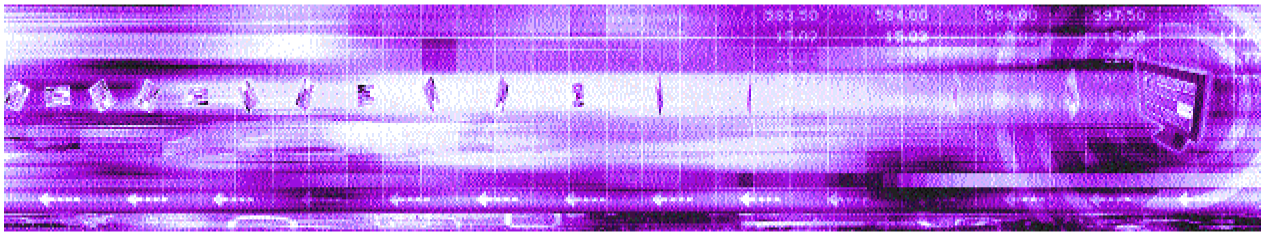


Caplin DataSource Overview 4.4

April 2008 ■ Confidential

CAPLIN

Contents

1 Preface

1.1 What this document contains

This document gives a technical overview of Caplin DataSource.

It aims to provide an understanding of

- what Caplin DataSource is and how it can be used,
- fundamental DataSource concepts & features,
- how DataSource fits into the Caplin Platform as a whole,
- how DataSource can integrate with your own software and network infrastructure,
- what “off the shelf” DataSource applications are available,
- DataSource SDKs that are used to produce custom DataSource applications.

1.2 Who should read this document

This document is intended for anyone who requires an introduction to Caplin DataSource.

Typical readers include:

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

1.3 Related documents

- **Caplin Platform Overview**

  Gives a technical overview of the whole Caplin Platform.
- **DataSource for C SDK Documentation**

  On-line reference documentation for the Caplin DataSource for C SDK.
- **DataSource for Java SDK Documentation**

  On-line reference documentation for the Caplin DataSource for Java SDK.
- **Caplin StreamLink 4.4 Technical Overview**

  Gives a technical overview of Caplin StreamLink.
- **Caplin Monitoring and Management Overview**

  Describes Caplin’s management and monitoring solution and its place in the Caplin Platform architecture.
- **Caplin Enterprise Management Console Getting Started Guide**

  Describes how to configure Caplin’s Enterprise Management Console.

- **Monitoring Socket Interface Specification**

  Defines the commands and responses used by Caplin's Monitoring Socket Interface (SOCKMON).

- **DataSource JMX SDK**

  On-line reference documentation for the Caplin Java Management Extensions SDK.

1.4 Typographical conventions

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

Type	***Uses***
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.

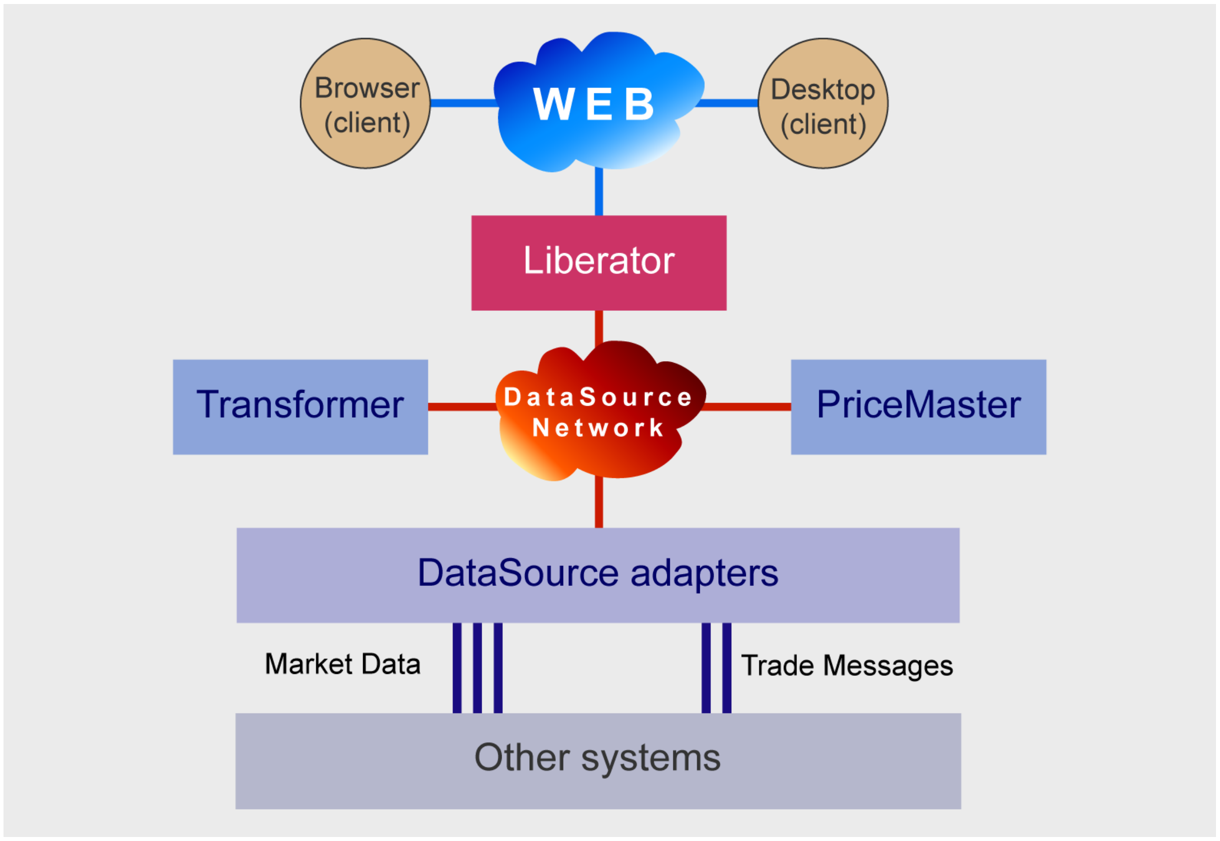
Please email your thoughts to documentation@caplin.com.

1.6 Acknowledgments

Java, *JMX* and *Java Message Service* are trademarks of Sun Microsystems, Inc. in the U.S. or other countries.

2 What is Caplin DataSource?

DataSource is the communications infrastructure on the data platform side of the Caplin Platform, that links Caplin components together and links Caplin components with other (non Caplin) systems. This is shown in the following diagram:



DataSource within the Caplin Platform Architecture

The term 'DataSource' refers to several related items:

- The **DataSource network** – a messaging network used within the Caplin Platform.
- The **DataSource protocol** – the protocol used by this messaging network.
- **DataSource applications** and **DataSource peers** –
  applications that can communicate using the DataSource protocol
- **DataSource adapters** that act as the interface between internal systems and the Caplin Platform.
- **DataSource SDKs** that are used to write custom DataSource adapters.

As the diagram shows, DataSource is used for communication between Caplin components located at the “back end” of the Platform. In contrast, the client facing “front end” software, such as browser and desktop based applications, does not use DataSource to communicate with the rest of the Platform, but instead talks to back end Liberator servers via the RTTP protocol.

3 Key concepts and features

3.1 DataSource network

The **DataSource network** is the network and supporting communications software used within the Caplin Platform. It provides a common communications mechanism for Caplin components, such as Liberator and Transformer, and allows these components to communicate with external (non-Caplin) components through DataSource adapters 5.

The network uses peer to peer messaging under a dedicated protocol, the DataSource protocol 4.

3.2 DataSource protocol

Messages are passed across the DataSource network using the **DataSource protocol**.

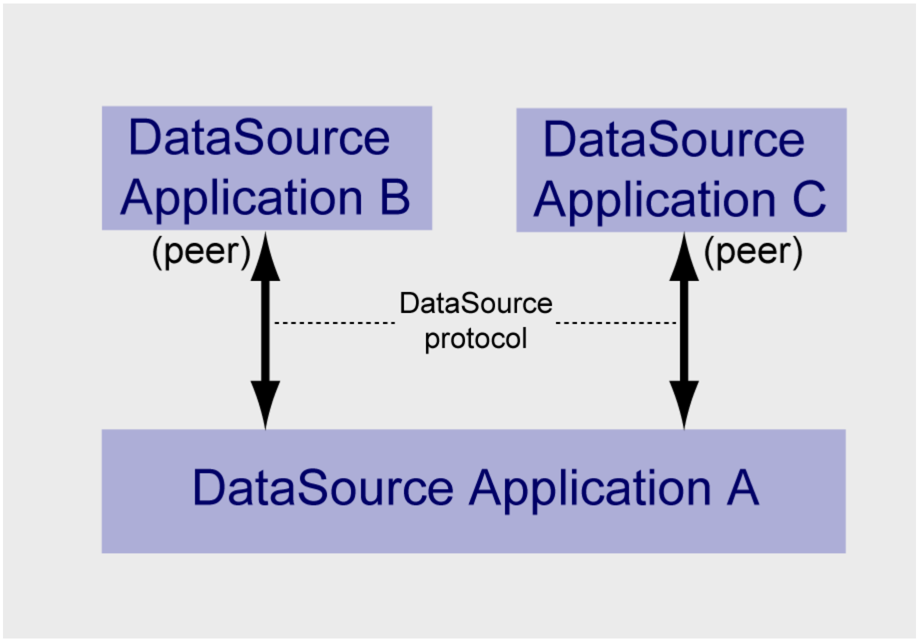
This bidirectional protocol is specifically designed to support real-time financial messaging, including the transmission of market data and financial trading messages. It is a lightweight protocol sitting on TCP/IP and implements efficient messaging with high throughput and low latency.

Most data is transmitted in text format, but the protocol will also handle binary data.

Where high security is required, DataSource connections can be configured to use the Secure Sockets Layer (SSL), providing an encrypted channel over which DataSource applications 5 can exchange data.

3.3 DataSource applications (DataSource peers)

A **DataSource application** is a software application that can communicate with another such application using the DataSource protocol. Caplin components such as Liberator server and Transformer are DataSource applications. A DataSource application can act as both a source and destination of data (see Sender and receiver 21).



DataSource applications (peers)

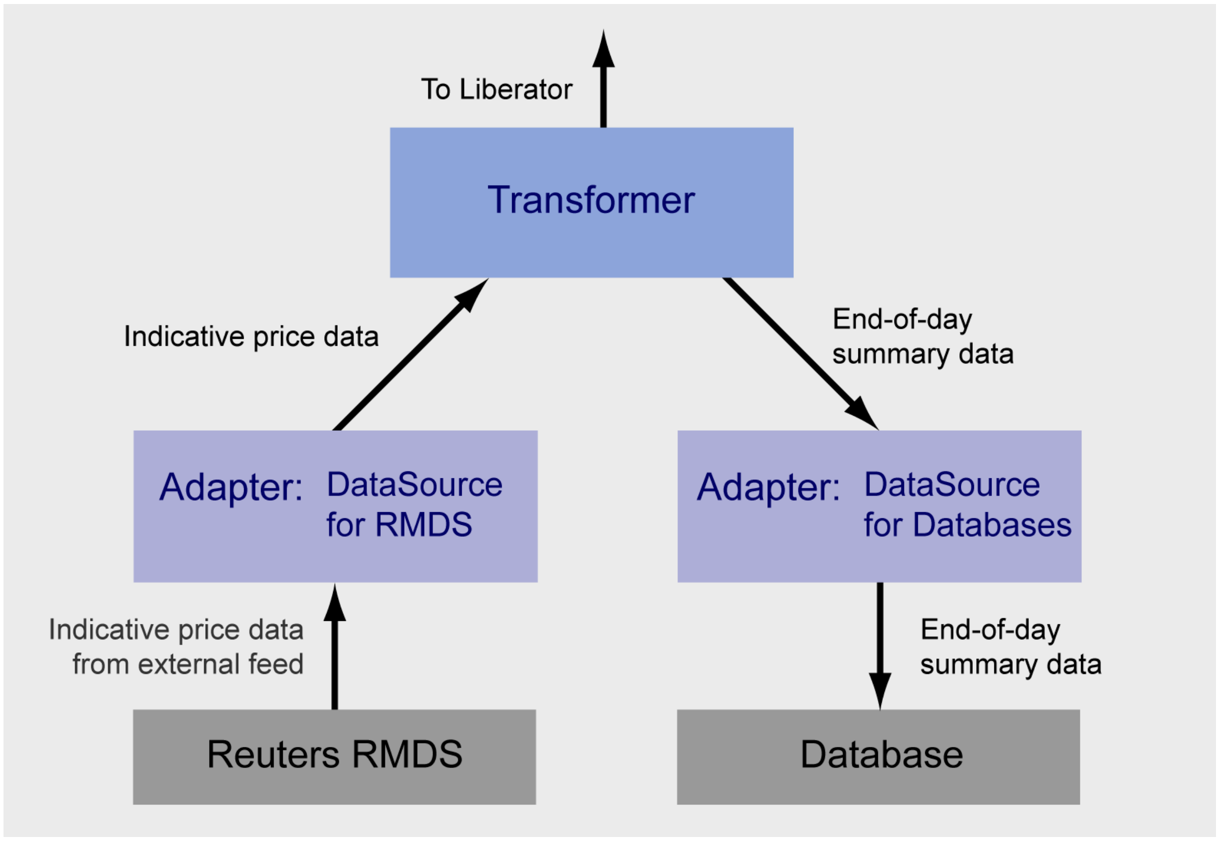
Because the DataSource protocol operates on a peer to peer basis, the other DataSource applications that are connected to a DataSource application are known as its **DataSource peers** or simply **peers**. In the above diagram DataSource Applications B and C are peers of DataSource Application A.

3.4 DataSource adapters

A **DataSource adapter** is a DataSource application that communicates with external (non-Caplin) components, such as trading order systems, market data distribution systems, vendor feeds, and databases.

An adapter transforms incoming data into DataSource messages that can be read by other Caplin components, and sends the messages to any of its DataSource peers that have requested the data. Adapters can also be implemented to convert DataSource messages received from their peers into the formats required by external systems and pass the data on to these systems.

The following diagram shows two adapters, DataSource for RMDS and DataSource for Databases.



DataSource adapters

DataSource for RMDS is an off-the-shelf adapter that interfaces to the Reuters RMDS data feed. In this example the adapter passes indicative prices to a Caplin Transformer. In turn the Transformer sends end-of-day summary data to a company database through the DataSource for Databases adapter.

Off-the-shelf adapters

Caplin can supply a number of off-the-shelf adapters for accessing existing market data distribution systems, vendor feeds, messaging middleware, and databases. These include:

- DataSource for Reuters RMDS
- DataSource for Reuters Triarch
- DataSource for TIB/RV
- DataSource for Databases

The **DataSource for Databases** adapter provides a mechanism for moving data between a JDBC^{TM} compliant database and the Caplin Platform. It includes a comprehensive XML-based configuration facility, allowing you to implement fully functioned applications without needing to write code.

3.5 DataSource library and SDKs

A DataSource application contains calls to the DataSource library. The DataSource library is a software library that allows the application to use DataSource messaging and related functionality.

The DataSource Library is packaged into a DataSource SDK, which allows you to create new applications that use DataSource messaging. Caplin provide DataSource SDKs for C, C++, and Java^{TM}, for rapid development of custom adapters.

A DataSource SDK contains:

- APIs to perform DataSource-specific tasks within an application.
- A configuration file to use as a template when configuring the DataSource aspects of your application.
- Example programs created using the kit, to demonstrate how it can be used.
- Documentation on how to use the SDK, including the APIs.

3.6 Custom DataSource adapters

You use the DataSource SDKs to write custom DataSource adapters. A custom DataSource adapter allows you to feed data into the Caplin Platform from your own internal (non Caplin) systems, or feed data back to your internal systems that has been modified or enhanced by Caplin components, such as Transformer and PriceMaster.

4 About the data

The following sections explain the types of data that can be passed between peers via the DataSource protocol.

4.1 Objects, subjects, symbols, and fields

DataSource messages are handled within DataSource applications as **objects**.

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

Simple record object:

Symbol	Fields
FX/EURUSD	price=1.334

In this example the record object is sent by a DataSource adapter that provides indicative prices obtained from a Foreign Exchange price feed. The symbol FX/EURUSD identifies the record as the indicative price for conversion between Euros and US Dollars. The price field contains the exchange rate in US dollars per Euro.

Subjects can have an arbitrary format, but they must start with a "/", as shown in the example above, if the data is to be fed to a Liberator server for ultimate consumption by end users. Subjects are usually organized in a "/" separated directory structure (see Directories 8 below).

The actual symbol structure and naming conventions used depend on the type of system that the Caplin Platform 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 the Caplin Platform and for presentation to end users (see Name mapping 29 under Data handling features).

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). As with symbols, the fields used to convey information about a particular financial instrument depend on the data source to which you are connected.

Internally to the DataSource protocol, fields are referred to by number, but DataSource applications can refer to individual fields by number or by name. The mapping between numbers and names is defined in a configuration file.

Directories

The Caplin Liberator server understands and utilizes the concept of a directory based hierarchical name space for objects. 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/"

This directory based structure is also understood by the RTTP protocol that Liberator uses to communicate with its client applications (see the **Caplin StreamLink 4.4 Technical Overview**

However, the DataSource protocol is not restricted to sending data containing such name structures, because it does not have a directory object. For example, you can send an object containing the symbol /FX/USD from a DataSource adapter to a Liberator server, but the DataSource protocol attaches no meaning to "/FX/USD", even though the Liberator does.

4.2 DataSource object types

The DataSource protocol is designed to handle financial market data and financial trading messages. The data in a message is formatted as a particular type – an “object type”. The following sections describe the object types that DataSource supports. These types are also used within the RTTP messages passed between Caplin Liberator and its client applications, so the Caplin Platform has a set of standard formats for data passing between DataSources and clients.

- Records
- Containers – referring to sets of other records
- Pages
- News headlines and news stories
- Permissions objects

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.

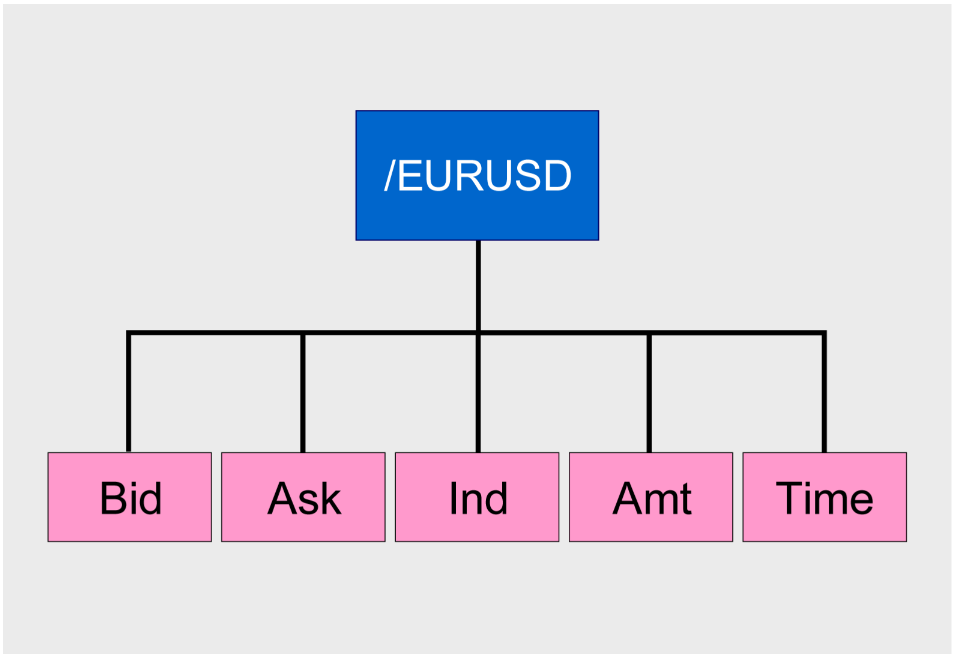
DataSource records can be structured in three different ways – these are known as type 1, type 2, and type 3 records.

Records can be used to hold trade messages 15 as well as financial market data.

Type 1 records

The majority of record based data is structured as Type 1 records. 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.



Example of a Type 1 record

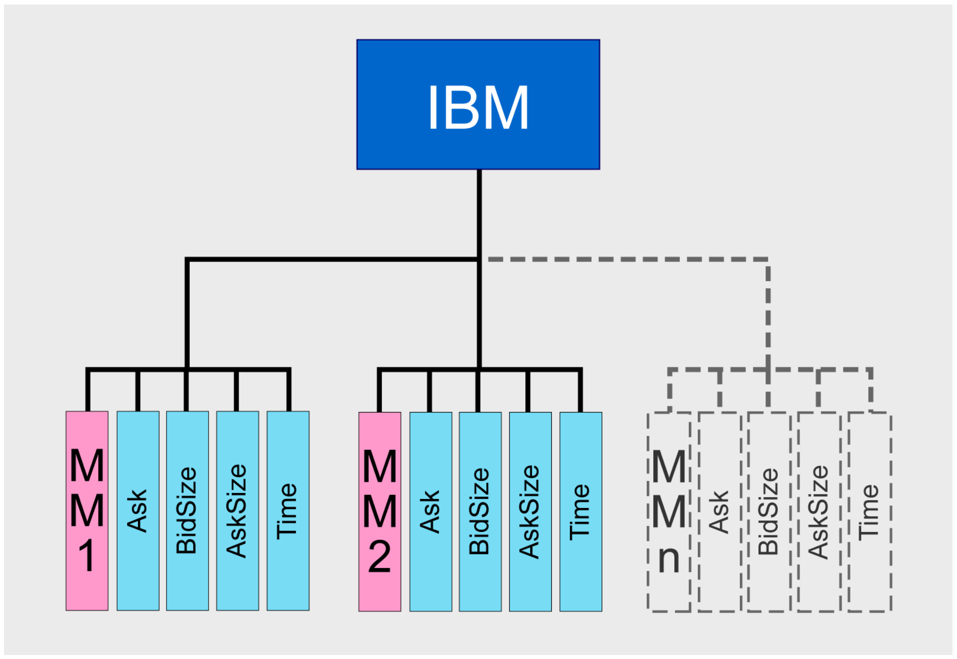
Here the single record /EURUSD contains a quote data for conversion between Euros and US Dollars. It has one level consisting of five fields: Bid (bid price), Ask (ask price), Ind (indicative price), Amt (maximum amount of dollars that can be traded at the quoted price), and Time (the date and time of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

Whenever an update arrives in Caplin DataSource for any of these fields, the value is over-written. A user newly subscribing to /EURUSD would then see this new value; the previous value would not be available.

Type 2 records

Type 2 records are often referred to as “level 2” data, as they are mostly used for level 2 quote data. 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.



Example of a Type 2 record

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 marker 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 records is to feed the display of a market order book that is in a tabular format:

IBM:

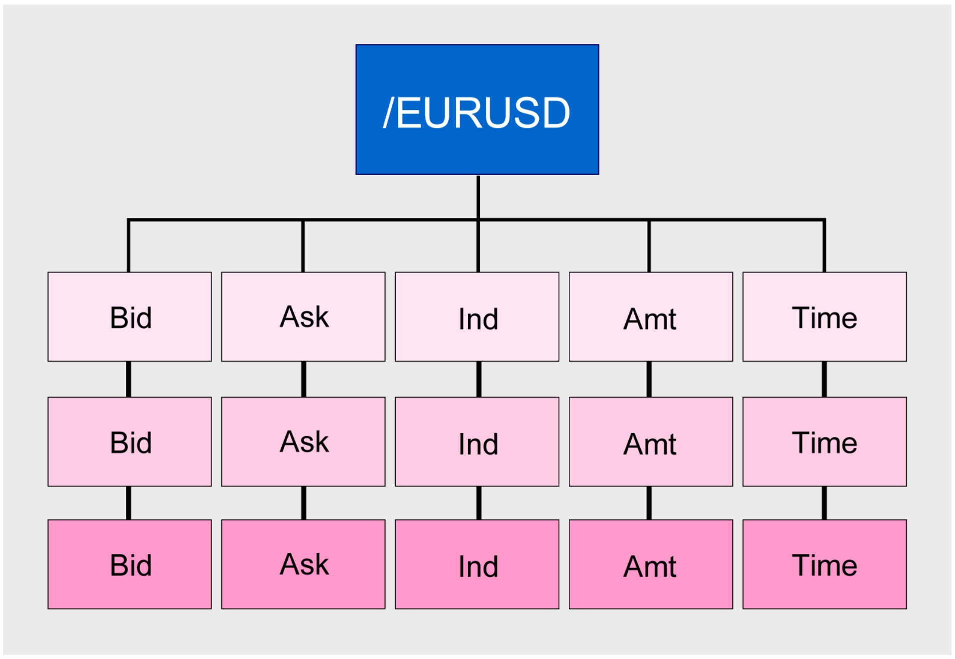
Market Maker	Ask Price	Ask Size
MM1	1.9823	1000
MM2	1.9999	2000
MM3	1.9613	1000

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

Type 3 records

Type 3 records store the history of updates to the record. A common use for records 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 cache is deleted and the update list starts again.

The following diagram shows a Type 3 record structure recording the history of quotations in a foreign exchange trading system.



Example of a Type 3 record

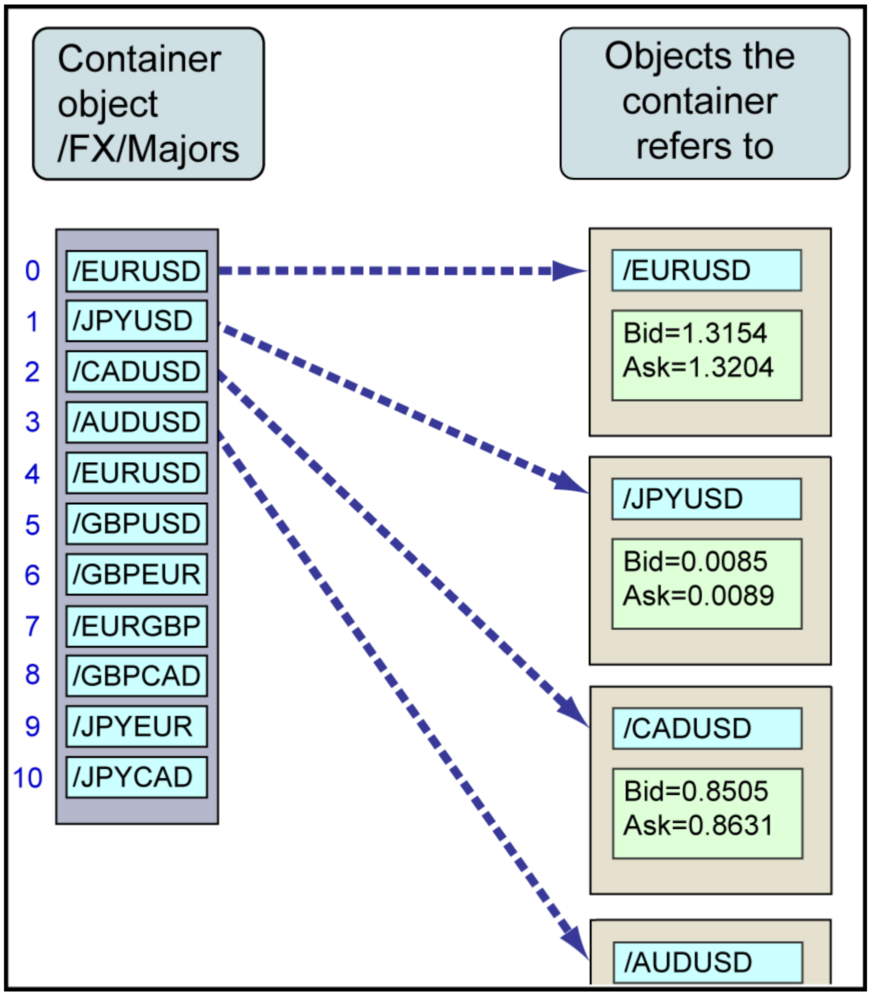
This particular record holds the history of quotes for conversion between Euros and US Dollars. The boxes running from left to right are the fields of the record; these are Bid (bid price), Ask (ask price), Ind (indicative price), Amt (maximum amount of dollars that can be traded at the quoted price), 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 record structures, and it allows you to configure the amount of update history to be retained in the structure.

Containers

A container object holds a set of references to other objects. This allows a DataSource to group objects together, as shown in the following diagram.



Structure of a container object

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

Client applications can subscribe to containers via Liberator. The client is automatically subscribed to the items referenced by the container, and the Liberator can supply the client with updates for a subset of these items as specified by the client. For more information on how clients can use containers, see the **Caplin StreamLink 4.4 Technical Overview**.

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 a news story object (see below), 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 **news story** is an arbitrary length text item, referred to by one or more news headlines.

Permissions objects

A permissions object allows changes in user permissions on objects to be sent between DataSources in real time.

The meaning of a permissions object and the actions that are taken on it are not preset, but are programmed according to business requirements. For example, a financial trading system may use custom permissions objects that allow the user to trade on objects using particular trading models, such as ESP (Executable Streaming Protocol) or RFS (Request for Stream).

After defining the permissions objects you require and the format and meaning of their content, you use the DataSource SDK to implement a new DataSource application (or modify an existing one) that generates the permission objects.

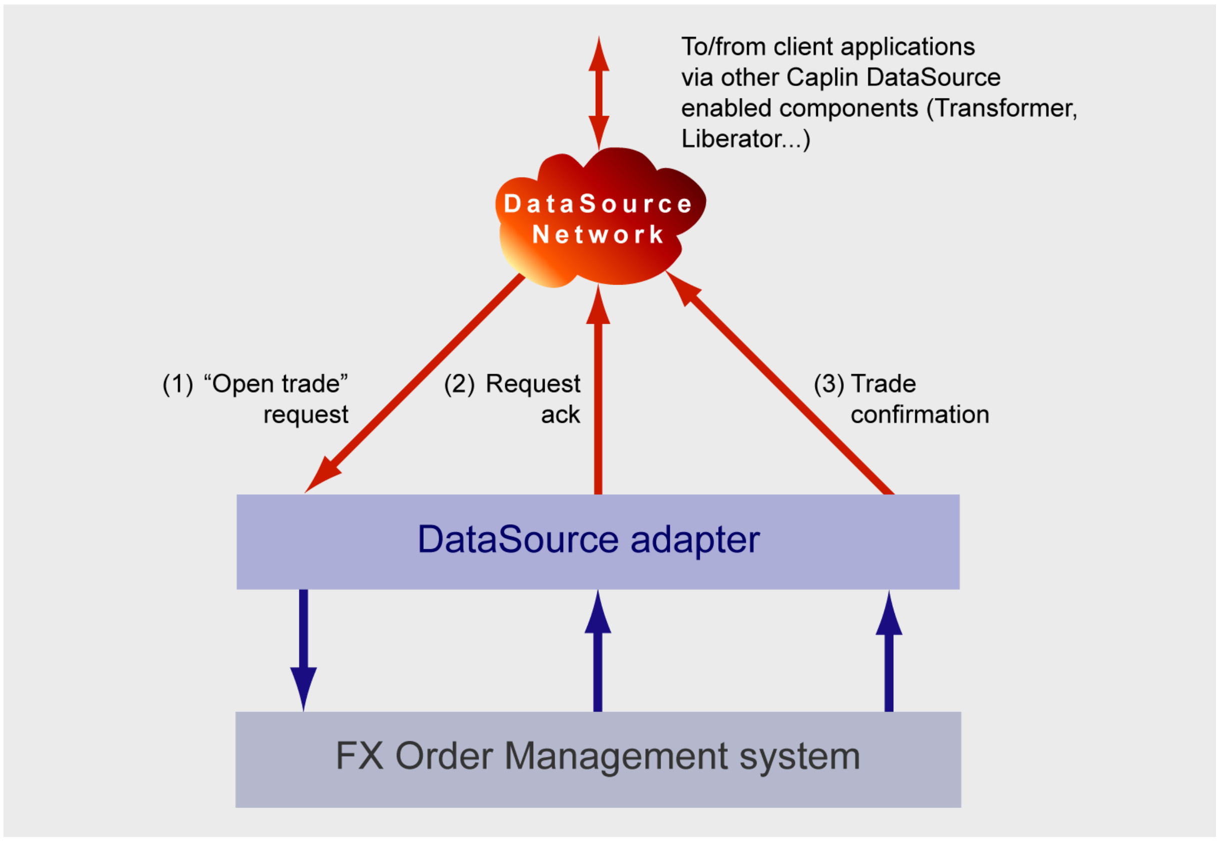
The Liberator server can receive and process updates to permissions objects via a custom auth (authorization) module, which interprets the updates and modifies access permissions accordingly.

Client applications can also make use of permissions objects. A client can subscribe to particular permissions objects, and receive updates to them from a Liberator server, through the standard update mechanism. The client then uses the updated permission information to modify the way the application behaves.

4.3 Trade messages

When the Caplin Platform is deployed in a trading system the DataSource network can be used to pass trade messages between Caplin components. Trade messages are transmitted across the DataSource network as type 1 records 10

The following examples show a typical set of trade messages received and sent by a DataSource adapter when an end user executes a foreign exchange trade. The DataSource adapter passes trade requests on to an external order management system, and passes the resulting responses on to other Caplin Platform components for ultimate transmission to the end-users making the trades, as shown in the following diagram.


To/from client applications via other Caplin DataSource enabled components (Transformer, Liberator...)
DataSource Network
(1) "Open trade" request
(2) Request ack
(3) Trade confirmation
DataSource adapter
FX Order Management system

Trade messaging using DataSource

In the examples the trade is a spot trade following the executable streaming price (ESP) trading model.

The subject of each record is the same, /TRADE/FX, identifying the record as a foreign exchange trade message. The individual message types that make up a trade transaction are identified by a "message type" field (MsgType).

(1) DataSource adapter receives request to open a trade:

Subject	Fields	Explanation of fields
/TRADE/FX	MsgType=Open	Message type: Open. Request to open (start) a trade.
	MsgVersion=0	Message version, used for guaranteed messaging.
	RequestID=1179314551125	Unique request ID that enables the client application to match subsequent response messages with this request message.
	Account=acct1	End user's trading account ID.
	TradingProtocol=ESP	Trading protocol: Executable Streaming Price.
	TradingType=SPOT	Trading type: Spot trade.
	BaseCurrency=AUD	Base currency being traded: Australian Dollars.
	TermCurrency=USD	Term currency (the other currency involved in the trade): US Dollars.
	DealtCurrency=AUD	Dealt currency (currency being bought or sold): Australian Dollars.
	BuySell=SELL	Buy/sell indicator: Sell (Australian dollars).
	Amount=2000000	Amount being traded in dealt currency units: 2,000,000 Australian dollars.
	Price=0.8255	Price: 0.8255 US Dollars per Australian Dollar
	Tenor=SPOT	Tenor (settlement date): Standard spot trade settlement date.

(2) DataSource adapter sends a request acknowledgement:

Subject	Fields	Explanation of fields
/TRADE/FX	MsgType=OpenAck	Message type: OpenAck. Acknowledges the request to open a trade.
	MsgVersion=0	Message version, used for guaranteed messaging.
	RequestID=1179314551125	Unique request ID that enables the client application to match this response with the original request message.
	Status=OK	Trade request accepted.
	TradeID=1000012	Trade ID: Unique identifier of the trade, generated by the external order management system.

(3) DataSource adapter sends a trade confirmation message when the trade has been executed:

Subject	Fields	Explanation of fields
/TRADE/FX	MsgType=TradeConfirmation	Message type: TradeConfirmation Confirms that the trade has been executed.
	MsgVersion=0	Message version, used for guaranteed messaging.
	RequestID=1179314551125	Unique request ID that enables the client application to match this response with the original request message.
	TradeID=1000012	Trade ID: Unique identifier of the trade.
	TradingType=SPOT	Trading type: Spot trade.
	Price=0.8255	Price at which the trade took place: 0.8255 US Dollars per Australian Dollar

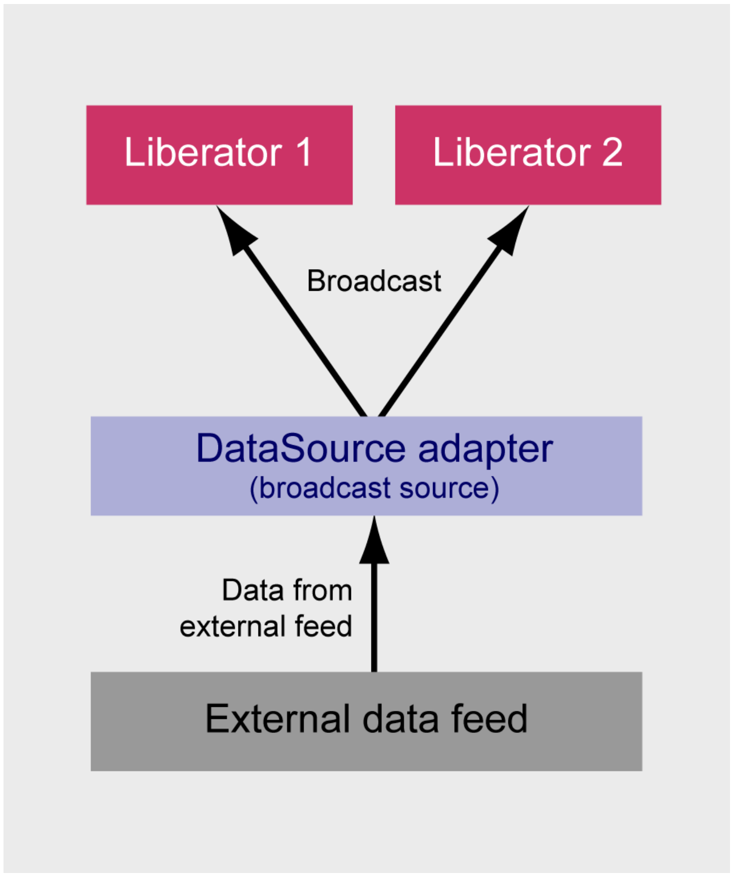
5 Messaging features

5.1 Broadcast versus subscription

DataSource supports two models of message propagation: broadcast and subscription

Broadcast messaging (broadcast source)

A DataSource peer can be configured as a **broadcast source**. This means that it sends data by broadcasting the data to all the peers attached to it, as shown in the following diagram.

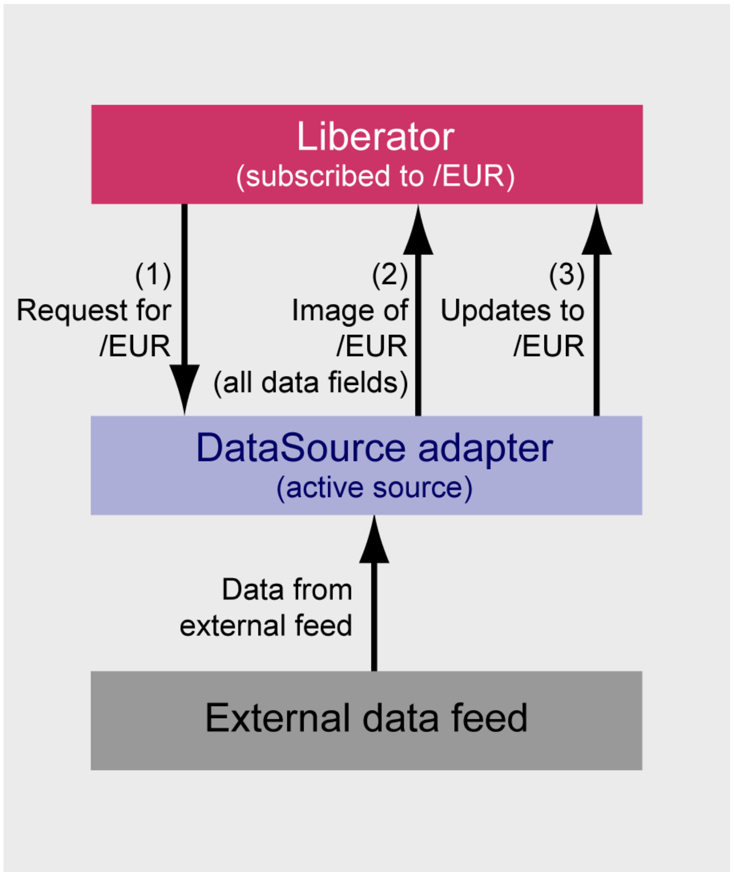


Broadcast DataSource and recipients

Subscription (active source)

In the **subscription** model (see the following diagram) a DataSource peer that wishes to receive information **subscribes** to the types of data (objects) it requires. It does this by sending a **request** for each object, or set of objects, to the DataSource peer that can supply the data.

The supplying (sending) DataSource is configured as an **active source**; this means it will only respond to requests for objects. When the DataSource receives a request (1), it sends the current image of the object back to the requesting DataSource (2), and then all subsequent updates for the object (3), until the requesting DataSource unsubscribes or disconnects.



Active DataSource and recipient

An active source is subscription aware; it keeps track of which of its peers have subscribed to (requested) which objects. When a peer subscribes to an object the active source sends it the most up to date image of the whole object. When any field values change within the object, the active source sends just the updated fields to the peers that have subscribed to that object.

Hybrid DataSources

A DataSource peer acting as a source of data can be both an active source and a broadcast source, that is, a hybrid source. Conversely, DataSources that receive data can be configured to use data services so they explicitly subscribe to objects with defined sets of subject names ("name spaces"), but can also receive other objects via broadcast messages. See the section on Data services 23.

5.2 Broadcast feeds and Active Cache

A DataSource adapter can turn an incoming broadcast-style data feed into an active request-based one. This capability is provided through the DataSource Active Cache mechanism.

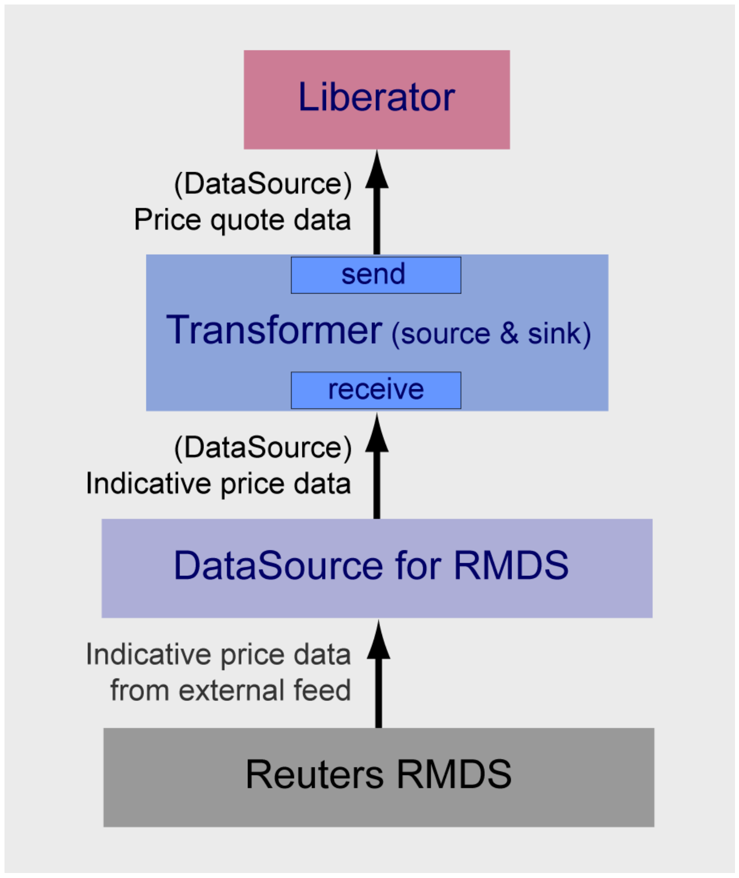
Using Active Cache, a DataSource programmer can send broadcast data to the DataSource API in a simple way, avoiding the need to explicitly program data caching and subscription management. The DataSource library caches the data and then handles active requests (subscriptions) for the data made by other DataSource peers.

This feature is useful when one DataSource adapter is feeding multiple DataSource peers and each peer is only interested in a small subset of the overall data.

5.3 Sender and receiver (source and sink)

A DataSource application can both send and receive data. When it sends data it is called a **source** and when it receives data it is called a **sink**.

For example, a Caplin Transformer can be a sink by subscribing to objects supplied by one or more DataSource adapters (see diagram below). At the same time it can also be an active source, responding to requests from a Caplin Liberator server. When the Transformer receives an update for one of its subscribed objects from a DataSource adapter, it transforms the data in the object according to configured business rules (for example it might calculate the spread applying to a received indicative price). It then sends the updates for the transformed object on to the Liberator that previously subscribed to the object.



Sending and receiving DataSource

5.4 Monitoring connection health

A DataSource application can monitor the connections to its DataSource peers by

- detecting loss of connection,
- timeouts placed on responses to DataSource requests,
- monitoring heartbeat messages exchanged with peers.

Request timeout values and heartbeat timer values are configurable.

When the DataSource detects that it has lost connection to a peer (which could be because of a network error, because the peer itself has failed, or because heartbeats were missed), it can try to reconnect or it can fail over to a different peer. Failover behaviour is specified using data services – see Data services 23 and Prioritized access and failover 25.

If the connection between two DataSource peers is lost, messages will be queued until the connection can be reestablished. The queue is flushed when a reconnection is successful. The length of the queue is configurable for each peer connection.

6 Data services

DataSource applications can obtain data from other DataSource peers through **data services**. A data service is an abstraction layer that allows the receiving DataSource to request objects, based on their subject names, without needing to know the specific DataSources from which the objects originate. The data service is defined through configuration, which includes

- a name that identifies the service,
- a regular expression pattern that defines the objects the service can supply,
- one or more DataSource peers to which requests for the objects are sent.

Data services allow you to group DataSource peers together so you can

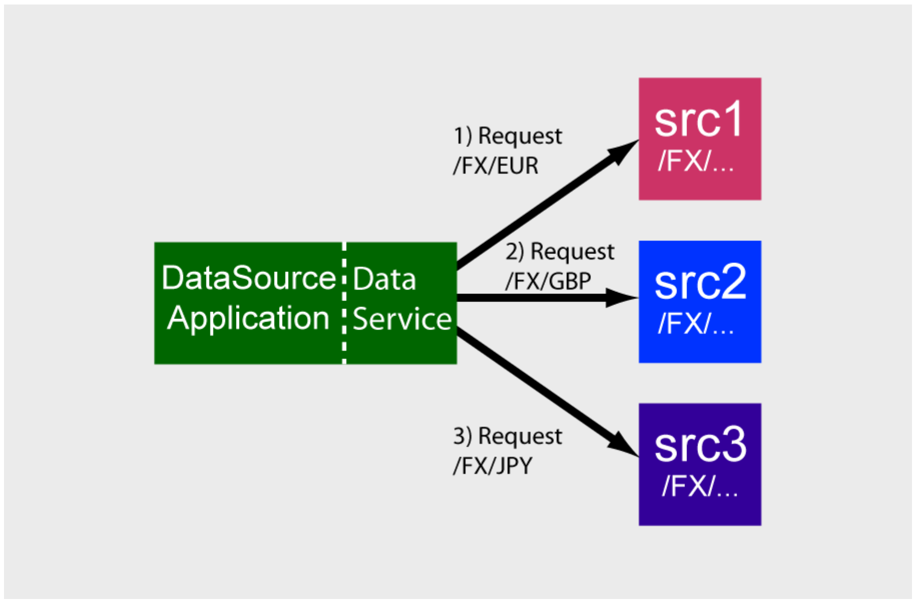
- implement load balancing 23,
- implement prioritized access and failover 25,
- provide access to objects when you do not know which of the peers hold the data (see Requesting objects from a set of peers 27).

Caplin components such as Transformer and the Liberator server make extensive use of data services. You can build data services into your own DataSource applications using the C/C++ DataSource SDK.

6.1 Load balancing

You can use data services to implement load balancing.

For example, assume the three DataSource peers src1, src2, and src3 can all supply the same set of foreign exchange indicative price data, under the symbol /FX, as shown in the following diagram.



Load balancing using a data service

You can define a data service consisting of the three peers, configured so that a request for data whose symbol begins with "/FX" is directed to the peer with the smallest number of existing subscriptions. This keeps the requests balanced across the peers, even though the DataSource application using the service may also unsubscribe from (discard) previously subscribed symbols.

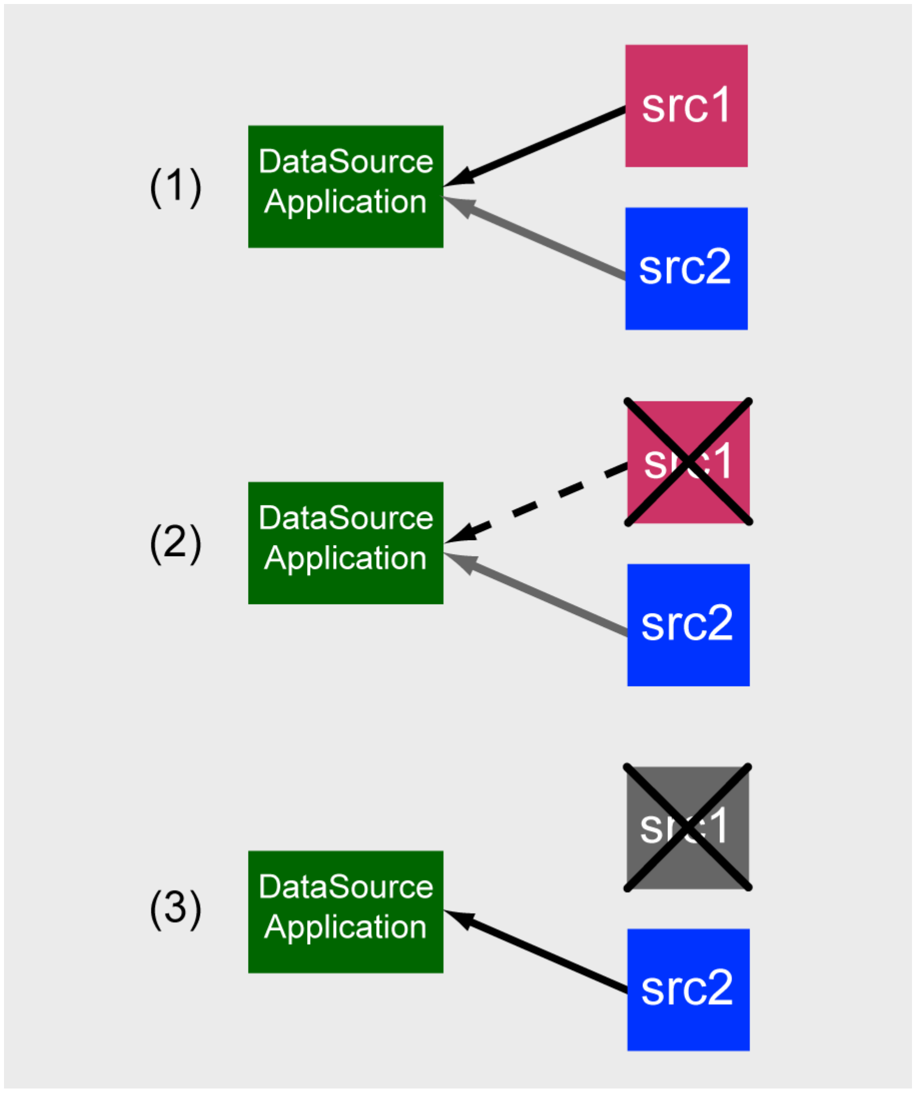
Each DataSource peer provides updates just for the specific symbols requested of it, and because the original requests are distributed evenly across the peers, the updates are also evenly distributed, thus balancing the load on the peers.

Note: In versions of the Caplin Platform prior to 4.4, load balanced requests are issued to the DataSource peers in a round-robin fashion, rather than according to subscription count.

6.2 Prioritized access and failover

Data services can be used to provide failover capability by means of prioritized access to DataSource peers.

This is shown in the following diagram.



Failover example

The DataSource application uses a data service that is configured with two data source peers, src1 and src2. Each peer is in a different “priority group”, such that peer src1 has priority over peer src2. The DataSource application is connected to both peers.

At (1) the DataSource application has requested data from the data service and is receiving updates via peer src1, the highest priority peer.

At (2) the connection between the DataSource application and peer src1 has failed. This could be because of a network problem, or because src1 has failed.

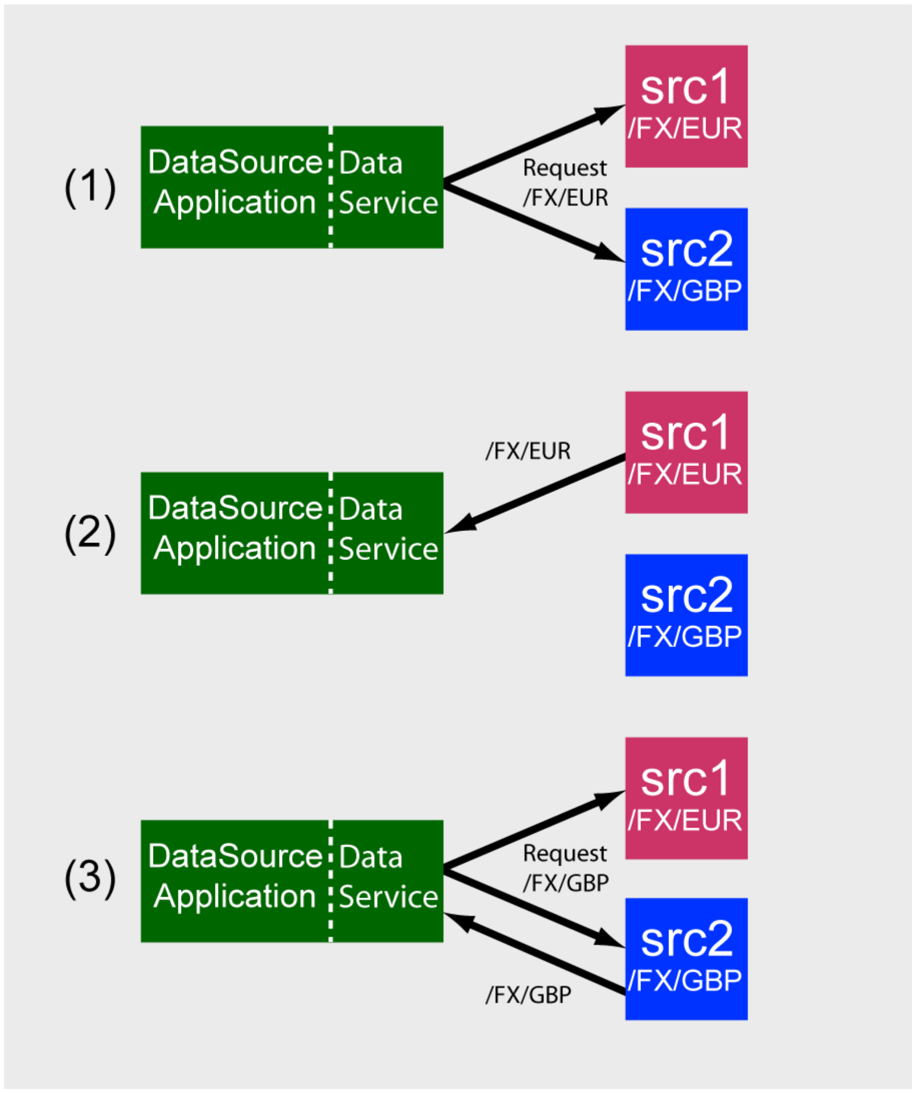
At (3) the data service responds by failing over the subscriptions from src1 to the peer with the next highest priority in the priority ordered list – in this case peer src2. The DataSource now obtains all further updates for these subscriptions from src2.

You can configure any number of alternate failover DataSource peers in priority order. If the data service cannot fail over the subscriptions to any of the alternate peers, it will pause, according to a degrading retry algorithm, and then try all the connections in the list again, and so on. The timeout values governing the failover behaviour are all configurable.

6.3 Requesting objects from a set of peers

Using data services you can request data from more than one DataSource peer. This allows you to access objects when you do not know which of the peers hold the data.

For example, consider a data service configured to connect to two DataSource peers, src1 and src2, as shown in the following diagram.



Requesting objects from a set of peers

The data service configuration specifies that any request with subject (symbol) beginning “/FX/” should be directed to both src1 and src2. Assume that /FX/EUR is held on src1 and /FX/GBP is held on src2.

When the DataSource application requests /FX/EUR from the data service, the request is sent to both src1 and src2 (see 1 in diagram). Only src1 sends back the data for /FX/EUR, and subsequent updates for this symbol (2 in diagram). Similarly, when the DataSource application requests /FX/GBP, only src2 sends back the data and subsequent updates for this symbol (3 in diagram).

Note that if both src1 and src2 have the requested data they will both send back updates, so this facility should normally only be used when it is not known which peer has the data but only one of them has.

7 Data handling features

7.1 Delay channels

Using the “delay channel” mechanism a DataSource can delay sending its data for a specified time. The delayed data can be identified as such by adding a prefix to the object name, so for example the delayed version of /EUR could be sent as /DELAY/EUR. The live channel can remain in operation or it can be blocked.

You can specify delay channels through configuration or programmatically. The configuration options specify a delay channel for all data sent by the DataSource. When developing a custom DataSource adapter you can use the delay API functions of the DataSource SDK to create any number of delay channels, applying finer control over what is delayed and for how long.

7.2 Name mapping

A DataSource application can be configured to change names (symbols) passed into it into a different format. For example, on receiving an object with subject name ABC.FX.EUR, the DataSource application could send it on with the modified subject name /FX/EUR

This facility can be used to map data between different namespaces and/or symbologies. It allows DataSources to be easily deployed in multiple implementations that use different namespaces.

8 Configuration

DataSource is highly configurable. Each DataSource application has a configuration file that defines how it behaves as a DataSource. Items that can be configured include:

- How the DataSource is referred to by its peers:
  - DataSource ID number
  - DataSource name
- Network connection information:
  - The network interface and port to listen on for connections from DataSource peers
- Connections between DataSource applications.
  Each DataSource application defines what DataSource peers it can connect to and the nature of the connections, such as:
  - The peer ID and name
  - The peer type (e.g. active source)
  - Restart and reconnection behaviour
  - A request timeout for the peer connection
  - Heartbeat timers
  - Data services:
  - The name of the data service
  - A request timeout for the data service
  - A regular expression pattern that defines the objects the service can supply
  - The peers that take part in the data service and how they are configured for requesting data, load balancing, and failover purposes (see Requesting objects from a set of peers 27, Load Balancing 23, and Prioritized access and failover 25).
- The mapping between field numbers and field names.

DataSource applications built using the C/C++ DataSource SDK use plain text configuration files, whereas the configuration for Java based DataSource applications is defined in XML.

9 Monitoring and management

The following facilities are available to help you monitor and manage DataSource applications:

- Logging
- Monitoring modules
- Latency Chaining

9.1 Logging

DataSource applications have inbuilt logging capability provided through the DataSource SDKs. Log files can be cycled on a timed basis. You can change various log attributes through configuration; for example, the logging level, log file name and directory, cycle time, and maximum log file size.

Using the DataSource SDK logging API you can create your own customized log files and log file contents.

All DataSources can generate a packet log (when enabled), which records all data sent and received by the application, including the source and destination information.

9.2 Monitoring modules

Applications built with Caplin's DataSource SDK can be enabled for monitoring and management. This includes standard and custom DataSource adapters. Every DataSource application provides a minimum standard set of information to the monitoring system, including process information, connection and peer information, and access to log messages.

When writing your own DataSource applications you can easily add new items for monitoring and management, using standard DataSource API functions.

Application monitoring and management capabilities are provided by a plug-in monitoring module, which is loaded at runtime by the DataSource process. The choice of monitoring module allows the monitoring and management features to be presented using a variety of mechanisms and technologies. Currently there are two monitoring modules available, SOCKMON and JMX^{TM}.

- SOCKMON provides socket-based monitoring and management using a simple command language.
- JMX provides Java MBean-based monitoring and management for Java clients, using a built-in JMX Server.

Both of these modules can be used to integrate existing monitoring systems with the Caplin Platform. You can also use Caplin's Enterprise Management Console to monitor and manage DataSource applications installed with the JMX module.

For further information see the documentation on Caplin's management and monitoring solution:

- **Caplin Monitoring and Management Overview**
- **Caplin Enterprise Management Console Getting Started Guide**
- **Monitoring Socket Interface Specification**
- **DataSource JMX SDK**

9.3 Latency Chaining

DataSource applications can be instrumented for test purposes using latency chaining. A latency chain is a way of measuring the latency of any given message as it passes through connected DataSources. When a message is originated at the first DataSource, it is time stamped. As the message is transmitted from DataSource to DataSource, each DataSource record's an entry and exit timestamp. These timestamps allow you to find out the message latency at each point in the system.

10 Performance considerations

The DataSource protocol is designed to operate in financial application areas that require high throughput and low latency messaging.

DataSource applications are multi-threaded to take advantage of high performance computing hardware that uses multiple processors. Each peer connection is handled by a separate thread; this helps to improve performance when the Liberator is connected to several DataSources, as there will be a separate thread for each DataSource.

Even when the DataSource application communicates with just one peer, in situations where the peer is feeding in updates at a high rate it is possible to improve performance by configuring more than one connection between the peer and the DataSource application. Each connection will use a separate thread, so the updates will be spread across multiple threads.

The multi-threading capability is provided through the DataSource library when you build an application using the DataSource SDK.

11 DataSource compared to other messaging middleware

A configured network

Some messaging networks are “self-discovering”; you can just add a new network node (“peer” in DataSource terms) and the network “discovers” the new node and automatically adds it to the configuration. In contrast a DataSource network is defined through explicit configuration, which allows centralized management and control of the network topography. This is often the preferred approach of financial institutions, where security considerations dictate that changes to networks must be centrally managed and authorized.

Subscription aware

Messaging networks based on middleware, such as Java^{TM} Message Service, typically pass messages across named channels. Generally the middleware is unaware of the object naming scheme used within the channel. This means that the applications at each end of the channel have to agree and implement the subscription scheme. If an application asks for (subscribes to) an object that does not exist, the application at the other end has to handle this, otherwise the requester can only ask for objects that exist.

In contrast DataSource is subscription aware and has subscription management built in. When an application that has been constructed with the DataSource library subscribes to (requests) an object, it will under normal circumstances always get a reply. If the object exists, the DataSource peer that receives the request will send back the latest value of the object. On the other hand, if the object does not exist the peer will send back a negative acknowledgement. The DataSource API in the requesting application can also time out the request if required.

12 Glossary of terms and acronyms

This section contains a glossary of terms and acronyms relating to Caplin DataSource.

Term	Definition
Active cache	See the section Broadcast feeds and Active Cache 20.
Active source	A DataSource application that is configured to respond to DataSource Request messages received from one or more of its **DataSource peers**, by keeping track of which objects have been requested, and sending to the requesting peers updates for just those objects.
API	Application Programming Interface
Broadcast source	See the section Broadcast versus subscription 18.
Client application	In the context of the Caplin Platform this is a software application that runs on the desktop or inside a browser, and communicates with a Caplin Liberator server via the **RTTP** protocol.
DataSource adapter	See the section DataSource adapters 5.
DataSource application	See the section DataSource applications (DataSource peers) 5.
DataSource library	See the section DataSource library and SDKs 7.
DataSource object	See the section Objects, subjects, symbols, and fields 8.
DataSource object type	See the section DataSource object types 9.
DataSource peer	See the section DataSource applications (DataSource peers) 5.
DataSource protocol	See the section DataSource protocol 4.
DataSource SDK	The **SDK** used to build applications that can communicate with other applications using the DataSource protocol 4. See the section DataSource library and SDKs 7
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.
Object	See the section Objects, subjects, symbols, and fields 8. Also called **DataSource object**.
Object type	See the section DataSource object types 9. Also called **DataSource object type**.
Peer	See the section DataSource applications (DataSource peers) 5.
Protocol	A set of rules, procedures, and data formats that define the way in which data is passed between end points in a communications network.
RTTP	Real Time Text Protocol Caplin's object-oriented real-time **protocol** for the distribution of financial data and trade messages over internet-protocol networks.
SDK	Software development kit
sink	See the section Sender and receiver (source and sink) 21.
source	See the section Sender and receiver (source and sink) 21.
SSL	Secure Sockets Layer A commonly-used **protocol** for transmitting messages securely across the Internet. SSL uses the public-and-private key encryption system,

Term	Definition
	which includes the use of a digital certificate.
Subscription	See the section Broadcast versus subscription 18
Symbol	The letters used to uniquely identify a financial instrument. For example, the symbol for Microsoft's common stock on the Nasdaq market is MSFT. Many symbol naming conventions (symbologies) exist, but each is applied consistently in each market, and one symbology is used across all North American equity markets.
TCP/IP	The **protocol** used for communication across the Internet.

Contact Us

Caplin Systems Ltd.
Triton Court
14 Finsbury Square
London EC2A 1BR
UK

Telephone: +44 20 7826 9600
Fax: +44 20 7826 9610

www.caplin.com

Document version 1