CAPLIN

# StreamLink 5.0

## Overview

September 2010

# Contents

# 1 Preface

## 1.1 What this document contains

This document gives a technical overview of Caplin StreamLink version 5.0.

It aims to provide an understanding of:

- What StreamLink is and how it can be used.
- Fundamental StreamLink concepts & features.
- How StreamLink fits into Caplin Xaqua.
- How StreamLink can integrate with your own client application software to provide streaming data display and support online trading of financial instruments.

**Note:** At the time of writing the StreamLink SDKs released at version 5.0 are StreamLink.NET, StreamLink for Silverlight™, StreamLink for Flex™, and StreamLink for iOS.

### 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 introduction to Caplin StreamLink.

Typical readers include:

- Technical Managers
- System Architects
- System Administrators
- Operators
- Software Developers

## 1.3 Related documents

- **Caplin Xaqua Overview**

  A business and technical overview of Caplin Xaqua.
- **StreamLink.NET API Reference**

  The reference documentation for the .NET implementation of StreamLink.
- **StreamLink for Silverlight API Reference**

  The reference documentation for the Silverlight implementation of StreamLink.
- **StreamLink Configuration XML Reference**

  Describes the XML-based configuration for StreamLink 5.n.
- **Caplin DataSource Overview**

  A technical overview of Caplin DataSource.
- **Caplin Liberator Administration Guide**

  Describes Caplin Liberator and its place in the Caplin Xaqua architecture, and describes how to install and configure a Liberator.

## 1.4 Typographical conventions

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

| ***Type*** | ***Uses*** |
|---|---|
| **aMethod** | Class and method names |
| `Some code;` | Program output and code examples |
| The `value=10` attribute is... | Code fragment in line with normal text |
| **XYZ Product Overview** | Document name |
| ◆ | Information bullet point |

**Note:** Important Notes are enclosed within a box like this.

## 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.

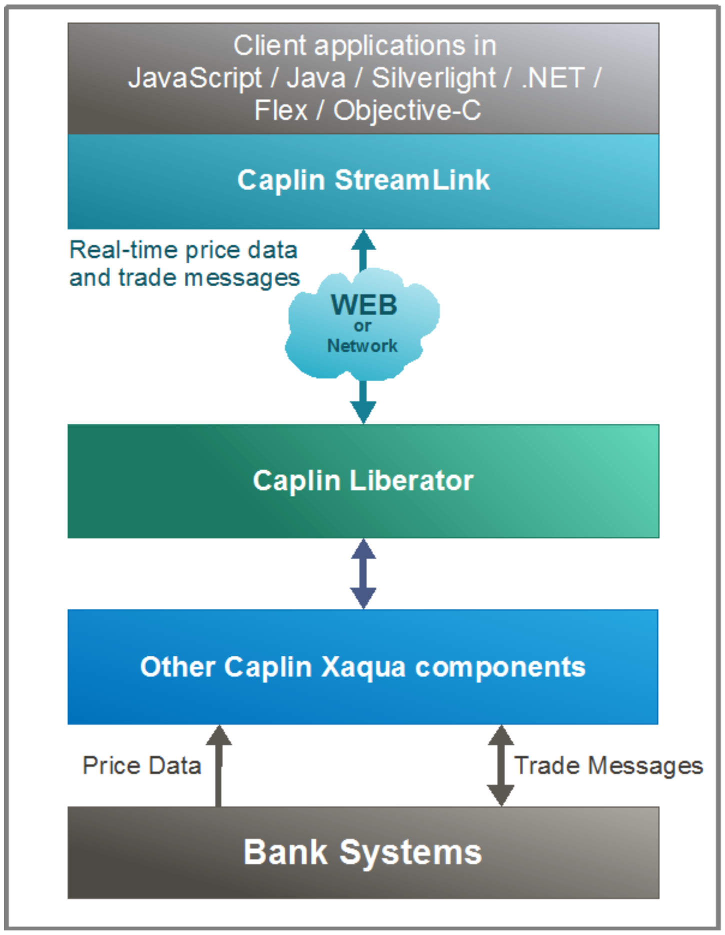
*Silverlight* is a trademark of Microsoft Corporation in the United States and other countries.

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

*Objective-C* is a trademark of Apple Inc., registered in the U.S. and other countries.

# 2 What is Caplin StreamLink?

StreamLink is an API and underlying code library that allows client applications to exchange real-time financial data and trade messages with Caplin Xaqua. This is shown in the following diagram.



**StreamLink exchanging financial data and trade messages**

The client application can be implemented in a variety of technologies. It can be:

- A browser-based (JavaScript™) application.
- A Java™ applet, Java application, or Java Web Start application.
- A Microsoft Silverlight or .NET application (typically implemented in C# or Visual Basic).
- An Adobe Flex application.
- An Objective-C application running under iOS.

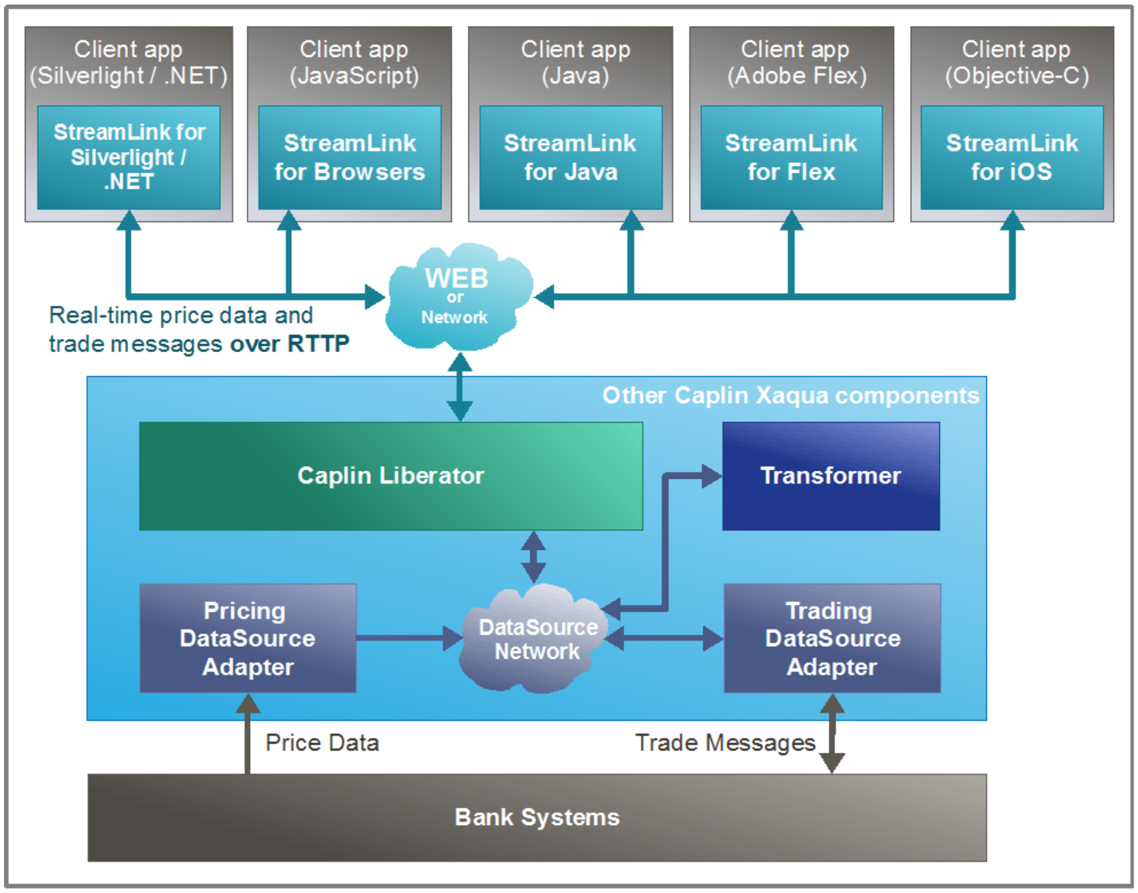
StreamLink acts as the interface between the client application and a Liberator server. Caplin Liberator is a real-time financial internet hub designed to deliver financial market data and trade messages across the Web, Intranets, or dedicated networks. Using StreamLink the client can connect to the Liberator and receive streaming price data and other market data that is updated in real time. With a suitably coded client, end users can place trades based on the displayed data. StreamLink passes the trade messages to the Liberator for onward transmission to a trading system and will receive the response messages via the Liberator, forwarding them to the client application.

# 3 Key concepts

This section introduces some key concepts about StreamLink.

## 3.1 StreamLink and Caplin Xaqua

StreamLink is a component of Caplin Xaqua that resides in client applications. The following diagram shows in a little more detail how it fits in with the rest of Caplin Xaqua.



**StreamLink within the Caplin Xaqua Architecture**

StreamLink communicates with a Liberator server using Caplin's Real Time Text Protocol, RTTP 7, typically across an Internet connection tunneled over HTTP or HTTPS. The StreamLink API conveniently hides from the client application the details of how to handle the RTTP connection and protocol.

Liberator acts as the gateway for all data flows between Caplin Xaqua and the StreamLink API in clients. The Liberator communicates with other Caplin Xaqua components using an internal protocol called DataSource. Liberator obtains data from and sends data to external systems via components called DataSource adapters (often just called "DataSources" in Caplin documents).

In a typical trading application the real-time financial data displayed by clients is obtained from one or more external data feeds via dedicated DataSource adapters and the Liberator. Messages about trades pass between clients and external trading system, via the Liberator and other dedicated DataSource adapters.

For more information about Caplin Xaqua and the components comprising it, see the **Caplin Xaqua Overview**. For more information about DataSource and DataSource adapters, see the **DataSource Overview**.

## 3.2 RTTP

RTTP stands for **R**eal **T**ime **T**ext **P**rotocol. This is Caplin's protocol for streaming real-time financial data from Liberator servers to client applications, and for transmitting trade messages between clients and Liberator in both directions.

RTTP is a robust text-based protocol that operates over Intranets, the Web, and over private networks (directly via TCP/IP). When operating over WANs, Intranets and the Web, RTTP is typically encapsulated in HTTP, or in HTTPS, to provide the highly secure messaging required by many financial applications.

RTTP has the following features:

- Automatically tunnels between clients and Liberator, through firewalls and proxy servers, with no special TCP/IP port requirements.
- Uses persistent virtual connections, providing rapid and seamless recovery from transient connection loss.
- Is designed to ensure that communication between a Liberator and its clients utilizes bandwidth extremely efficiently.
- Can handle information in numerous formats, including:
  - Structured records.
  - News headlines and news stories.
  - Containers.
  - Directories.

For more information on RTTP data formats, see Data Types 18 in About the data 17.

## RTTP connection types

RTTP works (via StreamLink) with various client application technologies: Java, JavaScript, .NET and Silverlight applications, and Objective-C running on iOS. Accordingly it has several connection types to support different combinations of application and network technology, as shown in the following table.

| RTTP Connection type | Client type | Characteristics |
|---|---|---|
| Type 1<br>(Direct) | Java,<br>.NET applications | This is a direct persistent connection via TCP/IP. The client connects to the server via a TCP/IP socket. The server streams data directly to the client across the connection. |
| Type 1 SSL<br>(Direct Encrypted) | .NET applications | This is a direct persistent connection via TCP/IP that is encrypted using the SSL protocol. The client connects to the server via a TCP/IP socket. The server streams encrypted data directly to the client across the connection.<br>Type 1 SSL connections are only supported in more recent versions of Liberator. For more information contact Caplin Support (https://support.caplin.com or support@caplin.com). |
| Type 2<br>(HTTP Tunneled) | Java, JavaScript .NET applications, Silverlight applications, Objective-C applications on iOS | This is a persistent connection that uses tunneling over HTTP or HTTPS. The client connects to the server by requesting a URL. The server holds the requested page open and streams data to the client until the client closes the connection. |
| Type 3<br>(Refresh) | Java, JavaScript, .NET applications, Silverlight applications | The client polls the server for data, issuing a fresh HTTP or HTTPS request each time.<br>Because the polling mechanism is relatively inefficient and resource hungry, and can increase message latency, it is usually only used as a fallback mode when type 2, 4, or 5 connections cannot be established. |
| Type 4<br>(Multipart Replace) | JavaScript | This is a variant of the Type 2 connection used only by JavaScript clients running in the Firefox® browser. The client makes an XmlHttpRequest with a MIME type of multipart/x-mixed-replace. The server streams data to the client across the connection made by this request.<br>Type 4 connections cannot be used from within the Internet Explorer® browser, because the browser does not support the required MIME type. |
| Type 5<br>(Streaming JavaScript) | JavaScript | This is a variant of the Type 2 connection that is used by only JavaScript clients, most often from within the Internet Explorer Browser.<br>The client obtains its updates through an IFrame. The Liberator streams data into the IFrame in the client for StreamLink to pick up. |

Type 2, 4, and 5 connections are commonly referred to as Comet connections, when used in a browser.

StreamLink can be configured to define which connection types should be used by a particular client; see Configuration 40.

## 3.3 StreamLink APIs and SDKs

StreamLink provides an object oriented API that allows applications to access RTTP functionality without needing to understand the details of the protocol itself. It is supplied as an SDK containing the API and an underlying software library. There is an SDK for each type of client technology that can access Caplin Xaqua.

Using the appropriate StreamLink SDK, you can build applications in JavaScript (Ajax), HTML, Java, .NET, Silverlight, and Objective-C on iOS. The following table shows which StreamLink SDK is used for which client technology.

| Client application technology | StreamLink SDK | SDK short name |
|---|---|---|
| *JavaScript/Ajax/HTML | **StreamLink for Browsers** | **SL4B** |
| *Java | **StreamLink for Java** | **SL4J** |
| .NET applications (typically implemented in C# or Visual Basic .NET) | **StreamLink.NET** | **SL.Net** |
| Silverlight applications (typically implemented in C# or Visual Basic .NET) | **StreamLink for Silverlight** | **SL4S** |
| Adobe Flex | **StreamLink for Flex** | **SL4F** |
| Objective-C on iOS | **StreamLink for iOS** | **SL4i** |

* At the time of writing the currently available versions of StreamLink for Browsers and StreamLink for Java (4.n) support most of the functionality defined in this overview, but do not use the StreamLink 5.0 Architecture. Versions that support the StreamLink 5.0 Architecture will be available in later StreamLink 5.0 releases.

Also see the StreamLink Architecture 27.

## 3.4 Caplin Liberator

Caplin Liberator is a real-time financial internet hub that delivers trade messages and market data to clients over any network that supports RTTP. StreamLink connects client applications to a Liberator and exchanges data with the rest of Caplin Xaqua (and with other clients) via the Liberator.

Liberator contains a high performance publishing engine capable of delivering a million updates per second from a single server to multiple clients. It also provides standard Web server functionality to clients using HTTP and HTTPS connections.

Liberator supports all the RTTP data formats – structured records, directories, and so on (see RTTP 7, and Data types 18 in About the data 17).

For more information about Liberator see the **Caplin Xaqua Overview** and the **Caplin Liberator Administration Guide**.

## 3.5 Subscriptions and real-time updating

StreamLink uses a publish and subscribe model. A Liberator server manages data which it publishes to clients. Clients *subscribe* through StreamLink to selected data. For example, a client may subscribe to a financial instrument, such as a foreign exchange currency pair – say EURUSD. The Liberator manages the subscriptions requested by all the clients connected to it.

The StreamLink API has a set of subscription classes that allow a client to name the data it wishes to subscribe to and to issue subscription requests. When a client sends a subscription request to the Liberator, if the Liberator does not have the required data item in its cache (see Caching data 11), it will in turn issue a subscription request for the item from an appropriate other DataSource (for example a DataSource adapter that supplies indicative price information).

When Liberator has the data available, it sends the client's StreamLink an initial image of the complete data item. StreamLink then makes the data available to the client application. For example, this could be a record with type 1 data 19 containing indicative bid and ask prices for EURUSD.

As the Liberator receives further updates to the data item, it updates its cache and pushes the updates to all clients that are subscribed to the item. Generally only the parts of the data item that have changed are sent out. For example if just the bid price of a currency pair changes the Liberator sends only the new bid price field to the clients that have subscribed to the currency pair.

The types of data that can be subscribed to are described in Data types 18. Subscriptions can specify parameters that restrict how much data is returned (see Making subscriptions more specific using parameters 33).

StreamLink clients can also publish data, so that it is available to the Liberator, other DataSources, and other subscribing clients (see Creating, updating, and deleting data) 10.

## 3.6 Creating, updating, and deleting data

Besides receiving data from Liberator, clients can use StreamLink to send data back to the Liberator and/or its DataSources.

**Note:** Some Caplin documents use the term “contribution” to mean updating Liberator data from a client.

A client can create new data items, update existing items, and delete existing items, using the appropriate StreamLink commands:

- **Create** – to create data items
- **Publish** – to send updates to data items
- **Delete** – to delete data items.

In practise, whether or not a client can do these things, and where resulting data changes are is visible, depends on which Caplin component *owns* the item. The client must also have write permission granted on the data items.

### Item is owned by a data service

(For an explanation of data services see the **Caplin Data Source Overview**.)

When a data service owns a data item*, clients cannot create or delete the item. A client can *update* the item, provided it has write access to it. The Liberator does not keep the update in its cache, rather it just sends it on to the DataSource(s) providing the data service. The owning DataSource may keep the updated information to itself, to forward it to an external system for example.

This type of data modification is typically used in trade messaging, where the data items are trade messages exchanged privately between a particular client and a Trading DataSource. Effectively a client sends a trade message to the DataSource by publishing an update that other clients cannot see.

Alternatively the DataSource could send the update back to the Liberator, whence the Liberator will cache the new data so that other clients (having read access permission to the data) will be able to see the new values.

* A data item is owned by a data service if its subject name matches the namespace configured for the data service, and the DataSource (or DataSources) that provide the data service are configured to accept updates.

### Item is owned by Liberator

If a data item is not owned by a data service, it is owned by the Liberator. In this case the client can create, update, and delete the item (provided it has write access to it). The changes are cached in the Liberator and are immediately available to other clients.

This type of data modification is used for client-to-client chat, for example.

## 3.7 Caching data

To ensure optimum performance and reduce network load, data is cached within the Liberator server and can also be cached in the StreamLink clients.

### Liberator cache

Liberator maintains a cache of all the items that clients have subscribed to. The cache is updated as updates are received from Liberator's DataSources or from StreamLink clients. Liberator records which clients have subscribed to which items, so as clients subsequently unsubscribe, it can delete items from its cache when they are no longer subscribed to by any client.

Liberator can also cache multiple levels of data (see Type 2 data 20), and record history for time-series replay (see Type 3 data 21).

### Client-side cache

**Note:** This feature is currently only available in StreamLink for Browsers.

StreamLink for Browsers (SL4B) maintains a client-side cache of subscribed data items received from Liberator. This cache is updated as new data images and updates to data items are received from Liberator. The cache allows the client application to display and update a data item in more than one place without imposing additional load on Liberator.

For example, an indicative price for a particular financial instrument may be displayed in a summary list and is updated in real time. SL4B maintains these price updates in its internal cache. When the end-user clicks on the instrument name, a dialog box appears displaying the details of the instrument, including its indicative price which updates in real time. When the dialog requests these details from SL4B, SL4B does not need to create another subscription on the Liberator to obtain the price; instead it just gets the price updates from its cache.

## 3.8 StreamLink in trading applications

StreamLink can be used within trading applications. As well as being the mechanism by which a client trading application obtains price information, StreamLink can provide the bi-directional transport for the message flows generated when the end-user executes trades.

For more information see Trade messages 26.

# 4 StreamLink 5.0 features

This is the list of StreamLink 5.0 features. At present the StreamLink SDKs that support the new StreamLink 5.0 API are StreamLink.NET (**SL4N**), StreamLink for Silverlight (**SL4S**), StreamLink for Flex (**SL4F**), and StreamLink for iOS (**SL4i**).

**Y** = Feature currently available.

**L** = Limited implementation of this feature

**N** = Feature not currently available

Many of the features in the list are currently available in version 4.n of the other StreamLink SDKs (StreamLink for Java, and StreamLink for Browsers). They will be brought forward to version 5.n of these SDKs. Note that the version 4.n SDKs do not have the same API architecture as StreamLink 5.0 (see StreamLink architecture 27).

**Supported data types**

| **Features** | **SL4N** | **SL4S** | **SL4F** | **SL4i** |
|---|---|---|---|---|
| ◆ Records (see About the data 17): | | | | |
| –Type 1: subject + fields. | **Y** | **Y** | **Y** | **Y** |
| –Type 2: Supports level 2 quote data. | **Y** | **Y** | **Y** | **N** |
| –Type 3: Holds history of updates. | **Y** | **Y** | **Y** | **N** |
| ◆ Containers. | **Y** | **Y** | **Y** | **Y** |
| ◆ Directories | **Y** | **Y** | **N** | **N** |
| ◆ Auto-subscription directories. | **N** | **N** | **N** | **N** |
| ◆ Pages. | **Y** | **Y** | **N** | **N** |
| ◆ News Headlines. | **Y** | **Y** | **N** | **N** |
| ◆ News Stories. | **Y** | **Y** | **N** | **N** |
| ◆ Chat. | **Y** | **Y** | **N** | **N** |
| ◆ Permissions. | **Y** | **Y** | **Y** | **N** |

**Supported connection types (to server)**

| **Features** | **SL4N** | **SL4S** | **SL4F** | **SL4i** |
|---|---|---|---|---|
| There are several connection types to support different combinations of application and network technology. See RTTP connection types 8. | | | | |
| ◆ Direct via TCP/IP. | **Y** | **N** | **Y** | **N** |
| ◆ HTTP/HTTPS tunneled. | **Y** | **Y** | **Y** | **Y** |

**Data transmission**

| Features | SL4N | SL4S | SL4F | SL4i |
|---|---|---|---|---|
| ◆ Real-time data updates. | Y | Y | Y | Y |
| ◆ Trade messaging support. | Y | Y | Y | Y |
| ◆ Binary data transmission as encoded text. | N | N | N | N |
| ◆ Batching of requests and discards – one request message can subscribe to many data items or discard many subscriptions. | Y | Y | Y | Y |

**Data manipulation**

| Features | SL4N | SL4S | SL4F | SL4i |
|---|---|---|---|---|
| ◆ Ability to specify server-side data filtering from the client (See Filtering data 34): | | | | |
| – Record image filtering. | Y | Y | Y | N |
| – Record update filtering. | Y | Y | Y | Y |
| – News filtering. | Y | Y | N | N |
| – Auto-directory filtering. | N | N | N | N |
| ◆ Client side cache<br>Allows the client application to display and update a data item in more than one place without imposing additional load on Liberator. See Caching Data 11. | N | N | N | N |
| ◆ Discarding specified fields.<br>If a data item has been requested with specific fields, you can discard some of these fields and keep receiving updates for others. | Y | Y | Y | Y |
| ◆ Helper class to decode received binary data. | N | N | N | N |

**Commands to modify data**

| Features | SL4N | SL4S | SL4F | SL4i |
|---|---|---|---|---|
| ◆ The client can modify data on the server, on other DataSources, and on other clients: | | | | |
| – Create new data items. | Y | Y | Y | N |
| – Delete existing data items. | Y | Y | Y | N |
| – Publish to existing data items (update them). | Y | Y | Y | N |

**Performance**

| Features | SL4N | SL4S | SL4F | SL4i |
|---|---|---|---|---|
| ◆ Efficient data transmission through the RTTP protocol. | Y | Y | Y | Y |
| ◆ Dynamically configurable throttling levels: global and per subject (see Throttling 38). | Y | Y | N | N |

**Resilience**

| Features | SL4N | SL4S | SL4F | SL4i |
|---|---|---|---|---|
| ◆ Rapid and seamless recovery from transient connection loss. | Y | Y | Y | Y |
| ◆ NOOP connection monitoring: RTTP detects connection loss even when TCP/IP sockets are not closed properly. | Y | Y | Y | Y |
| ◆ Failover modes of arbitrary complexity through XML configuration (see Resilience, failover, and load balancing 41) 41. | Y | Y | N | N |

**Security**

| Features | SL4N | SL4S | SL4F | SL4i |
|---|---|---|---|---|
| ◆ Single sign-on support through Caplin KeyMaster integration. | Y | Y | N | Y |

**Status reporting**

| Features | SL4N | SL4S | SL4F | SL4i |
|---|---|---|---|---|
| ◆ Liberator Data Service notifications and object status updates. | Y | Y | Y | Y |
| ◆ Access to connection statistics. | L | L | L | L |

**Configuration**

| Features | SL4N | SL4S | SL4F | SL4i |
|---|---|---|---|---|
| ◆ Configuration through XML (see Configuration 40). | Y | Y | Y | N |

**Logging**

| Features | SL4N | SL4S | SL4F | SL4i |
|---|---|---|---|---|
| ◆ Set log level and categories of information to log. | Y | Y | Y | Y |
| ◆ Set up custom logger class. | Y | Y | Y | Y |

**Instrumentation**

| Features | SL4N | SL4S | SL4F | SL4i |
|---|---|---|---|---|
| ◆ Message latency measurement. | N | N | N | N |

# 5 About the data

StreamLink and Liberator have a common view of the messages they pass between each other, based upon a set of standard data formats used by RTTP. These formats are designed to support efficient transmission of information for financial applications. The same formats are used internally to Caplin Xaqua within DataSource messages (see **DataSource Overview**).

**Note:** Within Liberator and other Caplin Xaqua components, such as DataSource adapters, the RTTP and DataSource data items are often described as “objects”.

## 5.1 Subjects, symbols, and fields

An RTTP message is identified by its **subject.** In the case of **records** the subject is usually an identifying **symbol** (a letter or sequence of letters used to identify a financial instrument). The body of a record message consists of fields.

**Simple record message:**

| Symbol | Fields |
|---|---|
| /FX/ EURUSD | price=1.334 |

In this example the record sent by Liberator to StreamLink contains price information. The symbol /FX/ EURUSD identifies the record as the price for conversion between Euros and US Dollars. The price field contains the indicative exchange rate in US dollars per Euro.

**Where did the price information in this example come from?**
The prices would typically originate from a Foreign Exchange price feed. A DataSource adapter in Caplin Xaqua converts the prices information into DataSource messages, which are passed to a Liberator. The Liberator caches the records and pushes them out to subscribing clients via RTTP and StreamLink.

**Subjects** can have an arbitrary format, but they must start with a “/” as shown in the example above, and are usually organized in a “/” separated directory structure (see Directories 24 in the Data types 18 section).

The actual symbol structure and naming conventions used depend on the type of system that Caplin Xaqua forms a part of (for example a foreign exchange trading system, or a securities market data system), and the naming conventions used by the external systems connected to the DataSource adapters. DataSources may map the symbols of records received from external systems into formats used internally to Caplin Xaqua, and for presentation to end users via StreamLink.

A **field** is a specific piece of information relating to the subject. For example, the fields for a record containing a foreign exchange quote might include “Bid” (the bid price), “Ask” (the asking price), and “ Amount” (the maximum amount of the base currency that can be traded at the quoted price).

When the Liberator sends an update message to a client, it normally* only includes in the message the fields that have changed.
For example, when a client first subscribes to Liberator to obtain a quote for /FX/EURUSD, the record returned will contain *all* the relevant fields:

/FX/EURUSD Bid=1.3442 Ask=1.3448 Amount=10000

If the Ask price subsequently changes, the Liberator will send the client an update containing just the new value of the Ask field:

/FX/EURUSD Ask=1.3450

The range of fields available for a particular financial instrument depends on the DataSource from which the data about that instrument has been obtained.

* Alternatively Liberator can be configured to always send *all* the subscribed fields of a data item to the client whenever the item is updated.

Also see Directories 24 in Data types 18.

## 5.2 Data types

StreamLink and RTTP can handle the following types of data:

- Records
- Containers
- Directories
- Auto subscription directories
- Pages
- News headlines and news stories
- Chat
- Permissions

**Note:** Other Caplin components, such as Liberator often refer to these data types as “object types”.

### Records

A record is a means of storing and displaying information. Records are composed of one or more fields which may be of different types. For example, a record containing foreign exchange data could have several price fields (such as the last bid and ask prices) together with time and date fields, whereas an index record would have a price field but no bid or ask values.

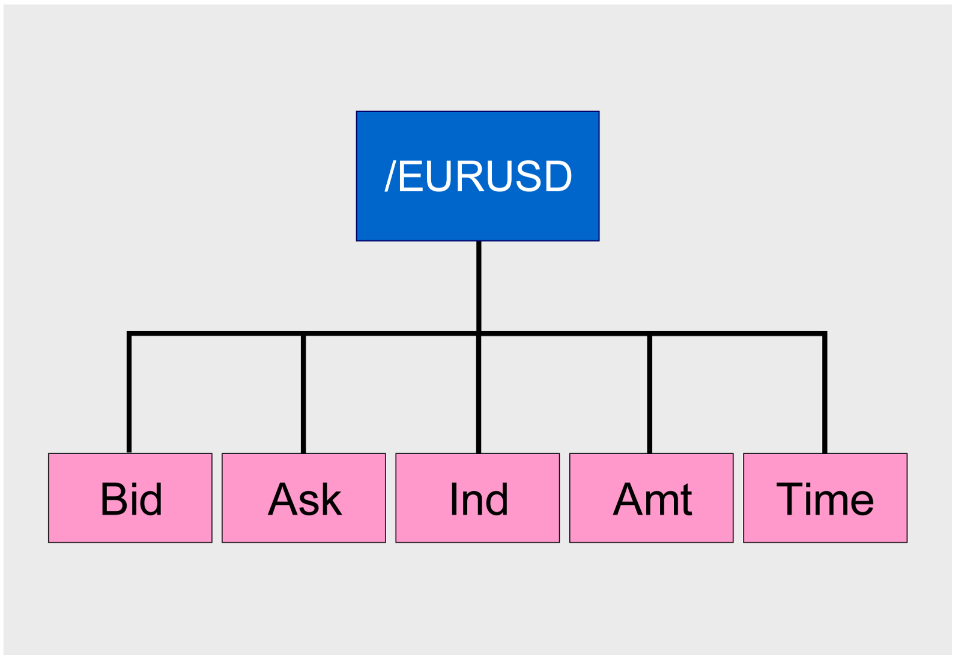
The data in a record can be structured in one of three different ways, known as Type 1 data, Type 2 data, or type 3 data.

Records can be used to hold trade messages 26 as well as financial market data.

## Type 1 data

The majority of record based data is structured as Type 1 data. This means there is only one level of fields under the symbol that identifies the record.

The following diagram shows an example field structure for a quotation in a foreign exchange trading system.



**Record containing Type 1 data**

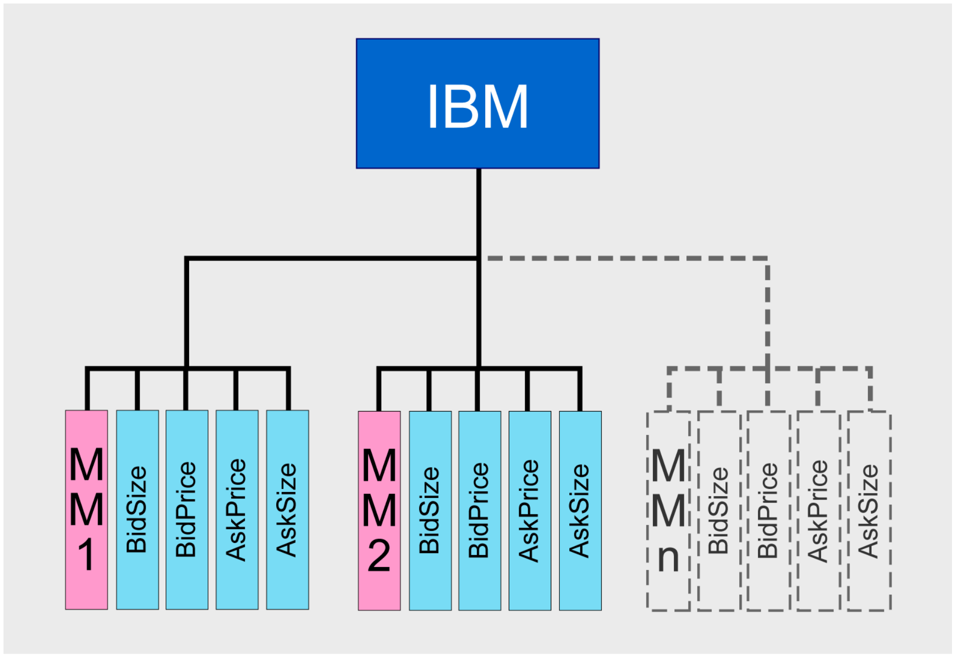
Here the single container /EURUSD contains a quote data for conversion between Euros and US Dollars. It has one level consisting of five fields: Bid (bid price), Ask (ask price), Ind (indicative price), Amt (maximum amount of dollars that can be traded at the quoted price), and Time (the date and time of the quote).

When a client first subscribes to /EURUSD, StreamLink receives a type 1 record containing all the fields with their latest values (assuming the client didn't exclude any of the fields from the subscription). Subsequently whenever StreamLink receives an update to /EURUSD, the record contains just the fields whose values have changed.

## Type 2 data

Type 2 data is often referred to as “level 2” data, as it is used for level 2 quotes. Level 2 quote data enables several price quotes per symbol (coming from different market makers or traders) to be available at all times.

The field structure shown in the following diagram might be applicable for a simple level 2 display for equity data (in this case IBM stock), where there are several active market makers.



**Record containing Type 2 data**

In this case the IBM symbol (primary key) has a secondary key of Market Maker (MM). The record contains quote data for each of the market makers providing quotes (MM1, MM2, and so on). This allows a subscriber to see the full set of quotes in the market. An update to the record will always have a market maker associated with it, so only the fields with that market maker as a secondary key will be overwritten.

A typical use for Type 2 data is to feed the display of a market order book that is in a tabular format:

**IBM:**

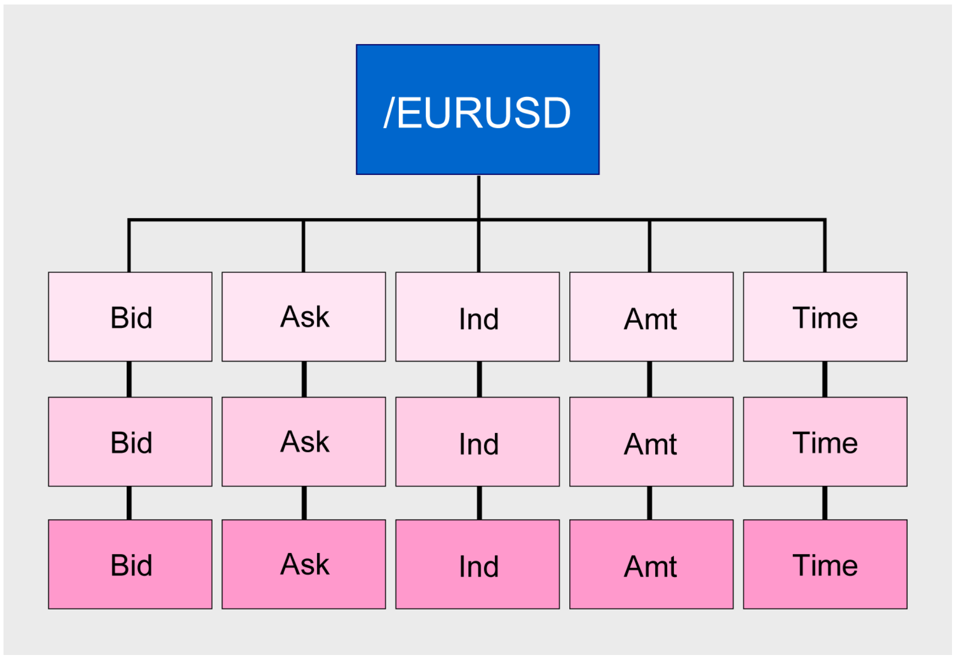
| MM (Market Maker) | BidSize | BidPrice | AskPrice | AskSize |
|---|---|---|---|---|
| MM1 | 1000 | 1.9814 | 1.9823 | 1000 |
| MM2 | 2000 | 1.9926 | 1.9999 | 2000 |
| MM3 | 1000 | 1.9602 | 1.9613 | 1000 |

A single field in the table can be easily updated from the record: for example, Key="MM1" AskSize="2500".

## Type 3 data

Type 3 data stores the history of updates to the record. A common use for data of this type is holding and viewing daily trade activity, where typically this mechanism will only be used for a day at a time before the Liberator's cache is deleted and the update list starts again.

The following diagram shows a record containing Type 3 data that records the history of quotations in a foreign exchange trading system.



**Record containing Type 3 data**

This particular record holds the history of quotes for conversion between Euros and US Dollars. The boxes running from left to right are the fields of the record; these are Bid (bid price), Ask (ask price), Ind (indicative price), Amt (maximum amount of dollars that can be traded at the quoted price), and Time (the date and time of the quote).

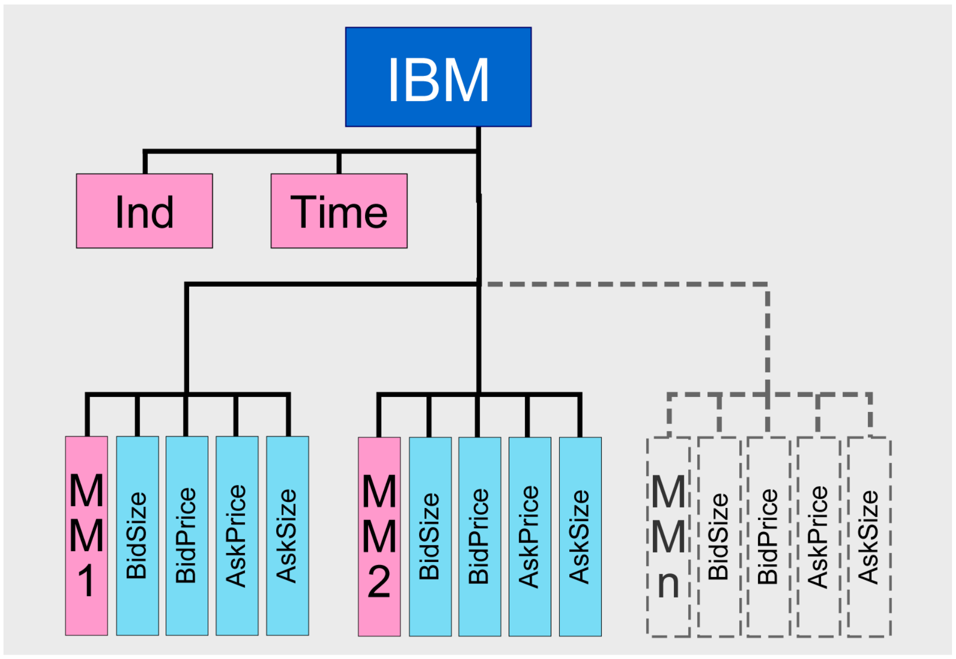
Each new record update is inserted at the end of the list. So the topmost line of fields contains the oldest known value of the record, and the bottom line of fields contains the most recent value of the record.

Caplin Liberator can maintain type 3 data structures, and it allows you to configure the amount of update history to be retained in the structure. When subscribing to a record containing type 3 data, the StreamLink client will immediately receive from Liberator all cached updates, followed by any new updates as they occur.

## Type 1 and Type 2 data combined

A single record can contain both Type 1 and Type 2 data.

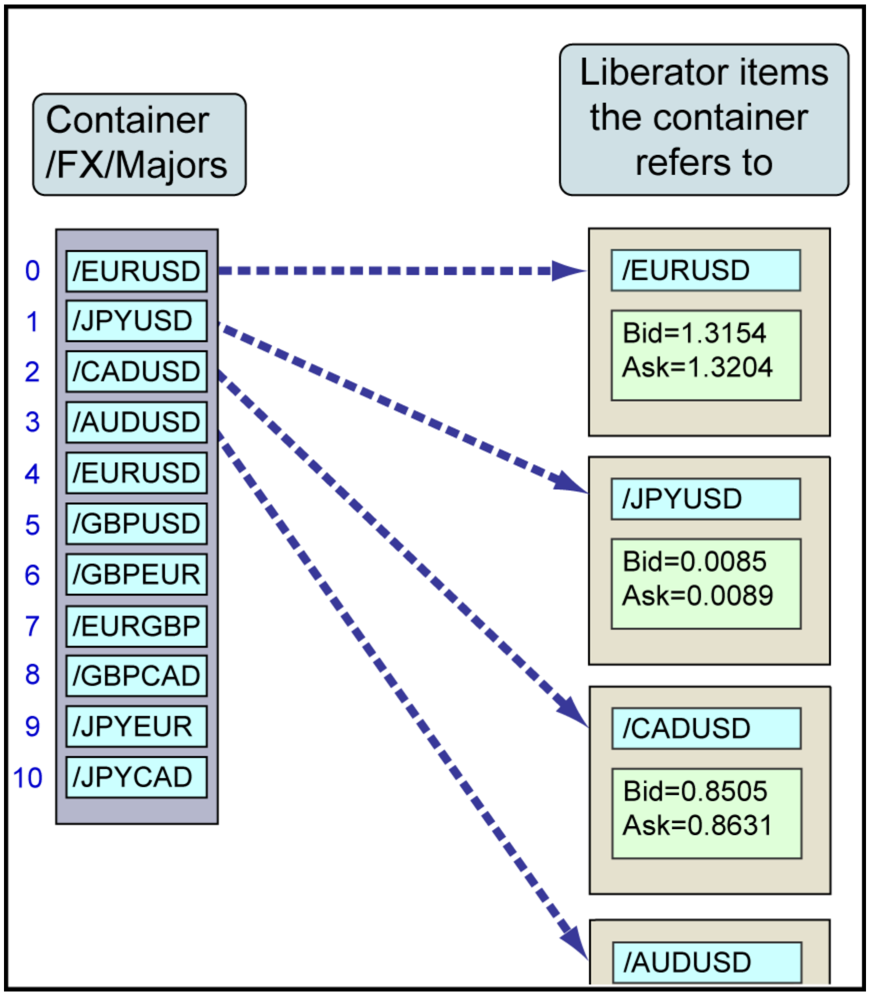
The following record is for equity data (in this case IBM stock), containing Type 1 data consisting of the indicative price (Ind) and the time of the quote (Time), followed by Type 2 data listing price quotes from several different market makers (MM1, MM2, and so on).



**Record containing Type 1 and Type 2 data**

## Containers

A container holds a set of references to other data items in the Liberator, as shown in the following diagram.



**Structure of a container**

When a client requests a container item from StreamLink, it is automatically subscribed to the linked items as well. Any parameters on the container subscription (such as fields or filters) are passed on to the linked item subscriptions.

Containers are managed on the server side (that is, by the Liberator) on behalf of all the subscribing clients, so there is less work for the clients to do. For example, the Liberator handles the addition and deletion of items in the container; it automatically adjusts client subscriptions accordingly and communicates the changes to the clients via StreamLink. In contrast, when a client subscribes to a directory (see Directories 24), it is not automatically subscribed to the items in the directory; the client must explicitly subscribe to them as required.

A client 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 – for more information see Specifying container parameters 37.

Containers are usually defined by a DataSource, although they can be provided by a different DataSource to the one that supplies the referenced data.

## Directories

StreamLink, RTTP, and Liberator understand and utilize the concept of a directory based hierarchical name space for data items. This is realized in the use of the "/" delimiter for symbols.

For example the symbol /FX/USD comprises

- the root directory "/" (the first "/" in /FX/USD)
- the directory "FX" (Foreign Exchange) underneath "/"
- the symbol "USD" (US Dollars) underneath "/FX/"

A client can subscribe to a directory and StreamLink will then inform the client when items are created or deleted within that directory. Subscribing to a directory does not automatically subscribe to the items in the directory; the client must explicitly subscribe to them as required.

## Auto subscription directories

This is not a separate data type, but rather a specialized way for clients to subscribe to a Directory. StreamLink requests the Liberator to automatically subscribe the client to all of the directory contents, as well as the directory itself, in a manner similar to the container object. Note, however, that any subdirectories within an auto subscription directory are *not* recursively auto-subscribed.

If the client specifies a filter for the directory (see Filtering data 34), the filter will also apply to all the items within the directory. This applies to both record filtering 34 and news filtering 36.

Auto subscription directories also provide the option to monitor filtering
(see Auto subscription directory filtering 37).

## Pages

A page is a free format piece of text made up of rows. This data type is normally used to display information that was originally formatted for terminals that only display text. Typical sizes are 14 rows of 64 characters ("Reuters small page") and 25 rows of 80 characters ("Reuters large page").

## News headlines and news stories

A news headline is a relatively short message containing free text, with a link to the more detailed news story behind the headline, and a date. The story link can be in any format; for example, it could be a link to an RTTP news story, or a URL pointing to an external web page. A headline can also have tags (or codes) associated with it, for use in searches.

A request for a news headline may contain a filter string using a simple logical syntax. The filter allows a client application to limit the headline updates it receives.

A news story is an arbitrary length text item, referred to by one or more news headlines.

## Chat

RTTP chat items allow users logged into a Liberator to chat in real-time. Each chat item represents a virtual chat room for 2 or more users. A client sends a message on a chat “channel” by updating (publishing to) the associated chat item.

## Permissions items

Clients can receive updates from Liberator about changes to access permissions. These updates are sent in permissions items, which have the same structure as Type 2 data 20. A client can use the updated permission information to modify the way the application behaves.

For more information see Permissions objects 39 in the Authentication and permissioning 39 section.

## 5.3 Trade messages

When Caplin Xaqua is deployed in a trading system, a client can use StreamLink to pass messages relating to trade transactions between the client and the trading system behind the Liberator. The client uses the StreamLink Publish command to send update messages to the trading system, and the trading system responds by sending updates back to the client.

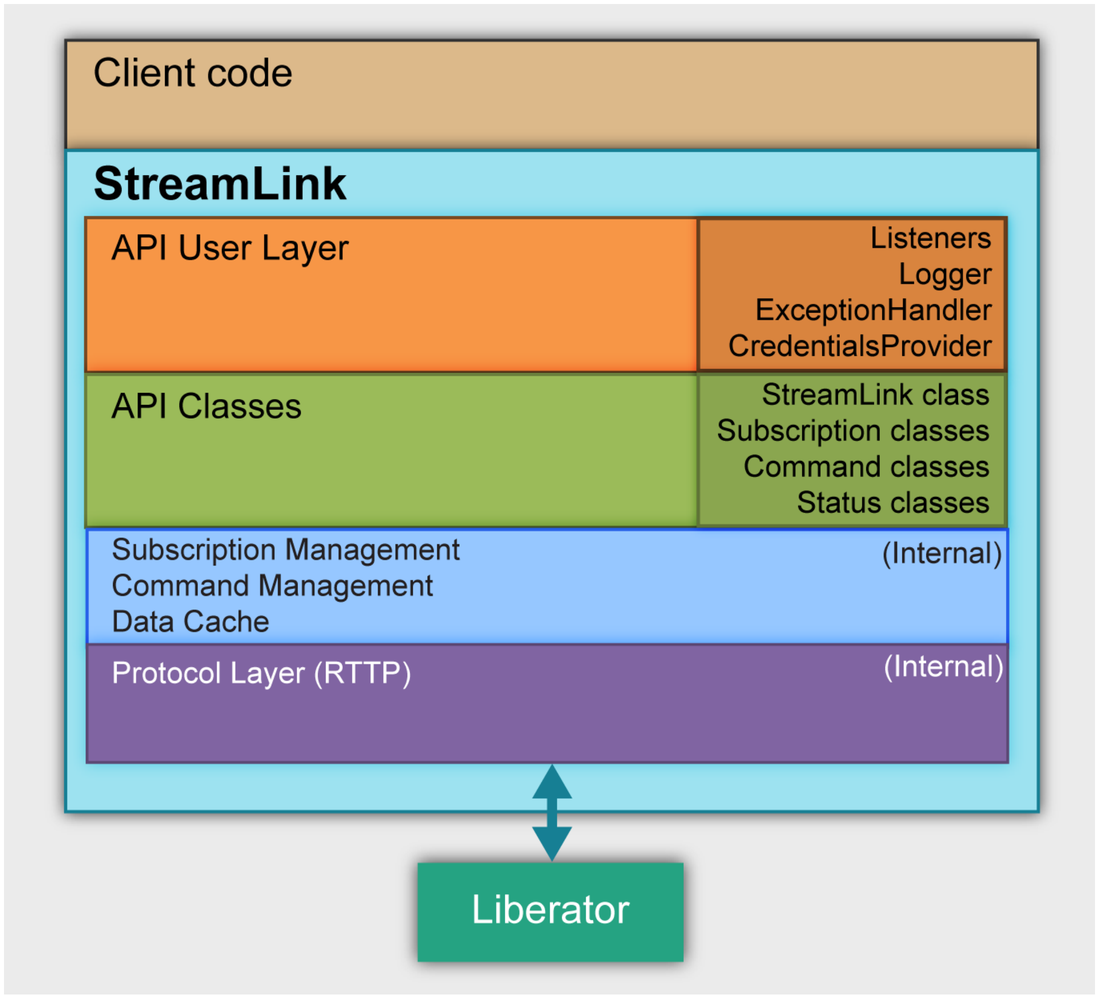
Typically the client issues an "open trade" request, which automatically subscribes the client to a subject dedicated to trade messaging (for example /TRADE/FX). The trading system sends the client (via a Trading DataSource and the Liberator) an update to /TRADE/FX containing the request acknowledgement. The client and the trading system then exchange further messages as updates to /TRADE/FX, according to the particular trade model being executed, until the trade is complete.

# 6 StreamLink architecture

The StreamLink 5.n APIs are provided as a set of classes and interfaces in the implementation language of the client application (see StreamLink APIs and SDKs 9). The StreamLink code is divided into four layers:

- The API User layer
- The API Classes
- Subscription Management, Command Management, and Data Cache
- The Protocol layer

These are shown in the following architecture diagram



**StreamLink architecture**

**Note:** The version 4.n StreamLink APIs have a different architecture.

## API User Layer

The API User Layer defines interfaces that developers using StreamLink must implement as concrete classes in the client code.

The most important of these interfaces are the Listeners which define callback methods to handle events returned by StreamLink. For example, **SubscriptionListeners** handle the data (images and updates) received from Liberator when the client has subscribed to a particular item. The actual implementations of such callbacks depend upon what the client code needs to do with the received data.

There is also a **Logger** interface, whose implementation enables the client to record StreamLink activity in an appropriate manner, and an **ExceptionHandler** interface for dealing with errors raised by StreamLink.

The **CredentialsProvider** interface is for supplying login information to the Liberator. The implementation of a **CredentialsProvider** could obtain this information from a database, a web site, or from a single sign-on system.

## API Classes

The API Classes layer contains the public classes and methods that client code uses to initiate and control interactions with StreamLink.

The main class is called **StreamLink**; every client must create an instance of this class to use the functionality provided by StreamLink. There are also **Subscription** classes for setting up subscriptions to data, and **Command** classes used to send commands to Liberator. Commands include, Create (to create data items), Publish (to send updates to data items), and Delete (to delete data items).

The **Status** classes allow a client to obtain information about the status of RTTP connections and Liberator's data services. (For an explanation of data services see the **Caplin DataSource Overview**.)

## Subscription Management and Command Management layer

This layer is internal to StreamLink. It implements the functionality of the StreamLink API; managing subscriptions, implementing the commands issued from the API User Layer, making calls on the RTTP protocol layer, and handling data and status information received from RTTP for onward transmission to the callback methods defined in the API User Layer.

## Protocol Layer

This layer is also internal to StreamLink. It manages the connections to the Liberator server, and implements the RTTP protocol that handles bi-directional communication with Liberator.

# 7 How to use StreamLink

This section contains some simple examples of how client applications can use the StreamLink 5.n API. The examples contain some simple code fragments, illustrating how the API classes and methods are used. Since the StreamLink API is written in a number of different object oriented languages (see StreamLink APIs and SDKs), the code fragments are in pseudo object oriented code – *the actual implementation details will vary from one StreamLink SDK to another*.

The rest of the section contains more detailed information about using StreamLink's features.

## 7.1 Asynchronous operation

The StreamLink API operates asynchronously. This means that when the client code issues a subscription request or a command to StreamLink, the response is not returned immediately through the method used to issue the request or command. Instead, you must set up a listener object containing one or more callback methods, and then supply this listener to StreamLink.

StreamLink calls the appropriate callback method(s) on the listener object, to communicate data and command responses back to the client code.

This method of operation is shown in the following examples.

## 7.2 Subscribing to data and receiving updates

To subscribe to data and receive updates to the data you need to

- implement (code) some interfaces from the API User Layer,
- call various methods from the API classes layer to connect to a Liberator and set up the subscription.

### Implementing the interfaces

Before using StreamLink you need to implement as concrete classes an appropriate set of listener and other interfaces defined in the API User Layer. These implementations effectively integrate StreamLink with your client application.

As a minimum, implement a **SubscriptionListener** and a **CredentialsProvider**.

StreamLink calls the **SubscriptionListener** to handle subscription events. For example, to deal with events concerning subscriptions to record-structured data, implement a **RecordSubscriptionListener**. This class must contain implementations of the methods **recordUpdated()**, **recordType2Updated()**, **recordType3Updated()**, to handle updates to type1, type2, and type 3 records respectively in a manner suitable for the client application. At run time, StreamLink passes in to each method both the subscription to which the event relates, and the updated fields as name-value pairs.

The **SubscriptionListener** also must implement a **subscriptionErrorReceived()** method to deal with errors in subscriptions, and a **subscriptionStatusUpdated()** method to handle changes to the status of the subscription. (A status message relates to the state of the Liberator data services that handle the subscription, such as “Stale”, “Limited”, “OK”. For more information about data services see the **Caplin DataSource Overview**.)

The **CredentialsProvider** implementation supplies the user credentials information used to log in to a Liberator. In the example shown in “Set up a CredentialsProvider...” below, it merely provides a user name and password that are set in the constructor. In a real implementation the credentials provider would usually be rather more sophisticated. For example, it could obtain login credentials from a database, a web site, or a single sign-on system. The credentials could take the form of a digitally signed secure token.

In a fully functioned implementation you would probably also want to implement an **ExceptionHandler** to deal with errors raised by StreamLink, and a **Logger** to record StreamLink activity.

## Calling StreamLink

The following simple example shows how to use the StreamLink API to subscribe to a simple record containing a share price. The subject is /MSFT. This pattern is used for subscribing to all types of data.

- The application will issue a subscription request to StreamLink, so the first thing to do is implement a **RecordSubscriptionListener** that receives the updates resulting from the subscription. In the pseudo code fragments below the record subscription listener implementation is called **MyRecordSubscriptionListener**.
- Create an instance of the StreamLink class:

```
StreamLink streamLink = new StreamLink();
```

This parameterless constructor call picks up configuration information from a default configuration file (this constructor is only available in StreamLink .NET). There are variants of the constructor that enable you to obtain the configuration from a specified file, or via a class that implements the **IStreamLinkConfiguration** interface. The configuration data includes a specification of which Liberator StreamLink should use and what type of RTTP connection it should establish (see RTTP connection types 8). For more information see Configuration 40.

**Creating a StreamLink instance with configuration from a file:**

```
StreamLink streamLink =
                     new StreamLink(new Uri("conf/myConfigurationFile.xml"));
```

- Set up a **PasswordCredentialsProvider** with the login information that Liberator needs:

```
ICredentialsProvider credentialsProvider =
                     new PasswordCredentialsProvider("admin", "admin");
streamLink.setCredentialsProvider(credentialsProvider);
```

- Set up the record subscription listener:

```
IRecordSubscriptionListener mySubsListener =
                            new MyRecordSubscriptionListener();
```

- Get the **StreamLinkProvider**. The **StreamLinkProvider** object provides access to the majority of the StreamLink functionality; in this example it creates the subscription.

```
IStreamLinkProvider streamLinkProvider = streamLink.getStreamLinkProvider();
```

- Create a subscription to /DEMO/MSFT via the **StreamLinkProvider**, supplying the subscription listener that will receive updates, events, and errors relating to the subscribed data item:

```
IRecordSubscription recordSubscription =
   streamLinkProvider.createRecordSubscription(mySubsListener, "/MSFT");
```

- Connect to the Liberator:

```
streamLinkProvider.connect();
```

  StreamLink connects to the Liberator defined in its configuration and logs in to the Liberator using the details previously set up in **credentialsProvider**.

- Now issue the subscription request.

```
recordSubscription.subscribe();
```

- Some time later Liberator returns an image of the /MSFT record through a call to the **recordUpdated()** method of **mySubsListener** (instantiation of **MyRecordSubscriptionListener**). Subsequently, each time StreamLink receives an update to /MSFT, it calls **recordUpdated()**, passing the updated fields and their new values.

  You only need to connect to the Liberator once; subsequent subscription requests are passed to the Liberator immediately. You can make subscription requests *before* connecting to the Liberator; StreamLink queues them until a connection is established.

## 7.3 Subscribing to more than one data item

The StreamLink API supports subscriptions to multiple data items using a single connection to the server. So there is no need to create multiple instances of the **StreamLink** class in order to make multiple requests.

For example, you can subscribe to multiple records and receive updates into a single instance of **RecordSubscriptionListener**. However, StreamLink also allows you to register a different listener for each subscribed item.

- The following example shows how to subscribe to both /MSFT and /YHOO and receive updates for both of these subjects into the same **RecordSubscriptionListener**.

```
IRecordSubscription msftRecordSubscription =
   streamLinkProvider.createRecordSubscription(mySubsListener, "/MSFT");
IRecordSubscription yhooRecordSubscription =
   streamLinkProvider.createRecordSubscription(mySubsListener, "/YHOO");
```

**Note:** StreamLink automatically batches multiple subscription requests into a single request call to maximize efficiency “over the wire”.

## 7.4 Creating a data item on Liberator

This example shows how a client application can use StreamLink to modify data on a Liberator server. In this case the application creates a new record data item. This pattern is used for issuing all StreamLink commands.

- The application will need to issue a *command* to StreamLink, so the first thing to do is implement a **CommandListener** to receive the results of the command. In the pseudo code fragments below the command listener implementation is called **MyCommandListener**.
- Create an instance of the **StreamLink** class and set up a **CredentialsProvider** with the login information that Liberator needs (see Subscribing to data and receiving updates 29):

```
StreamLink streamLink = new StreamLink(
                            new Uri(“conf/myConfigurationFile.xml”));

ICredentialsProvider credentialsProvider =
                     new PasswordCredentialsProvider("admin", "admin");

streamLink.setCredentialsProvider(credentialsProvider);
```

- Get the **StreamLinkProvider**. The **StreamLinkProvider** object provides access to the majority of the StreamLink functionality; in this example it provides access to the **ParametersFactory**, described below.

```
IStreamLinkProvider streamLinkProvider = streamLink.getStreamLinkProvider();
```

- Get the **ParametersFactory** used to create parameters for the command:

```
IParametersFactory parametersFactory =
                                    streamLinkProvider.getParametersFactory();
```

- Set up the parameters needed to create the subject. These are predefined for each type of data item and are identified by an “enum”. In this case the enum is `SubjectType.Record`.

```
ICreateParameters subjectCreationParameters =
   parametersFactory.CreateCreateParameters(SubjectType.Record);
```

- Set up the command listener:

```
ICommandListener myCommandListener = new MyCommandListener();
```

- Issue a “Create Subject” command to the **StreamLinkProvider** to create the subject. Assuming the application is already connected to the Liberator, StreamLink will send the command immediately.

```
streamLinkProvider.CreateSubject(myCommandListener,
                                 "/MyRecord",
                                 subjectCreationParameters);
```

When the command has executed successfully, StreamLink calls the **CommandSuccess()** method of **myCommandListener** (instantiation of **MyCommandListener**). If the command fails, for example because the user does not have permission to create record subjects on Liberator, StreamLink calls **CommandError()** instead.

## 7.5 Updating data

A client can update data by executing the StreamLink Publish command. Where the resulting data changes are visible, depends on which Caplin component owns the item. See Creating, updating, and deleting data 10.

## 7.6 Discarding data

When a client no longer wishes to receive updates to a subscribed data item it can *unsubscribe* from the item. This effectively discards it as far as the client is concerned. However, the item is still present on the Liberator, so the client can subsequently subscribe to it again if required.

A client can also permanently delete an item from Liberator by executing the StreamLink Delete command. For this to succeed the client must have write access permission to that item on the Liberator .

## 7.7 Making subscriptions more specific using parameters

The client can supply various types of parameter to a subscription request, to ensure that only specifically required data is returned. For example, you can specify which fields of a record are to be returned (see Specifying fields 33), you can further restrict the date using filters (see Filtering data 34), and you can control which data is returned within a container (see Specifying container parameters 37).

### Specifying fields

When the client subscribes to a record it can specify which fields of the record are to be returned, so the fields that are irrelevant to the application are ignored (Liberator does not send them across to the client).

## Filtering data

When a client subscribes to data it can specify a filter that restricts the information returned, so the client only receives values that it is interested in. The filtering is done on the Liberator, rather than in the client, so that network traffic and client side processing are reduced.

The filter is defined as an expression based on the fields in the data item.

For example:

- An FX trading client application subscribes to the currency pair /EURUSD, requesting updates for the Bid field. The subscription request includes a filter "Bid > 1.3440", meaning that the client (and hence the end-user) is only interested in receiving updates for /EURUSD where the Bid price is greater than 1.3440
- A subscription to news headlines includes a filter that selects only headlines containing the word "industrials".

## Record filtering

Data structured as records (see Records 18) can be filtered using filter expressions containing the following operators:

| Character | Meaning |
|---|---|
| \| | or |
| & | and |
| = | equal to |
| != | not equal to |
| < | less than |
| > | greater than |
| <= | less than or equal to |
| >= | greater than or equal to |
| ~ | Provides matching for items in comma-separated string lists. For example, `~abc` matches a field in `"xyz,abc,def"` but not in `"xyzabcdef"` |
| ( ) | Parenthesis – perform these filters first |

For example, the filter expression

```
(BidSize > 1000) & (AskSize > 1000)
```

selects only those records where both the `BidSize` and `AskSize` fields each contain a value greater than 1,000.

There are two sorts of record filter: update filters and record filters.

An **update filter** is applied to the data *within* a particular update of the data item. An update record does not necessarily contain all the fields that the client originally subscribed to within the item. If the filter is dependent on a field that is not present within an update, the filter operation will result in “no match”, and the Liberator will not send the update to the client.

An **image filter** is applied *after* the data within a particular update has been applied to the image of the data item. The filter may specify a field that is not present within the update. When this happens, the Liberator uses its cached value of the field to determine whether the update matches the filter.

The following table shows an example of the difference in behavior between update filters and image filters. In both cases the filter used is
`(BidSize > 1000) & (AskSize > 1000).`

Each row of the table represents the result of an update, where the updates have been applied to the same record item in the order 1, 2, 3, 4, 5.
{none} means the field in question was not updated.

**Updated record fields:**

| Update no | Bid | BidSize | Ask | AskSize | Matches Update Filter? | Matches Image Filter? |
|---|---|---|---|---|---|---|
| 1. | 54.25 | 1500 | 55.00 | 2000 | Yes | Yes |
| 2. | 54.00 | 2000 | {none} | {none} | No | Yes |
| 3. | 54.50 | 500 | {none} | {none} | No | No |
| 4. | 54.25 | 1500 | 54.75 | 500 | No | No |
| 5. | 54.00 | 2000 | 54.75 | 1500 | Yes | Yes |

## News filtering

News headlines can be filtered by specifying a news filter when creating an object. The filter expression consists of news codes and/or whole words to search for. A news filter expression can contain the following operators:

| Character | Meaning |
|---|---|
| `[space]` | or |
| `\|` | or |
| `+` | and |
| `&` | and |
| `-` | and |
| `=` | equal to |
| `!` | not |
| `~` | not |
| `'` | start or end of free text search string |
| `"` | start or end of free text search string |
| `( )` | parenthesis – perform these filters first |

For example, the filter expression

```
UKX | ('stock rises' & (!'preference'))
```

selects only those news headlines with code `UKX`, or where the headline contains the phrase “stock rises” and does not contain the phrase “preference”.

News codes can only contain capitals and numbers, and have a maximum length defined in the Liberator configuration. Capitalized items in the headline that are longer than the maximum news code length are treated as text for the purposes of searching, as are items in mixed case and lower case. Use `'` or `"` to force a text search. For example `'UKX'` selects news headlines containing the text “UKX”.

## Auto subscription directory filtering

Filters can be applied to auto subscription directories. For large directories this allows you to reduce the amount of data displayed and updated within the directory on the client.

Liberator applies the filter to all the items within the directory. If an item does not match the filter, it is unsubscribed from the client (though it remains subscribed on Liberator); conversely if an item that did not match the filter on a previous update now matches it, the item is added back to the auto-subscription directory on the client. This behavior means that a directory with many entries but a very restrictive filter will only have a small number of objects auto subscribed on the client, and the number of updates sent to the client will be correspondingly reduced.

When the filter does not apply to a particular item in the directory (for example a record filter does not apply to a news item) the behavior is undefined. However, in most situations a directory should only contain one type of data, ensuring this situation does not arise.

Clients can also monitor auto subscription directory filtering. As the data in an item changes it may match the filter when it previously did not, or cease to match the filter when it previously did. When monitoring is enabled, if an item ceases to match the filter because of an update, Liberator sends the client an item “deleted” notification. Conversely, Liberator sends the client a notification if the item is updated so that it matches the filter when previously it didn't.

## Specifying container parameters

When a client subscribes to a container it can specify a window into the container, that is, the start and end rows of the container data that are to be returned. The Liberator only sends the client updates for the items currently in the window.

The client can ask StreamLink to move the window on the container (for example because the end-user has scrolled down through the displayed view of the container window). StreamLink will then ask Liberator to supply the data for the items that are currently in the scope of the window.

This server-side paging capability helps reduce the processor and memory requirements on the client in situations where the container refers to a large number of items, but the end-user views just a few of them at a time through a small scrollable window. Rather than handling updates for the entire list, even though only a small part of it is on view, the client merely has to manage updates for the few items in the container window (which maps onto the display window).

## Specifying page rows

When the client subscribes to a page type data item it can specify a subset of the rows on the page, for example rows 10 to 16 inclusive.

## 7.8 Monitoring the connection

StreamLink can notify the client application of life cycle events relating to the connection between the client and the Liberator. This is done through an implementation of the **IConnectionListener** interface.

## 7.9 Throttling

Using StreamLink you can limit the rate at which updates are sent to a client; this is called **throttling**. There are a number of reasons for doing this:

- To reduce network usage levels, both on leaving the Liberator and entering the client.
- To reduce the load on the client, for instance when too high a screen update rate would overload the client machine.
- To reduce the load on the Liberator, by reducing the rate at which it needs to send updates to its clients.

The amount of throttling is defined as a time interval. For example, if the throttle time for a data item is 1 second then Liberator will send an update for that item to subscribed clients at most every second. So if there are three updates to the item within a second, only the third one will be sent to the clients, at the end of the one second interval.

Decreasing the throttle time increases the maximum frequency of updates received by the client, whereas increasing the throttle time decreases the maximum frequency of received updates.

When Liberator identifies that a data item is updating less frequently than the throttle time, it does not activate throttling and sends the updates to the subscribed clients immediately.

Throttle times are set up in the Liberator's configuration; these are global settings which apply to all data items unless overridden for specific items. The configuration is defined as a set of throttle levels. For example, an item could have five throttle levels:

1. No throttling.
2. Throttling at 0.5 seconds.
3. Throttling at 1 second.
4. Throttling at 2 seconds.
5. The stopped state (no updates are forwarded to clients).

The Liberator can have a default throttle level applying to all data items when the client initially subscribes. This is typically the lowest level, but it could be set to one of the other levels. A client will start at the default throttling level when the user logs in and the client subscribes to data via StreamLink.

The client can then change the throttle level by sending a command to StreamLink. A client cannot set a custom throttle time, as the times are configured in the Liberator, but the client can move up or down a throttle level, go to the minimum or maximum level, or stop receiving updates altogether and subsequently resume them.

A client can also alter the throttle level for a particular data item it has subscribed to, so the level is different to that applying to other items.

**Note:** Liberator's throttling behavior is also known as “conflation” because several update values spread across time are effectively combined into one value (the latest one).
StreamLink also “conflates” messages it sends to Liberator, but this is a different sort of optimization, where multiple requests/discards are batched up into one message for transmission.

## 7.10 Authentication and permissioning

StreamLink provides a public interface called **ICredentialsProvider** that is responsible for providing a user name and password to be used when logging in to Liberator. StreamLink developers can implement this interface themselves to perform custom logic, such as integrating with a single sign on system, or retrieving a password from an external system.

The StreamLink library includes several sample implementations of **ICredentialsProvider** depending on the technology in use:

- **PasswordCredentialsProvider**

  A basic implementation that provides user credentials consisting of a set username and password.

- **StandardKeyMasterCredentialsProvider**

  An implementation that attempts to retrieve a password token for a specified user name from Caplin KeyMaster. It can also optionally poll a specified URI on the KeyMaster server in order to keep the HTTP session alive.

- **AuthenticatingKeyMasterCredentialsProvider**

  An implementation that attempts to retrieve a password token from Caplin KeyMaster server that requires basic HTTP authentication. This implementation has the same behavior as **StandardKeyMasterCredentialsProvider** except that it attaches the user credentials to each web request that it makes.

**Note:** **StreamLink for Silverlight**
**AuthenticatingKeyMasterCredentialsProvider** is not available in StreamLink for Silverlight, because (at the time of publication) Silverlight does not allow web requests to include user credentials.

### Permissions objects

A permissions object allows changes in user permissions on objects to be sent in real time between DataSources, and between Liberator and client applications.

In StreamLink you can create a **PermissionSubscription** to subscribe to a permission object, in much the same way as you create a **RecordSubscription** to subscribe to a record (see Subscribing to data and receiving updates 29). The client application will then receive the permission object, and any subsequent updates to it, and can use this information as appropriate. For example, the client could modify the way the application behaves according to changes in the permission object, such as enabling or denying trading on particular instruments.

For more information about permissions objects, see the **Caplin DataSource Overview**.

**Note:** **StreamLink for iOS and StreamLink for Flex**
At the time of publication, StreamLink for Flex (SL4F) and StreamLink for iOS (SL4i) do not support the use of permissions objects.

## 7.11 Error handling and logging

### The ExceptionHandler

StreamLink attempts to deal with errors in its internal operation without involving the client code. However, if it encounters an error situation that it cannot resolve internally, it will throw the exception to an **ExceptionHandler** defined in the API User Layer. When such an exception occurs it indicates that the internal state of the StreamLink API can no longer be relied upon as being correct.

The **ExceptionHandler** is supplied as an interface for you to implement in a manner appropriate to your application. The recommended action is to inform the end-user and close down the application gracefully.

### The logger classes

StreamLink includes a console logger class to aid application development. You can programmatically set the logging level and categories of items to be logged. At run time StreamLink will then display on the screen diagnostic information about its operation. Developers can implement custom logger classes, by implementing the **ILogger** interface.

There is also a file logger that captures the diagnostic information in a file. This is primarily intended to help Caplin Support staff diagnose and fix StreamLink related problems encountered by customers.

## 7.12 Configuration

StreamLink 5.n is configured using definitions in XML format.

The XML allows you to define:

- A set of allowed RTTP connection types (see RTTP connection types 8)

  including the TCP/IP port number for the connection to Liberator, and for Internet connections, whether HTTP or HTTPS.

- The Liberator servers to which StreamLink can connect

  organized into groups for the purposes of failover and load balancing.

For typical examples of the XML, see the section Resilience, failover, and load balancing 41.

The XML configuration is defined in detail in **StreamLink XML Configuration Reference**.

You can also configure StreamLink programmatically by implementing the **IStreamLinkConfiguration** interface. For example StreamLink.NET and StreamLink for Silverlight come with a convenience implementation of **IStreamLinkConfiguration** called **RttpConfiguration**, and a helper class, **SimpleRttpConnectionConfiguration**. These classes allow you to write simple code to set the hostname and port for connection to a single Liberator:

- Define the required host name and port for a type 2 HTTP connection:

```
SimpleRttpConnectionConfiguration config =
   SimpleRttpConnectionConfiguration.CreateType2HttpConnection(
                                                                  "localhost",
                                                                   1234);
```

- Set the **RttpConfiguration**:

```
RttpConfiguration rttpConfiguration = new RttpConfiguration();
rttpConfiguration.setServiceConfiguration(config);
```

- Now create a StreamLink instance that connects to the Liberator using the defined configuration:

```
streamLink = new StreamLink(rttpConfiguration);
```

**Note:** **StreamLink for iOS**
At the time of publication, StreamLink for iOS (SL4i) does not provide any configuration capabilities, other than the ability to specify in application code the URI and password for connecting to a Liberator. The RTTP connection is always type 2 (HTTP Tunneled).

## 7.13 Resilience, failover, and load balancing

StreamLink supports highly resilient operation by providing the ability to connect to alternative Liberator servers (failover) and by trying alternative types of RTTP connection. These capabilities are defined through XML-based configuration (see Configuration 40).

**Note:** **StreamLink for Flex and StreamLink for iOS**
At the time of publication, StreamLink for Flex and StreamLink for iOS (SL4i) do not provide any failover or load balancing capabilities.

### Configurable failover strategy

When a client's connection to a Liberator server fails (either because there is a persistent network failure or because the Liberator has failed), the client can connect to an alternative server according to a configurable failover scheme. StreamLink also uses this scheme when it first connects the client to a server.

The configuration technique is flexible and allows you to define sophisticated failover schemes. Liberator servers can be collected into server groups, and groups can be nested within outer groups. A group can be an 'ordered' group or a 'balance' group.

**'ordered' group**

On first connection or during failover, StreamLink tries each of the servers or groups in an 'ordered' group in turn, in the order they have been declared within the configuration.

**'balance' group**

The servers or groups in a 'balance' group are tried at random.

**Example:**

```
<ServerGroup Type="balance">

   <!-- 1st Group -->
   <ServerGroup Type="order">
      <Server Name="P1" .../>
      <Server Name="B1" .../>
   </ServerGroup>

   <!-- 2nd Group -->
   <ServerGroup Type="order">
      <Server Name="P2" .../>
      <Server Name="B2" .../>
   </ServerGroup

</ServerGroup>
```

This configuration defines two 'order' groups (“1st group” and “2nd group”) within an outer 'balance' group. This means that when StreamLink first tries to connect to a server, it randomly chooses between '1st group' and '2nd group'.

Each 'order' group defines a primary server (`"Pn"`)and a backup server (`"Bn"`). Within the selected group StreamLink will first attempt to connect to the primary server; if this fails it will attempt to connect to the backup server. If neither server can be reached, StreamLink will attempt to connect to the servers in the other group.

So the sequence of servers to be tried is either

P1 > B1 > P2 > B2

or

P2 > B2 > P1 > B1

In a failover scenario, StreamLink would first attempt to reconnect to the next server in the current 'order' group, followed by the servers in the other 'order' group. The failover sequence is repeated a configurable number of times.

## Load balancing

'Balance' groups in the configuration allow you to implement server load balancing. If there are several servers (or server groups) within a 'balance' group, each client will connect at random to one of the servers, so when there are many connected clients the connections are distributed fairly evenly across the available servers.

A configuration that implemented server load balancing but no failover capability would look like this:

**Example: Server load balancing only**

```
<!-- Load balance across four servers -->

<ServerGroup Type="balance">
   <Server Name="S1" .../>
   <Server Name="S2" .../>
   <Server Name="S3" .../>
   <Server Name="S4" .../>
</ServerGroup>
```

## Alternative RTTP connection types

When a client attempts to connect to a Liberator, StreamLink refers to an ordered list of RTTP connection types[8]. It tries each type of connection in sequence until one succeeds. If none of them succeed, StreamLink will try to connect to another Liberator, according to the configured failover scheme.

For example, the RTTP connection list could be configured as

1. Type 1 on port 15000
2. Type 2 over HTTP on port 8080
3. Type 3 over HTTPS on port 8081

– Reconnect interval: 3 sec

StreamLink will first attempt to connect to the Liberator using a type 1 RTTP connection (direct TCP/IP) on port 15000. If the connection is not established within the reconnect interval of 3 seconds, StreamLink then tries a type 2 connection (tunneled) over HTTP. Again, if the connection is not established within 3 seconds, StreamLink tries a type 3 connection (refresh) over HTTPS.

# 8 Glossary of terms and acronyms

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

| Term | Definition |
|---|---|
| **Adobe Flex** | A software development kit and integrated development environment from Adobe Systems Inc. for creating Rich Internet Applications based on the Adobe Flash platform. Flex uses ActionScript code and XML-based user interface descriptions (MXML) and compiles them into binary Flash files (SWF files). |
| **Ajax** | Asynchronous Java and XML<br>A combination of Web technologies used to implement interactive Web clients. |
| **API** | Application Programming Interface |
| **Caplin Liberator** | Caplin Liberator is a real-time financial internet hub designed to deliver financial market data and trade messages across the Web, Intranets, or dedicated networks.<br>StreamLink communicates with Liberator servers. |
| **DataSource** | Synonym for **DataSource adapter** |
| **DataSource adapter** | A **DataSource peer** that obtains data from, and/or sends data to, an external (non-Caplin) system. |
| **DataSource peer** | An application that can communicate with another application using the **DataSource protocol**. |
| **DataSource protocol** | The Caplin protocol that transmits data between components of Caplin Xaqua, such as **Caplin Liberator**, **Caplin Transformer**, **DataSource adapters**, and other **DataSource peers**. |
| **Flex** | See **Adobe Flex**. |
| **IFrame** | Inline Frame<br>An HTML tag that allows another HTML document to be inserted within a floating frame in an HTML page.<br>Within **Ajax** frameworks an IFrame object can be used to exchange data with the web server (instead of using the XMLHttpRequest object). Caplin uses this technique in RTTP Type 5 connections, within the client. |
| **iOS** | An operating system for mobile devices, produced by Apple Inc. |
| **Failover** | The transfer of operation from a hardware or software component that has failed to an alternative copy of the component, to ensure uninterrupted provision of service. |
| **.NET** | A Microsoft framework for developing distributed applications that run under Microsoft Windows® operating systems and can easily intercommunicate with applications and systems running on different operating systems. |
| **.NET application** | A client application that is integrated with the Microsoft .NET framework. |

| Term | Definition |
|---|---|
| **Objective-C** | A reflective, object-oriented programming language which adds Smalltalk-style messaging to the C programming language.<br>Definition from Wikipedia contributors, “Objective-C”,Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Objective-C (accessed Aug 2010). |
| **SDK** | Software Development Kit |
| **Silverlight** | Silverlight is a browser plug-in from Microsoft that delivers media and rich interactive applications for the Web. It runs in multiple browsers and platforms and is based on the Microsoft .NET Framework. |
| **SL4B** | StreamLink 4 (for) Browsers |
| **SL4F** | StreamLink 4 (for) Flex |
| **SL4i** | StreamLink 4 (for) iOS |
| **SL4J** | StreamLink 4 (for) Java |
| **SL4N** | StreamLink(4).NET |
| **SL4S** | StreamLink 4 (for) Silverlight |
| **The Web** | The World Wide Web. |
| **WAN** | Wide Area Network |

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**