[17]</sup>
  - You can modify the look, feel, and behavior of the supplied Caplin Trader display components to meet the needs of your application
  - You can create your own display components, and integrate them into the application by implementing Caplin Trader's **Component** interface
- ◆ You can build Composite Components, combining Caplin Trader display components with your own
- ◆ You can use Element Renderers to determine how data is rendered in controls, and to respond to events on controls  
  
Element renderers are used extensively in Grids, Trade Tiles, and Trade Tickets.
- ◆ The Form Builder allows you to quickly create form-like display components, such as Trade Tickets and Trade Tiles
- ◆ You can define new Trade Models using XML
- ◆ You can directly call the underlying streaming data API (StreamLink for Browsers)  
  
You may need to do this if you are implementing a display component that needs to handle data in a different way to Caplin Trader's "built-in" component types (a "Chat" component for example).
- ◆ The development framework helps you build and test your trading application

The following sections explain how Caplin Trader can help you to build an effective trading application. Each successive section concentrates on a more detailed aspect of building the application.

### 5.1 Getting started

Caplin Trader is shipped in a kit that includes:

- ◆ The Caplin Trader framework software, including the API libraries
- ◆ A **Reference Implementation** of a Caplin Trader application
- ◆ All the necessary Caplin Xaqua components

Between them, these items provide both a fully-functioned example of a Caplin Trader application, and a basic environment for developing your own application.

The Caplin Xaqua components run on a Linux® or Sun Solaris™ box, and include:

- ◆ Caplin Liberator, configured for use with Caplin Trader
- ◆ Caplin Transformer, including business modules to support:
  - Data containers (for managing lists of instruments in a group)<sup>1</sup>
  - Container filtering and sorting (for filtering and sorting lists of instruments displayed in Grids)
  - Personal watchlists
  - Data normalization<sup>2</sup>
  - FX cross rates<sup>2</sup>
  - Price tiering<sup>2</sup>
  - Latency heartbeat (for measuring connection latency in Caplin Trader applications)

- ◆ Demonstration Pricing DataSource adapters for FX and FI, to supply live, real-time demonstration data to the Reference Implementation
- ◆ A demonstration Trading DataSource Adapter that allows you to try out the Reference Implementation's trading facilities

The Caplin Trader framework software and the Reference Implementation are usually installed on a Windows-based PC. The kit also includes a Java application server (Tomcat) and a SQL database server (MySQL), which you can install on the same PC to quickly get the Caplin Trader installation working. Alternatively, you can configure the installation to use your own Java application server and SQL-compliant database server.

Once the Caplin Trader kit has been installed, you can immediately run the Reference Implementation in a web browser to explore its capabilities, and you can explore its code. If you wish, you can use the Reference Implementation's code as the starting point for a Caplin Trader application that meets your own particular requirements. Alternatively, you can just write your own application code from scratch, referring to the Reference Implementation code for guidance as required.

---

**Footnotes:**

1. For a fuller definition of the **container** data item, see the [Glossary of terms and acronyms](#)<sup>52</sup>.
2. This Transformer module is for example/demonstration purposes, to support the Reference Implementation.

## 5.2 The Caplin Trader development framework

Caplin Trader comes with a development framework to help you build and test your trading application. The tools provided include a build and dependency management framework, a unit test framework, and an acceptance test framework.

### Build and dependency management framework

This is built on Apache ANT and Apache Maven. It provides the infrastructure – module dependency information and build scripts – to build a collection of source files into a deployable application (typically deployed as a WAR).

The build scripts are extensible, so you can amend them to fit with your existing build and release procedures. They will run "out-of-the-box" on a development machine, allowing you to easily build a deployable package, or to build and run subsets of the application for unit test and/or acceptance test purposes.

The framework can be fitted into a continuous build and test environment.

### Unit test framework

The unit test framework is built on JUnit and Mock4JS. It allows you to write and run unit tests for your JavaScript application code. You can run a single test or whole suites of tests. Tests and test suites can be invoked from build scripts, so unit tests can be automatically run within a continuous build and test environment.

### Acceptance test framework

The acceptance test framework is built on Selenium. It includes a business Domain Specific Language (DSL) for rapidly creating functional acceptance tests. Like the unit test framework, you can run single tests or test suites, and acceptance tests can be invoked from scripts for use in a continuous build and test environment.

### Using the JavaScript Preprocessor in development and test environments

When the JavaScript Preprocessor runs in "production mode", it selects just the files required and concatenates them into a single load file (see [JavaScript Preprocessor](#)<sup>[23]</sup>). The preprocessor can also be set to "development mode", which makes it load all the JavaScript code into the load file; this gives developers and testers access to all the code within the client.

### 5.3 Defining the layout and overall appearance

Caplin Trader's Layout Manager controls the overall look and feel of the application. One of the first development tasks is to define the required look and feel by configuring the Layout Manager. You can restrict the end-user's ability to configure the application layout as much as you wish. For example, it may be desirable for an application aimed at the lower-end retail market to have a largely fixed layout.

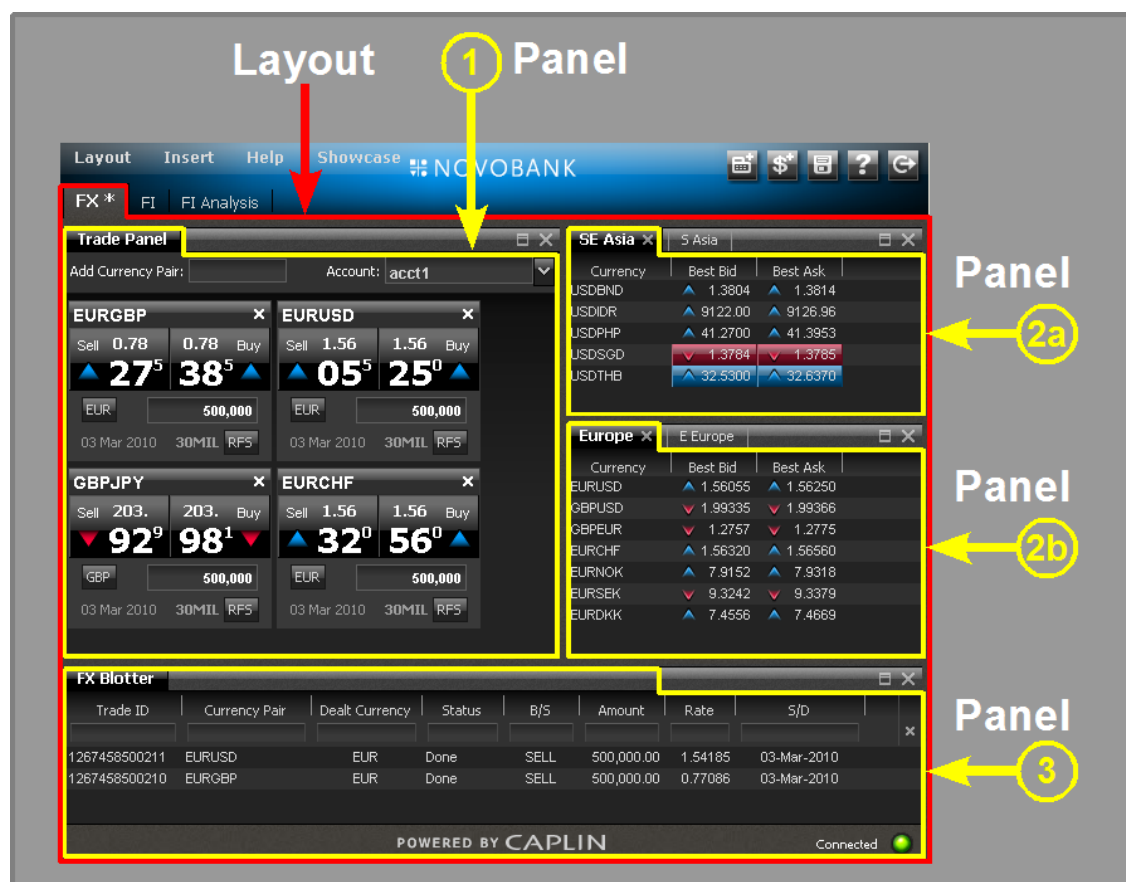
The following sections assume you are allowing maximum freedom to the user.

#### Layout concepts

##### Layouts and panels

A Caplin Trader application is presented to the end-user as a page in a web browser. The display components that comprise the user's view of the application are grouped into **layouts**.

The following diagram shows a layout (red border) that provides price data and trading capabilities for Foreign Exchange (FX) instruments. There are two other layouts available: Fixed Income (FI) and FI Analysis. The layouts are organized in tabs (FX, FI, FI Analysis), and the end-user can switch between layouts by selecting the relevant tab.



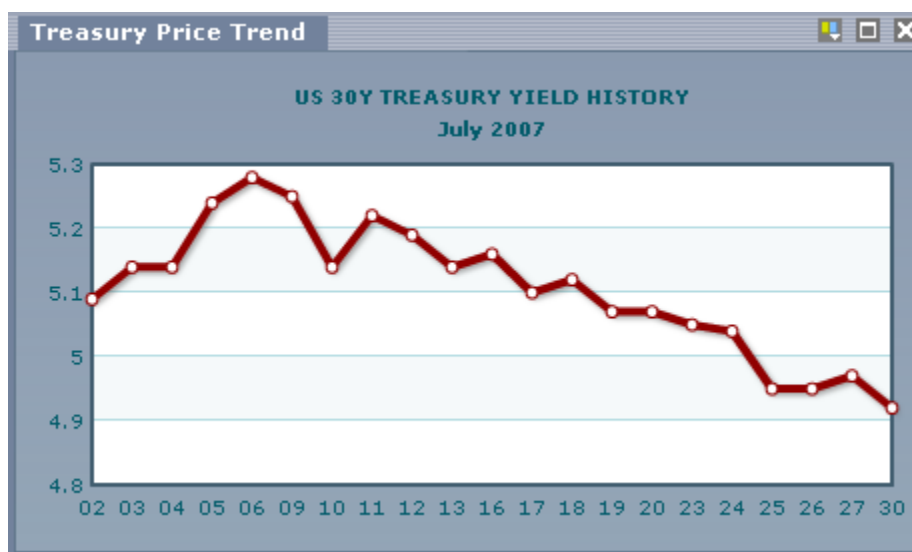
Caplin Trader Layout and Panels

The layout contains a number of **panels**. A panel is a rectangular area that gives the end-user a particular data view and/or access to particular functionality. In the previous diagram, the areas outlined with a **yellow border** are panels: the panel numbered **1** displays Trade Tiles through which the user can trade selected instruments with a single mouse click; panels **2a** and **2b** each display instrument prices in a grid layout where the prices are updated in real time; and panel **3** displays, in a blotter, the status and history of trades.

### Panel contents

The Layout Manager manages the panels, rendering HTML in each of them as supplied by the application. Typically the HTML for many of the panels is generated dynamically by Caplin Trader, through the JavaScript code of the display components, such as Grids and Trade Tiles. However, the HTML can also just be a static web page, either obtained locally, or through a URL that refers to an external web site.

Components implemented in other technologies can easily be embedded in panels – the following example shows a panel containing a chart implemented using Adobe Flash.

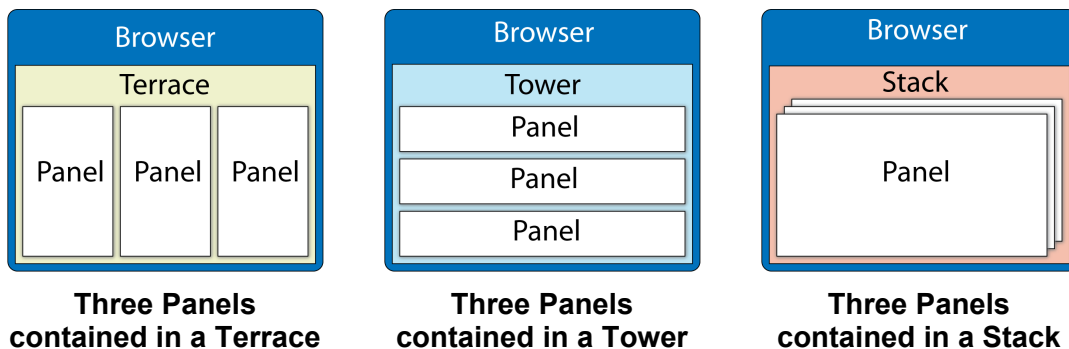


Panel containing a Flash Chart

### Terraces, towers and stacks

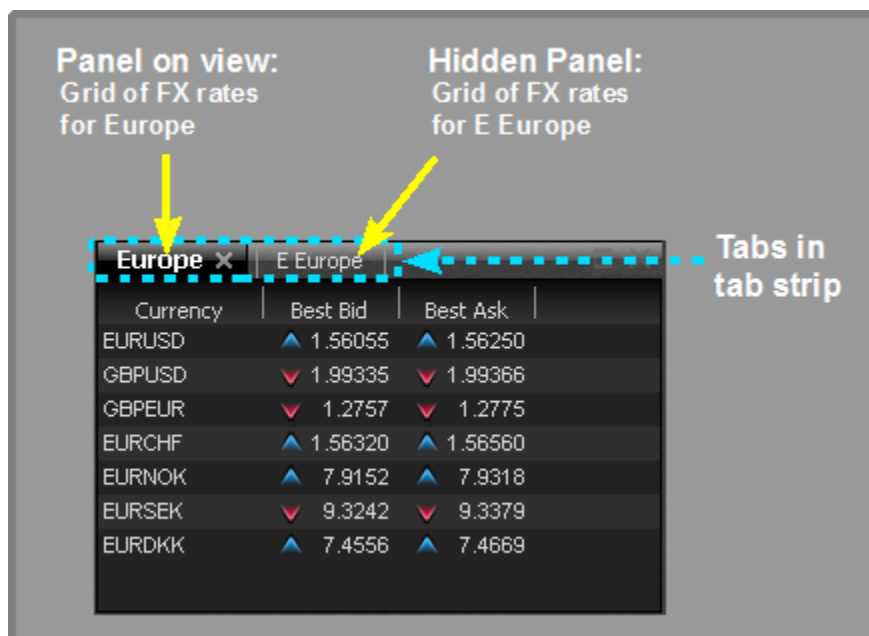
The Layout Manager allows panels to be grouped in various ways:

- ◆ The Terrace – panels grouped horizontally
- ◆ The Tower – panels grouped vertically
- ◆ The Stack – panels stacked on top of each other, with tabs to select them



The Layout Manager renders stacks with a tab at the top of each panel, so the end-user can bring one of the hidden panels to the front of the stack by selecting the relevant tab in the tab strip.

For example, the FX currency Grid called Europe in the previous Caplin Trader screen image (numbered 2b) is displayed in one panel at the top of a stack of two. The other (hidden) panel is E Europe.



Two panels in a stack with tab strip



## Changing the appearance of the application

The appearance of a Caplin Trader application is completely customizable. You can change how the panels, menus, and buttons look, the layout of the items displayed on the screen, and the appearance of individual display components.

Because a Caplin Trader application is web-based, its appearance is customized mostly by modifying a few XML and CSS files, and in some cases by modifying JavaScript code. For example, in the Reference Implementation, to change the logo on the top bar you edit an XML file, and to change the background image of the bar you edit a CSS file.

Other global features that can be changed by editing a single CSS file include: the colors of the panel background, border, and tabstrip, the background image of the menu bar, and the colors and fonts of the menus.

You can set up the application to have alternative themes, allowing the end-user to choose between them. For example:



Noir theme

and

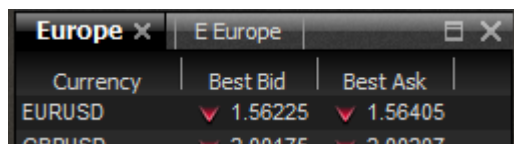





Pastel theme

## Layout Manager features

The Layout Manager gives a windows look and feel to the displayed page. For example, an end-user can :

- ◆ Manipulate a stack of panels like a window: resize the stack, move it by dragging, drag a single panel into a different stack, or even move a panel into a new position on the screen
- ◆ Use the panel buttons to maximize or remove panels:



- The  button maximizes the stack of panels to fill the whole working area of the Caplin Trader page
- The  button closes the whole stack of panels (or the single panel if there is only one tab)
- The  button (on the right hand side of the tab for the panel that is currently selected) closes just that panel, leaving the rest of the panels in place

- ◆ Drag on a tab to move its panel into one of the other stacks, or into a new position all by itself on the screen

Layouts can be saved to the persistence database and subsequently restored, so that end-users can customize their screens and re-use them in subsequent sessions.

## Customizing layouts

As a developer, you can configure which features the Layout Manager will support. You can pre-define the structure of a page and its layouts, and lock them down so that the end user has a fixed interface. Alternatively the predefined layouts could just be starting points for pages that end-users can subsequently customize for themselves. You can also change the look of the panels, windows, menus, and buttons.

You can define your Caplin Trader application so that it contains some pre-defined layouts when the end-user first loads the application, and allow the user to modify parts of the layout, or create new ones, to suit their own requirements.

Layouts are defined using XML, as shown in the following example.

### Layout configuration XML

```
<Tower splitters="true">
  <FrameItems>
    <Panel background="#ccf" src="content.html" caption="Window1">
    </Panel>
    <Terrace splitters="true">
      <FrameItems>
        <Frame src="http://www.caplin.com" caption="Window2">
        </Frame>
      </FrameItems>
      <Panel caption="FX Major">
        <state>
          <grid displayedColumns="description, rate">
            ...
          </grid>
        </state>
      </Panel>
    </Terrace>
  </FrameItems>
</Tower>
```

This configuration defines a layout as a Tower (**<Tower>...</Tower>**), consisting of:

- ◆ A panel called **Window1**  
(**<Panel ... caption="Window1">...</Panel>**)
- ◆ A Terrace  
(**<Terrace>...</Terrace>**)

The panel contains a locally sourced HTML page:

```
<Panel ... src="content.html" ...>
```

The Terrace consists of two panels, the first of which (**Window2**) contains content from a public URL. When a panel is to contain externally obtained content, it is defined using a `<Frame>` tag:

```
<Frame src="http://www.caplin.com" caption="Window2">
```

The second panel of the Terrace renders a Caplin Trader Grid display component:

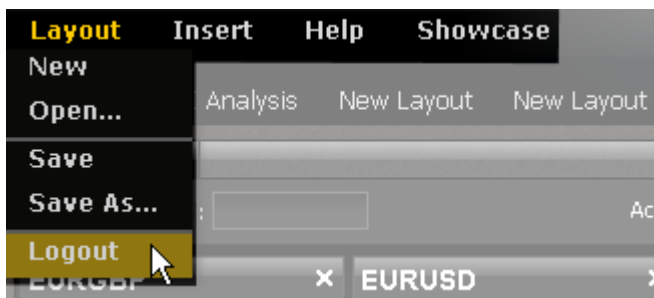
#### Panel containing a Grid

```
<Panel caption="FX Major" >
  <state>
    <grid displayedColumns="description, rate">
      ...
    </grid>
  </state>
</Panel>
```

The `<grid>` tag is part of the Grid configuration XML; see [Customizing display components with XML \(Grids\)](#)<sup>35</sup>.

#### Menu items

The application level menu items are defined in XML, so you can easily modify their content and appearance, and add new items. For example, the Reference Implementation has application menus called Layout, Insert, Help, and Showcase.



Example Application Menu

The following XML defines the top level of these application menu items:

```
<Tower>
  <FrameItems>
    <MenuBar handle_height="24" align="top">
      <MenuButton menu_id="layout" caption="Layout" />
      <MenuButton menu_id="insert" caption="Insert" />
      <MenuButton menu_id="help" caption="Help" />
      <MenuButton menu_id="showcase" caption="Showcase" />
    </MenuBar>
    <Tabstrip style="layout" handle_height="36" align="top" />
  </FrameItems>
</Tower>
```

This XML defines the items within the Layout menu:

```

<ContextMenu menuitem_height="22"
              menuitem_left_margin="9"
              menuitem_right_margin="15">
  <Decorators>
    <Border style="popup" border_bottom_width="8" border_right_width="8" />
  </Decorators>
  <!-- Layout menu -->
  <MenuItems>
    <MenuItem menu_id="new" label="New" context="layout">
      <loadPage page_id="NewLayout" />
    </MenuItem>

    <MenuItem menu_id="open" label="Open..." context="layout">
      <showDialog xref="Declarations/showDialog[@id='open-layout']"
                 defer="true" />
    </MenuItem>

    <MenuItem label="separator" context="layout">
    </MenuItem>

    <MenuItem menu_id="save" label="Save" context="layout">
      <saveLayout />
    </MenuItem>

    <MenuItem menu_id="save_as" label="Save As..." context="layout">
      <showDialog xref="Declarations/showDialog[@id='save-layout']"
                 defer="true" />
    </MenuItem>

    <MenuItem label="separator" context="layout">
    </MenuItem>

    <MenuItem menu_id="logout" label="Logout" context="layout">
      <eval xref="/Applications/Declarations/action[@id='logout']"
           defer="true" />
    </MenuItem>
    ...
  </MenuItems>
</ContextMenu>

```

## 5.4 Customizing display components

Caplin Trader's display components are highly customizable. The look and behavior of most types of component can be easily customized using XML. In addition, the Caplin Trader Javascript API allows you to customize features of a component that cannot be specified through XML, and implement new component types by extending the core API classes.

Third party and custom display components can be built into layouts, or they can be added dynamically through user interaction with menus or other controls. Such components can be hooked into the Event Manager using the Caplin Trader API. This allows the component to handle events such as 'resize', and takes full advantage of Caplin Trader's runtime features.

The Caplin Trader Reference Implementation includes a showcase menu containing examples of components that have been customized to produce a different look and feel to the components in the main layout of the application.

For example, the "standard" FX Grid in the Reference Implementation looks like this:

Currency	Best Bid	Best Ask
USDCAD	0.99770	0.99857
EURUSD	1.56395	1.56617
USDJPY	102.220	102.232
USDRUB	23.4962	23.5011
GBPUSD	1.99860	1.99895
GBPCAD	1.9938	1.9959
GBPEUR	1.2760	1.2780
GBPJPY	204.296	204.356
EURJPY	159.866	160.112

Standard FX Grid

This grid can be customized (using [element renderers](#) <sup>387</sup>) to display the price pips in a larger font:

Currency	Best Bid	Best Ask
EURUSD	1.56 <b>395</b>	1.56 <b>617</b>
USDJPY	102. <b>220</b>	102. <b>232</b>
GBPUSD	1.99 <b>845</b>	1.99 <b>876</b>
USDCHF	1.00 <b>180</b>	1.00 <b>208</b>
AUDUSD	0.91 <b>790</b>	0.91 <b>817</b>
USDCAD	0.99 <b>770</b>	0.99 <b>857</b>
NZDUSD	0.78 <b>510</b>	0.78 <b>542</b>
EURJPY	159. <b>866</b>	160. <b>112</b>
EURGBP	0.78 <b>246</b>	0.78 <b>369</b>
EURCHF	1.56 <b>676</b>	1.56 <b>942</b>

FX Grid with pips  
in large font

## Customizing display components with XML (Grids)

Here is an example of how Caplin Trader components can be customized using XML.  
In this case the components are two Grids:

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

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

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

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

### XML defining the Grids

```
<gridDefinitions xmlns="http://www.caplin.com/CaplinTrader/grid">
  <dataProviderMappings>
    <dataProviderMapping id="rttcpContainerGridDataProvider"
      className="caplin.grid.RttcpContainerGridDataProvider" />
  </dataProviderMappings>
  <decoratorMappings>
    <decoratorMapping id="dragDecorator"
      className="caplin.grid.decorator.DragDecorator" />
  </decoratorMappings>
  <templates>
    <gridTemplate id="All">
      <decorators>
        <dragDecorator />
      </decorators>
    </gridTemplate>
    <gridTemplate id="FX" baseTemplate="All" displayedColumns="description,rate">
      <columnDefinitions>
        <column id="description"
          cellRenderer="fx-description"
          fields="InstrumentDescription"
          displayName="Currency" width="70"
          mandatory="true"/>
      </columnDefinitions>
    </gridTemplate>
  </templates>
</gridDefinitions>
```

```

        <column id="rate"
            cellRenderer="fx-spread"
            fields="BestBid,BestAsk"
            displayName="Rate"
            width="100"/>

        <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"/>
    </columnDefinitions>
</gridTemplate>
</templates>

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

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

```

Here is a brief explanation of what the above XML does:

- ◆ **<dataProviderMappings>** contains the definition of the data provider that supplies the data to a grid
- ◆ **<decoratorMappings>** contains the definition of a grid decorator called `dragDecorator`  
A drag decorator allows an instrument to be dragged out of a grid, so it can be dropped into another screen component, such as a trade panel. Also see the section on [Decorators](#) <sup>[40]</sup>.
- ◆ **<templates>** contains the definitions of two grid templates (**<gridTemplate>**)  
Grid templates allow grids to be defined in an inheritance hierarchy.
- ◆ **<grids>** defines the actual grids that can be added to layouts  
In this case, there are two grids, Major FX (**<grid id="FX.Major" ...>**) and Minor FX (**<grid id="FX.Minor" ...>**), each of which inherits its characteristics from the "FX" grid template, which in turn inherits from the "All" template.
  - Both grids display description and rate information in columns of the same widths. To change these aspects of the appearance of both these grids, you only need to modify the definition of the parent "FX" grid template.
  - Both grids also allow the end-user to drag an instrument out of the grid. They inherit this behavior from the top level "All" template.

- Each grid obtains its data from a different container (container="/CONTAINER/FX/Major", container="/CONTAINER/FX/Minor"), so the two grids display different sets of FX instruments.
- The XML for the "FX.Minor" grid also defines a header renderer that is applied to the "description" column, overriding the renderer that is defined for this column in the parent "FX" grid template:  

```
<column id="description" headerRenderer="inlineTextFilter" />
```
- The `inlineTextFilter` renderer provides a text box into which the end user can enter search criteria to select the rows displayed in the grid:

The image shows a screenshot of a 'Minor FX' grid window. On the left, a red-bordered box contains the XML snippet: `<column id="description" headerRenderer="inlineTextFilter" />`. A red line connects this box to a search input field in the grid's header, which is also circled in red. The grid displays a table of currency pairs and their rates.

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

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

Also see the section on [Element Renderers](#) <sup>38</sup>.



## Element renderers

Caplin Trader uses Element Renderers to render data within display components such as Grids, Trees, Trade Tickets, and Trade Tiles. The data may be real-time prices or static information, and can come from a variety of sources. Element Renderers can also be used in custom display components and display components built using the Form Builder (see [Using the Form Builder](#) <sup>44</sup>).

Element Renderers allow you to define templates that are used throughout the application. These templates specify:

- ◆ The format and style of the data that is rendered in a control  
For example, format could be the number of decimal places displayed, style could be underlined or bold.
- ◆ 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.
- ◆ Input controls  
For example, input through a text box.

Here is an example of a grid that displays indicative prices for four FX currency pairs. It uses element renderers to determine the appearance of the price data.

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

**Renderers in a 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:

- [1] Open a Trade Ticket for a currency pair when the end-user clicks the indicative price for that currency pair. The price is underlined when it can be traded.
- [2] Flash prices with a green background for half a second when the indicative price increases (and flash prices with a red background for half a second when the indicative price decreases).
- [3] Render stale prices with a strike through.

### Defining and using element renderers

An Element Renderer configuration is defined using XML definitions, and renderer instances are created by the Element Renderer Framework at runtime, as required by the display components. 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 the Element Renderer JavaScript classes provided with Caplin Trader.

The following example shows a typical Element Renderer definition in XML.

### 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>
```

A display component can use a particular renderer simply by specifying the renderer in the component's configuration XML. For example, to use the above **fx-price** renderer in a Grid:

### XML configuration for a Grid that uses the fx-price renderer

```
...
<columnDefinitions>
  <column id="bestbid"
    cellRenderer="fx-price"
    fields="BestBid"
    displayName="Best Bid"
    width="100" />
  ...
</columnDefinitions>
...
```

## Decorators

Caplin Trader uses Decorators to 'decorate' display components with new visual and behavioral features. Whereas an element renderer modifies the appearance and/or behavior of an individual element within a display component (such as a cell in a grid), a decorator is applied to the display component as a whole.

For example, some of the Decorators that can be applied to Grids are:

- ◆ **ColumnMenuDecorator**

Adds a drop-down menu to each column of the Grid; the menu allows the user to add and remove columns.

- ◆ **ColumnResizingDecorator**

Allows the width of a Grid column to be changed by dragging the edge of the column header.

- ◆ **DragDecorator** and **DropDecorator**

Allow instruments to be dragged out the Grid and dropped into the Grid.

- ◆ **DisconnectedDecorator**

Displays a message to the end user if there is no connection to a DataSource when the Grid initially opens.

### Using pre-defined Decorators

You can add a Decorator to a display component using XML configuration. For example, the following fragment of Grid configuration XML adds the **ColumnMenuDecorator** to a Grid template. All grids that inherit from this template have this Decorator.

### XML for adding the ColumnMenuDecorator to Grids

```
...
<decoratorMappings>
  <decoratorMapping id="columnMenuDecorator"
    className="caplin.grid.decorator.ColumnMenuDecorator" />
</decoratorMappings>
...

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

### Implementing your own Decorators

You can write your own Decorators by implementing the appropriate Decorator interface in JavaScript.

For example, assume you want to add a button to a Grid, where the button pops up a menu when the user clicks on it. To achieve this, you could write a **PopupMenuDecorator** that implements the Caplin Trader **GridDecorator** interface. The code for this would typically look like the following:

**JavaScript code that implements a PopupMenuDecorator for grids**

```

caplin.namespace("caplin.grid.decorator");

caplin.include("caplin.grid.decorator.GridDecorator", true);
caplin.include("caplin.grid.GridViewListener", true);

caplin.grid.decorator.PopupMenuDecorator = function(mConfig)
{
};
caplin.implement(caplin.grid.decorator.PopupMenuDecorator,
                 caplin.grid.decorator.GridDecorator);

caplin.grid.decorator.PopupMenuDecorator.prototype.setGridView = function(oGridView)
{
    this.m_oGridView = oGridView;
};

caplin.grid.decorator.PopupMenuDecorator.prototype.onContainerHtmlRendered = function()
{
    var eMyButton = document.createElement("DIV");
    eMyButton.className = "mybutton";
    var eHolderElement = this.m_oGridView.getRowContainerElement();
    // Adding the popup menu button to the grid
    eHolderElement.appendChild(eMyButton);
};

```

**PopupMenuDecorator** implements the **GridDecorator** interface:

```

...
caplin.implement(caplin.grid.decorator.PopupMenuDecorator,
                 caplin.grid.decorator.GridDecorator);
...

```

The implementation includes the **setGridView()** method, which obtains a reference to the **GridView** object that is responsible for displaying the rows of the Grid:

```

...
caplin.grid.decorator.PopupMenuDecorator.prototype.setGridView = function(oGridView)
{
    this.m_oGridView = oGridView;
};
...

```

This allows the **PopupMenuDecorator** access to the **GridView**, so it can decorate the **GridView** as required.

**setGridView()** is automatically called by the Caplin Trader framework when it parses the XML that adds the **PopupMenuDecorator** to the Grid:

```

...
<decoratorMappings>
  <decoratorMapping id="popupMenuDecorator"
                   className="myapp.grid.decorator.PopupMenuDecorator" />
</decoratorMappings>

<grid id="FX.Major" displayName="Major FX" baseTemplate="FX">
  ...
  <decorators>
    <popupMenuDecorator />
  </decorators>
  ...
</grid>
...

```

**setGridView()** cannot itself decorate the **GridView**, because the HTML for the Grid is not yet rendered when the Grid configuration XML is parsed. Instead, the **PopupMenuDecorator** implements the **onContainerHtmlRendered()** listener. This call-back is invoked when the grid view has rendered its container HTML, and at this point **PopupMenuDecorator** can add the button that pops up the menu:

```
...
caplin.grid.decorator.PopupMenuDecorator.prototype.onContainerHtmlRendered = function()
{
    var eMyButton = document.createElement("DIV");
    eMyButton.className = "mybutton";
    var eHolderElement = this.m_oGridView.getRowContainerElement();
    // Adding the popup menu button to the grid
    eHolderElement.appendChild(eMyButton);
};
```

If you want to add a button to every row of the grid instead, you can use an element renderer (see [Element renderers](#)<sup>[38]</sup>).

## 5.5 Adding your own component types

To add your own type of display component, you write a JavaScript class defining the component type, define the XML that configures an instance of the component, and register the component type with Caplin Trader.

Here is some example JavaScript code for creating and adding a new component type called **MyComponent**:

```
caplin.namespace("xyztrader");

caplin.include("caplin.component.Component", true);
caplin.include("caplin.component.ComponentFactory", true);

xyztrader.MyComponent = function()
{
    // Construct the component here -
    // including storing any arguments that were passed in.
};

caplin.implement("xyztrader.MyComponent", "caplin.component.Component");

// ...
// Implementation of the caplin.component.Component methods.
// ...

// Static factory method for creating an instance of
// xyztrader.MyComponent from the specified XML string.

xyztrader.MyComponent.createFromSerializedState = function(sXml)
{
    // Process the XML to extract any relevant state information.
    return new xyztrader.MyComponent();
};

// Register the component type's
// createFromSerializedState() static factory method
// with Caplin Trader's ComponentFactory.
// The root node of the XML configuration for the component
// is called <myComponent>
```

```
caplin.component.ComponentFactory.registerComponent(  
    "myComponent",  
    xyztrader.MyComponent.createFromSerializedState);
```

In more detail:

- ◆ The new component type implements the Caplin Trader component interface, **caplin.component.Component**.

The methods of this interface include callbacks that inform the component of events resulting from the end-user's interaction with it. Examples of these callbacks are:

- **onOpen()** – invoked when the component is first displayed within the container
- **onResize()** – invoked when the dimensions of the container within the windowing environment have changed
- **onMaximize()** – invoked when the container has been maximized by the windowing environment

- ◆ Design the schema for the XML that defines an instance of the component.

For example, see the Grid XML in [Customizing display components with XML \(Grids\)](#)<sup>35</sup>.

- ◆ Implement a static factory method **createFromSerializedState()**, that returns an instance of the component.

This method takes a string argument that is the XML defining (according to your schema) the component instance to be constructed. The method must create an object that has the characteristics defined by the XML.

- ◆ In your Caplin Trader application, register the static factory method with Caplin Trader by calling **caplin.component.ComponentFactory.registerComponent()**.

This call takes:

- The name of the root node of the XML that defines the component instance to be constructed (for example, the name passed is "myComponent" if the XML root node is <myComponent>)
- The name of the static factory method for creating the component

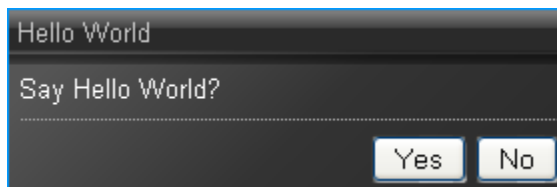
When the component type is registered at run-time, Caplin Trader automatically creates instances of the component as required. For example, this can be when the application starts up and Caplin Trader sets up predefined layouts according to the defined XML configuration. If there is XML configuration defined for your component type, Caplin Trader will include those components in the predefined layouts as required.

Alternatively, a component can be created when a layout is restored from the persistence database. If your component type implements the **getSerializedState()** method, instances of it can be saved to the persistence database in the XML format when layouts are saved. When Caplin Trader restores a layout, it supplies **createFromSerializedState()** with the XML for the object's serialized state.

## 5.6 Using the Form Builder

The Form Builder is a software framework and tools for quickly designing and implementing forms for a Caplin Trader application.

Here is a simple example of a form implemented using the Form Builder:



A simple "Hello World" form

The form is defined as a template, using HTML markup with some extensions:

```
<template name="form:hello-world">
  <p>Say Hello <strong data-name="who">World</strong>?</p>
  <hr>
  <fieldset class="buttons">
    <button command="yes" data-enabled="yes-enabled">Yes</button>
    <button command="no">No</button>
  </fieldset>
</template>
```

The `<template>` tag defines the name of the form.

The `data-name` attribute identifies as a “data point” the part of the form following the fixed text “Say hello”. This is where the application can dynamically change the contents of the text “World” to something else – for example, “Universe”.

The data on the form could be updated from a real-time feed. For example in a trading application, the form could be a Trade Ticket, and the buy/sell price fields would be defined as data points that are updated in real time with prices streamed from the Liberator.

The `command` attribute of each button (`command="yes"`, `command="no"`) specifies the name of an event that is generated when the button is selected.

The `data-enabled` attribute defines a boolean data point called `yes-enabled`. The button can be enabled and disabled programmatically through this data point – set `yes-enabled` to `false` and the button is disabled. (The default initial state of the button, defined by the HTML `<button>` tag, is “enabled”.)

The form template does not itself specify the CSS class that styles the form; this is done in a `<caplin:Form>` entry in the XML that configures themes. This approach allows alternative styles to be easily applied to the same form.

### Styling the form in the theme definition XML

```
<caplin:Form
  dialog="true" width="300" height="400"
  caption="Hello World"
  template="form:hello-world"
  class="alert-background"
  presentation="caplinx.demo.HelloController" >
  <property setter="setWho" value="World"/>
</caplin:Form>
```

In the above example, the form's appearance is styled through the CSS class called `alert-background`.

The `template` attribute specifies the name of the template that defines the form, and the `presentation` attribute specifies the name of the JavaScript class that implements the Form Controller (see [How does the Form Builder work?](#)<sup>[46]</sup>).

### Complex forms

You can use the Form Builder to quickly and efficiently create complex forms, such as this Trade Ticket:

The screenshot shows a dark-themed 'FX Trade Ticket' dialog box. At the top, it displays 'ID: 1277372969160'. The form contains several sections: 'Account' with a dropdown set to 'acct1'; 'Currency Pair' with a dropdown set to 'EURUSD' and a currency symbol '\$'; 'Trade Type' with a dropdown set to 'SPOT/FWD'; a section for 'Amount' (500,000) and 'EUR' currency; 'Settlement' with a dropdown set to 'SPOT'; and 'Date' (28 Jun 2010) with a calendar icon. Below these are three buttons: 'SELL EUR' with the rate '1.33540', a central 'Spot Rate' display showing '28', and 'BUY EUR' with the rate '1.33565'. At the bottom right are 'Close' and 'Cancel' buttons.

A complex form  
(Trade Ticket)



### How does the Form Builder work?

The Form Builder is based on a Model View Controller (MVC) architecture; it consists of a Form View and Form Controller, and optional Form Models.

At run time, a Form View is constructed according to the specification in the form template. The Form View manages the appearance and behavior of the form as rendered on the page.

To plug the form into the application, you write a Form Controller class. Its functions include:

- ◆ Ensuring that server-side updates to displayed data are passed on to the Form View
- ◆ Handling events notified by the Form View (such as the end-user clicking on a button, or entering data in a form field)
- ◆ Mapping button states and data points in the form template to states and data in the business domain, and managing the interaction between them.
- ◆ Managing the interaction between the Form View and the business logic

For example, prices quoted on a Trade Ticket would be mapped by the controller to the real-time price data being fed to the application from a Liberator. Changes in the state of the Trade Model would be mapped to changes in the state of the Trade Ticket, and visa versa, so (for example) the controller would disable the ticket's Get Quote button when the Trade Model state changed to "obtaining quote".

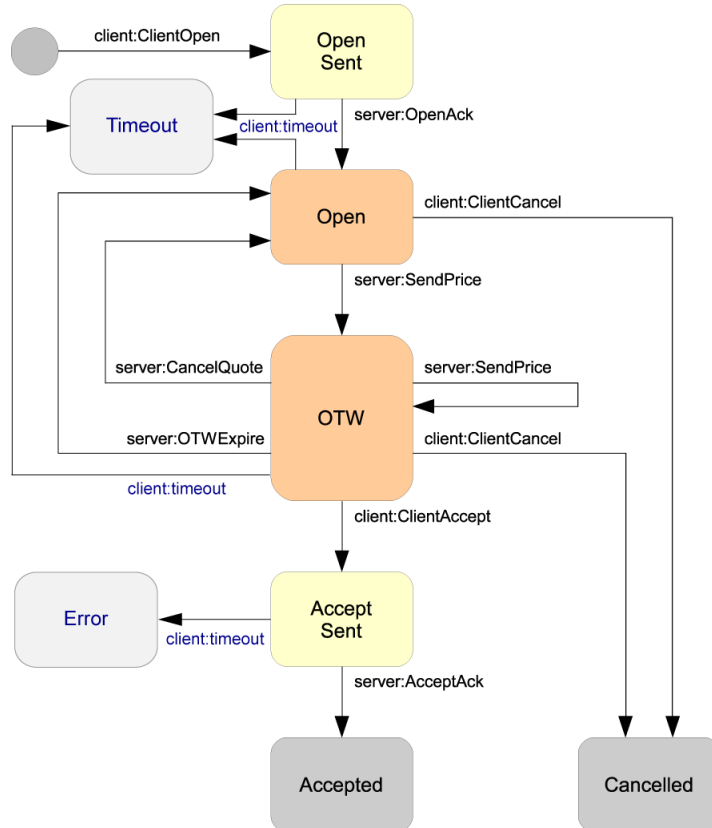
Form Models are optional. They model the business data and states that the Form Controller maps to the Form View. Examples of Form Models are Trade Models and permissions models.

### The Form Harness

The Form Builder includes a Form Harness tool to assist the conceptual design and development of forms before putting them into production. The harness allows you to see a visual representation of the form without the business logic, and to interact with it. You can ensure that the correct events are being fired and check that communication between the form and the rest of the application is working as expected.

## 5.7 Defining a Trade Model

New Trade Models are defined using XML. For example, here is a simple RFQ (Request For Quote) Trade Model shown as a state transition diagram, followed by the equivalent definition in XML:



State diagram for simple RFQ Trade Model

### XML definition of simple RFQ trade Model

```
<tradeModels>
  <tradeModel name="RFQ" initialState="Initial">
    <state name="Initial">
      <transition target="OpenSent"
        trigger="ClientOpen"
        source="client" />
    </state>

    <state name="OpenSent" timeout="10" timeoutState="Timeout">
      <transition target="Open"
        trigger="OpenAck"
        source="server" />
    </state>

    <state name="Open" timeout="60" timeoutState="Timeout">
      <transition target="OTW"
        trigger="PriceUpdate"
        source="server" />
      <transition target="Cancelled"
        trigger="ClientCancel"
        source="client" />
    </state>

    <state name="OTW" timeout="60" timeoutState="Timeout">
      <transition target="AcceptSent"
        trigger="ClientAccept"
        source="client" />
      <transition target="OTW"
        trigger="PriceUpdate"
        source="server" />
      <transition target="Open"
        trigger="OTWExpire"
        source="server" />
      <transition target="Cancelled"
        trigger="ClientCancel"
        source="client" />
    </state>

    <state name="AcceptSent" timeout="10" timeoutState="Error">
      <transition target="Accepted"
        trigger="AcceptAck"
        source="server" />
    </state>

    <state name="Accepted" />
    <state name="Cancelled" />
    <state name="Timeout" />
    <state name="Error" />

  </tradeModel>
</tradeModels>
```

## 6 Appendix A: Recommended browsers

At the time of publication, the browsers recommended for the best end-user experience of Caplin Trader applications are Mozilla Firefox version 3.6, and Internet Explorer 8.0.

For the latest information on recommended browsers, contact Caplin Systems, or if you have a version of Caplin Trader, refer to the release notes.

## 7 Appendix B: Customization status

The table in this section summarizes (at the time of publication) the extent to which each type of Caplin Trader component/feature can be customized. The goal of the Caplin Trader framework is to allow complete customization of every feature and component. Incremental releases of Caplin Trader are delivered approximately once a month and with each release more features will be fully customizable.

**Key to Customization status in the following table:**

Full	Component/feature is fully customizable using XML and/or CSS and/or JavaScript.
Partial	Component/feature is partially customizable and/or a customization technique is non-standard. You may require assistance from Caplin when customizing these features.
Caplin	Customization of the component/feature requires assistance from Caplin Systems.

Component/Feature	Customization status	Notes
Blotters	Full	Specialization of Grid.
Charts	Full	Limited functionality, but fully configurable.
Composite components	Full	Examples of composite display components in the Reference Implementation are the Product Finder, Portfolio Manager, and Market Overview.
Custom content	Partial/Caplin	The current Caplin Trader API does not completely support creating some types of more complex content – contact Caplin Systems for assistance with this.
Element Renderers	Full	Currently at version 4 level.
Form Builder	Partial	Currently a beta version with limited documentation and examples. Development of the Form Builder is ongoing and these limitations will be removed in subsequent releases of Caplin Trader.
Grids	Full	Grids currently use a mixture of version 4 and older (version 2) Element Renderers.
Layouts	Full	
Locale	Partial	The renderer framework provides some customization capability.
Preferences for users	Partial	There is an API for setting user preferences, but no standard GUI forms.
Permissions	Full	
Simple Forms	Full	This will be deprecated and replaced with the Form Builder.
Themes	Partial	Documentation on how to implement new themes is not complete at the time of publication.
Trade Tickets (FX and FI)	Partial	The Caplin Trader 2.0 framework includes a new <a href="#">Form Builder</a> <sup>[44]</sup> tool that will make it much easier to customize Trade Tickets. At the time of publication the Form Builder is a beta release.

Component/Feature	Customization status	Notes
Trade Tiles (FX)	Caplin	Limited configurability at the time of publication – assistance from Caplin Systems is needed to augment Trade Tile functionality. The new <a href="#">Form Builder</a> <sup>44</sup> tool will make it much easier to customize Trade Tickets.
Trees	Full	
Watchlists	Full	Specialization of the Grid. See the Reference Implementation.

## 8 Glossary of terms and acronyms

This section contains a glossary of terms, abbreviations, and acronyms relating to Caplin Trader.

Term	Definition
<b>Apache ANT</b>	An open source, Java-based, build tool. For more information, see <a href="http://ant.apache.org/">http://ant.apache.org/</a> .
<b>Apache Maven</b>	An open source software project management and comprehension tool, based on the concept of a project object model. For more information, see <a href="http://maven.apache.org/">http://maven.apache.org/</a> .
<b>Application server</b>	Software that serves up web pages (typically with dynamically constructed content) and web-applications, for rendering or execution by client web browsers.
<b>Blotter</b>	A <b>display component</b> that displays information about each trade.
<b>Caplin DataSource+</b>	Caplin DataSource+ is the interface between <b>Caplin Xaqua</b> and the bank's systems. It plays a key role in implementing Caplin Xaqua's anti-corruption layer; for example, its domain-specific APIs help cushion the rest of Caplin Xaqua and the client applications from changes in the bank's systems. It comprises a set of domain specific APIs representing the key Caplin Xaqua subsystems. See also <b>DataSource</b> .
<b>Caplin Liberator</b>	Caplin Liberator is a real-time financial internet hub that delivers trade messages and market data to and from subscribers over any network.
<b>Caplin StreamLink+</b>	Caplin StreamLink+ is the interface between client applications and <b>Caplin Xaqua</b> . Its APIs provide access to Caplin Xaqua's financial domain functionality, allowing a client application to interact with the bank's systems, while decoupling it from the implementation specific features of those systems. See also <b>StreamLink</b> .
<b>Caplin Trader</b>	A web application framework for constructing browser-based financial trading applications Caplin Trader was formerly called "Caplin Trader Client".
<b>Caplin Trader application</b>	A <b>Caplin Xaqua client</b> that has been built using <b>Caplin Trader</b> .
<b>Caplin Transformer</b>	Caplin Transformer is an event-driven real-time business rules engine.
<b>Caplin Xaqua</b>	A framework for building single-dealer platforms that enables banks to deliver multi-product trading direct to client desktops. Caplin Xaqua was formerly called "the Caplin Platform".
<b>Caplin Xaqua client</b>	A client desktop or web application that interfaces with <b>Caplin Xaqua</b> to deliver multi-product trading to end-users. The application can be implemented in any technology that is supported by Caplin Xaqua; for example JavaScript, Microsoft .NET, Microsoft Silverlight™, Adobe Flex™, and Java™. Also see <b>Caplin Trader application</b> .
<b>Caplin Xaqua Client APIs</b>	An alternative name for <b>Caplin StreamLink+</b> .

Term	Definition
<b>Chart</b>	A <b>display component</b> that represents data visually.
<b>Composite component</b>	A <b>display component</b> that groups together other display components.
<b>Container</b>	<p>A container is a <b>Caplin Liberator</b> data structure that holds a set of references to other data items in the Liberator. When a <b>Caplin Trader application</b> requests a container item via <b>Caplin StreamLink+</b>, it is automatically subscribed to the linked items as well.</p> <p><b>Caplin Trader</b> uses containers to manage the data displayed by certain of its components, such as <b>Grids</b>.</p> <p>The Caplin Trader application can request the Liberator to provide a windowed view of the items in the container, which can help reduce the processor load and memory usage on the client.</p>
<b>CSS</b>	<p><u>C</u>ascading <u>S</u>tyl <u>S</u>heets.</p> <p>The style sheet language used to define the look and formatting of web pages written in HTML.</p>
<b>DataSource</b>	<p>DataSource is the internal communications infrastructure used by <b>Caplin Xaqua's</b> server components such as <b>Caplin Liberator</b>, <b>Caplin Transformer</b>, and <b>DataSource adapters</b>.</p> <p>In some older documents DataSource is also used as a synonym (but non-preferred term) for <b>DataSource application</b>.</p> <p>See also <b>Caplin DataSource+</b>.</p>
<b>DataSource adapter</b>	A <b>DataSource application</b> that integrates with an external (non-Caplin) system, exchanging data and/or messages with that system.
<b>DataSource application</b>	A <b>Caplin Xaqua</b> application that uses the <b>Caplin DataSource+</b> APIs to communicate with other Caplin Xaqua applications via the <b>DataSource</b> protocol.
<b>Display component</b>	<p>A GUI component of <b>Caplin Trader</b> that can be rendered in a page on the screen.</p> <p>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 <b>Grids</b>, <b>Trade Tiles</b>, and the <b>Blotter</b>.</p>
<b>DSL</b>	See <b>Domain Specific Language</b> .
<b>Domain Specific Language</b>	<p>A programming language or specification language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique.</p> <p>Definition from Wikipedia contributors, "Domain-specific language", <i>Wikipedia, The Free Encyclopedia</i>, <a href="http://en.wikipedia.org/wiki/Domain-specific_language">http://en.wikipedia.org/wiki/Domain-specific_language</a> (accessed June 2010).</p>
<b>ExtJS</b>	<p>A JavaScript library that is used by <b>Caplin Trader</b> to render <b>Trees</b> in a layout.</p> <p>For more information, see <a href="http://www.sencha.com">http://www.sencha.com</a>.</p>
<b>JSP</b>	See <b>JavaServer Page</b> .
<b>JavaServer Page</b>	A Java technology for serving dynamically generated Web pages.



Term	Definition
<b>JUnit</b>	An open source unit testing framework for client-side (in-browser) JavaScript code. For more information, see <a href="http://www.jsunit.net/">http://www.jsunit.net/</a> .
<b>Grid</b>	A <b>display component</b> that displays data in a tabular format.
<b>GUI</b>	<u>G</u> raphical <u>U</u> ser <u>I</u> nterface
<b>LAN</b>	<u>L</u> ocal <u>A</u> rea <u>N</u> etwork
<b>Layout</b>	A grouping of display components that comprise the user's view of a Caplin Trader application. See <a href="#">Layout concepts</a> <sup>[27]</sup> .
<b>Mock4JS</b>	An open source dynamic mock library for JavaScript, used to write tests that verify interactions between JavaScript objects. For more information, see <a href="http://mock4js.sourceforge.net/">http://mock4js.sourceforge.net/</a> .
<b>Panel</b>	A rectangular area in a Caplin Trader <b>layout</b> that gives the end-user a particular data view and/or access to particular functionality. See <a href="#">Layout concepts</a> <sup>[27]</sup> .
<b>Reference Implementation</b>	A reference implementation of a <b>Caplin Trader application</b> , supplied with <b>Caplin Trader</b> .
<b>RDBMS</b>	<u>R</u> elational <u>D</u> atabase <u>M</u> anagement <u>S</u> ystem
<b>RTTP</b>	<u>R</u> eal <u>T</u> ime <u>T</u> ext <u>P</u> rotocol. Caplin's protocol for streaming real-time financial data from <b>Caplin Liberator</b> servers to client applications, and for transmitting trade messages between clients and Liberator in both directions.
<b>Selenium</b>	An open source suite of tools to automate web application testing across many platforms. For more information, see <a href="http://seleniumhq.org">http://seleniumhq.org</a> .
<b>Stack</b>	A set of <b>panels</b> stacked on top of each other, with tabs to select them. See <a href="#">Layout concepts</a> <sup>[27]</sup> .
<b>StreamLink</b>	The StreamLink libraries connect client applications to Liberator via the <b>RTTP</b> protocol. They provide an object oriented API that gives access to RTTP functionality. See also <b>Caplin StreamLink+</b> .
<b>Terrace</b>	A set of <b>panels</b> grouped horizontally. See <a href="#">Layout concepts</a> <sup>[27]</sup> .
<b>Tile</b>	A <b>display component</b> that displays information in a square or rectangular area of the page. The most typical implementation of a Tile is the <b>Trade Tile</b> .
<b>Tower</b>	A set of <b>panels</b> grouped vertically. See <a href="#">Layout concepts</a> <sup>[27]</sup> .
<b>Trade Model</b>	The formalization of the trade flow applying to a type of trade, expressed as a set of states and the transitions between them. See <a href="#">Internal modules</a> <sup>[22]</sup> and <a href="#">Defining a Trade Model</a> <sup>[47]</sup> .
<b>Trade Tile</b>	A <b>display component</b> that allows the user to trade on a product

<u>Term</u>	<u>Definition</u>
	with a single mouse click.
<b>Tree</b>	A <b>display component</b> that displays information in a tree-like structure.
<b>Web application framework</b>	<p>A web application framework is a type of framework specifically designed to help developers build web applications.</p> <p>Definition from <i>DocForge, an open wiki for programmers</i> <a href="http://docforge.com/wiki/Web_application_framework">http://docforge.com/wiki/Web_application_framework</a> (accessed Jun 2010).</p>
<b>WebDriver</b>	<p>An open source web application testing tool.</p> <p>For more information, see <a href="http://seleniumhq.org">http://seleniumhq.org</a>.</p>
<b>WAR</b>	<p><u>Web Application Archive</u></p> <p>A [Java] JAR file used to distribute a collection of JavaServer Pages, servlets, Java classes, XML files, tag libraries and static Web pages (HTML and related files) that together constitute a Web application.</p> <p>Definition from Wikipedia contributors, "WAR (Sun file format)", <i>Wikipedia, The Free Encyclopedia</i>, <a href="http://en.wikipedia.org/wiki/Web_application_archive">http://en.wikipedia.org/wiki/Web_application_archive</a> (accessed February 2010).</p>
<b>XML</b>	<u>Extensible Markup Language</u>

## 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](http://www.caplin.com)

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

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

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

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

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

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