CAPLIN

Caplin Xaqua 1.0

How To Use Containers

October 2011

CONFIDENTIAL

Contents

1 Preface

1.1 What this document contains

This document describes what containers are, and how Caplin Xaqua applications can use them to group and manage objects such as financial instruments for manipulation in a user interface. It explains how the elements of a container can be managed using the windowing feature of Caplin Liberator, and how Caplin Refiner can be used to filter and sort the container elements on behalf of a client.

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 Developers who need to understand what containers are and how to use them effectively.

1.3 Related documents

- **Caplin Xaqua: Caplin Refiner Benchmarks**

  Documents the results of a set of performance benchmarks for Caplin Refiner, and contains recommendations for ensuring optimal performance when filtering and sorting using Caplin Refiner.

- **Caplin StreamLink Overview**

  A technical overview of Caplin StreamLink.

- **Caplin DataSource Overview**

  A technical overview of Caplin DataSource.

- **Caplin Liberator Administration Guide**

  Explains how to install, configure, and manage the Caplin Liberator server, and includes configuration reference information.

- **Caplin Xaqua: Permissioning Overview And Concepts**

  Introduces permissioning concepts and terms, and shows the permissioning components of the Caplin Xaqua architecture.

- **Caplin Xaqua: How to Create a Permissioning DataSource Adapter**

  Describes how you can use the Permissioning DataSource API to create a Permissioning DataSource adapter.

- **Permissioning DataSource: API Reference**

  Documents the Java™ classes and interfaces that allow you to integrate Caplin Xaqua with a Permissioning System.

- **Caplin Trader: API Reference**

  Documents the JavaScript libraries that allow developers to extend Caplin Trader by writing custom JavaScript code. It includes an API for filtering and sorting containers using Caplin Refiner.

- **StreamLink 5.0 Overview**

  A technical overview of Caplin StreamLink. Applies to StreamLink 5.0 and above (for example, StreamLink.NET and StreamLink for Silverlight).

- **StreamLink .NET API Reference**

  The API reference documentation for StreamLink .NET (SL4N), which includes an API for container filtering and sorting using Caplin Refiner.

- **StreamLink for Silverlight API Reference**

  The API reference documentation for StreamLink for Silverlight (SL4S), which includes an API for filtering and sorting containers using Caplin Refiner.

- **StreamLink for Java API Reference**

  The API reference documentation for StreamLink for Java (SL4J), which includes an API for filtering and sorting containers using Caplin Refiner.

1.4 Typographical conventions

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

Type	*Uses*
aMethod	Function or method name
aParameter	Parameter or variable name
/AFolder/Afile.txt	File names, folders and directories
`Some code;`	Program output and code examples
The `value=10` attribute is...	Code fragment in line with normal text
Some text in a dialog box	Dialog box output
`Something typed in`	User input – things you type at the computer keyboard
Glossary term	Items that appear in the “Glossary of terms and acronyms”
XYZ Product Overview	Document name
◆	Information bullet point
■	Action bullet point – an action you should perform

Note: Important Notes are enclosed within a box like this.
Please pay particular attention to these points to ensure proper configuration and operation of the solution.

Tip: Useful information is enclosed within a box like this.
Use these points to find out where to get more help on a topic.

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

1.5 Feedback

Customer feedback can only improve the quality of our product documentation, and we would welcome any comments, criticisms or suggestions you may have regarding this document.

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

1.6 Acknowledgments

2 About containers

The following sections explain what containers are and what they are used for, their features and benefits, and how they work within the Caplin Xaqua architecture.

2.1 What is a container?

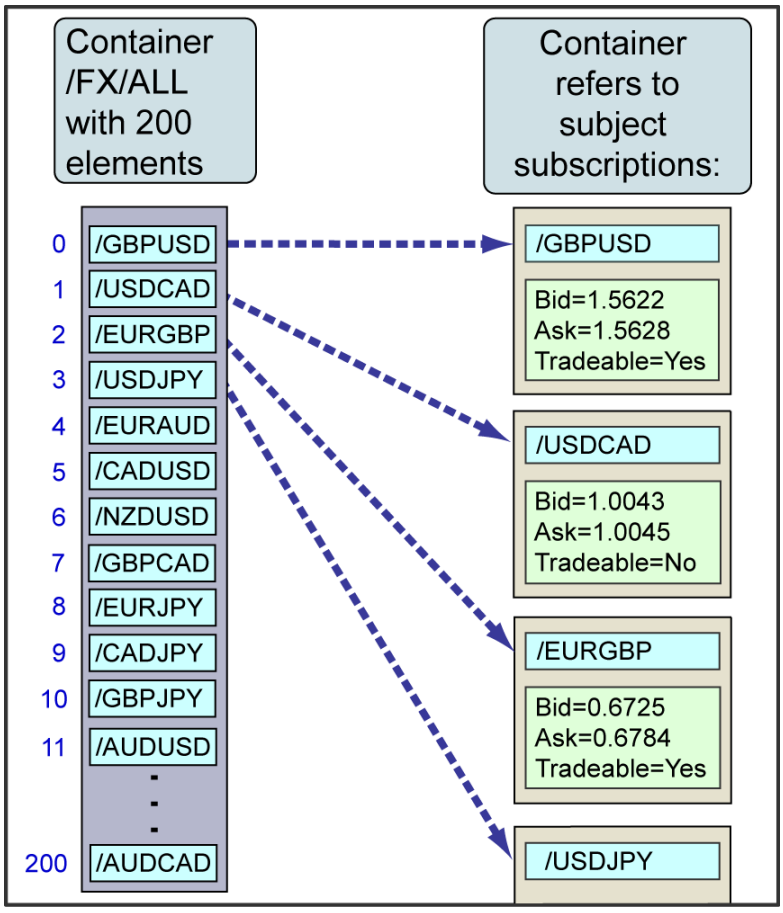
In Caplin Xaqua, containers are a useful way to group and manage objects, such as financial instruments, for manipulation in a client user interface (**UI**). They save the client application from having to locally manage and manipulate large lists of instruments.

Consider the following:

A bank wishes to make 200 common FX instruments (currency pairs) available to its trading customers through a trading application that provides updated exchange rates in real time. The list of 200 instruments is too large to display in its entirety on an end-user's screen, so the trading client application provides a scrollable UI grid window that allows the end-user to see just 6 instruments at a time.

The client could subscribe directly to all 200 instruments and manage this list itself, only displaying the 6 instruments that the end-user currently has in view in the displayed grid. However, the client still has to keep images of the other 194 instruments in local memory. It also receives (in real time) all updates to these instruments, and must maintain them even though the end-user only sees updates to the 6 instruments on view in the grid. This is clearly inefficient, and for a large list with frequent updates, the client application could perform badly or even lock up entirely.

In a Caplin Xaqua client the instruments can be put in a container, which simplifies the management of the list and reduces the load on the client application. A container object holds a list of subject names of other objects – the elements of the container. For example the elements could be the subject names of 200 currency pair records (`/GBPUSD`, `/USDCAD`, and so on), as the following diagram shows.



Structure of a container object

The client application subscribes to the container (requests it via **StreamLink**), and is then automatically subscribed to the subjects that the container refers to. So when the client subscribes to the FX currency pairs container, it is automatically subscribed to the 200 currency pair records, receiving updates for them each time the field values change.

Typically some fields in the subscribed records are static, meaning they have a set value that does not change, such as the `Tradeable` field (in this case the value of the `Tradeable` field would be determined for each user when they log in, depending on which currencies they are authorized to trade and which ones they can only view). Other fields are dynamic, meaning they change value; in this example, the values of the `Bid` and `Ask` price fields will change throughout the day, perhaps as often as several times a second.

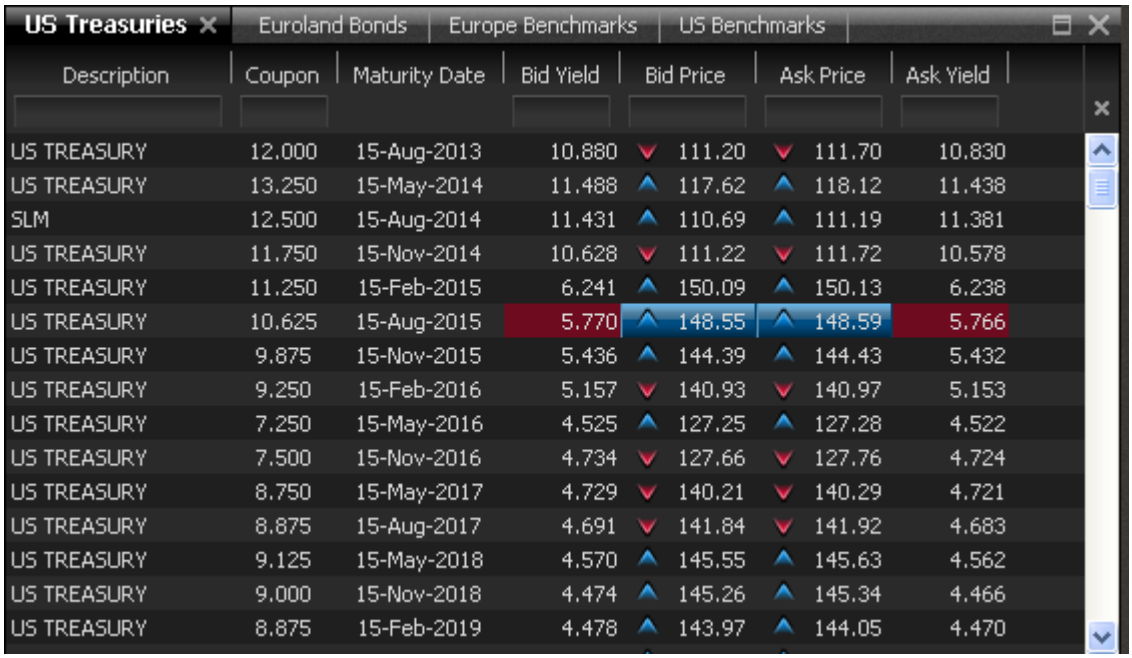
The container is located within Caplin Xaqua and is managed by the Liberator (and sometimes other Caplin Xaqua components) on behalf of all the subscribing clients. For example, the Liberator handles the addition and deletion of container elements; it automatically adjusts client subscriptions accordingly and communicates the changes to the clients via StreamLink.

The order of the elements within the container is defined by the DataSource that provides the container. This ordering is known as “**natural order**” or “**container order**”.

Containers in a user interface

Containers typically hold lists of financial instruments that are displayed in a grid format. For example:

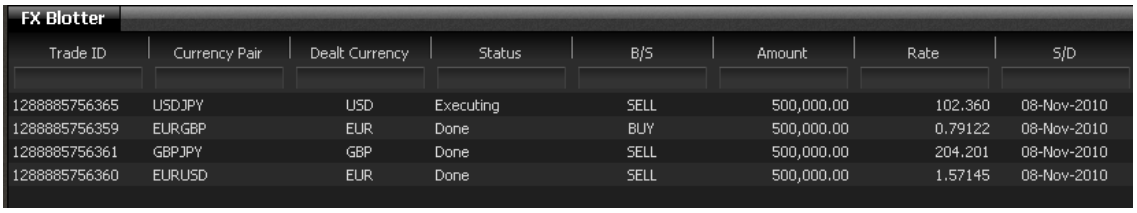
- Fixed Income – all the bonds held by the bank.
- Foreign Exchange – all the major FX currency pairs traded by the bank, all the minor FX currency pairs traded by the bank.



Description	Coupon	Maturity Date	Bid Yield	Bid Price	Ask Price	Ask Yield
US TREASURY	12.000	15-Aug-2013	10.880	111.20	111.70	10.830
US TREASURY	13.250	15-May-2014	11.488	117.62	118.12	11.438
SLM	12.500	15-Aug-2014	11.431	110.69	111.19	11.381
US TREASURY	11.750	15-Nov-2014	10.628	111.22	111.72	10.578
US TREASURY	11.250	15-Feb-2015	6.241	150.09	150.13	6.238
US TREASURY	10.625	15-Aug-2015	5.770	148.55	148.59	5.766
US TREASURY	9.875	15-Nov-2015	5.436	144.39	144.43	5.432
US TREASURY	9.250	15-Feb-2016	5.157	140.93	140.97	5.153
US TREASURY	7.250	15-May-2016	4.525	127.25	127.28	4.522
US TREASURY	7.500	15-Nov-2016	4.734	127.66	127.76	4.724
US TREASURY	8.750	15-May-2017	4.729	140.21	140.29	4.721
US TREASURY	8.875	15-Aug-2017	4.691	141.84	141.92	4.683
US TREASURY	9.125	15-May-2018	4.570	145.55	145.63	4.562
US TREASURY	9.000	15-Nov-2018	4.474	145.26	145.34	4.466
US TREASURY	8.875	15-Feb-2019	4.478	143.97	144.05	4.470

FI instruments (US Treasury Bonds)
displayed in a grid with an underlying container

Containers can also be used to manage the data displayed in trade blotters, since the blotter is typically based on a grid. A trade blotter is a record of the details of trades made by an end-user. It shows the status of trades in progress and the history of trades.



Trade ID	Currency Pair	Dealt Currency	Status	B/S	Amount	Rate	S/D
1288885756365	USDJPY	USD	Executing	SELL	500,000.00	102.360	08-Nov-2010
1288885756359	EURGBP	EUR	Done	BUY	500,000.00	0.79122	08-Nov-2010
1288885756361	GBPJPY	GBP	Done	SELL	500,000.00	204.201	08-Nov-2010
1288885756360	EURUSD	EUR	Done	SELL	500,000.00	1.57145	08-Nov-2010

A trade blotter

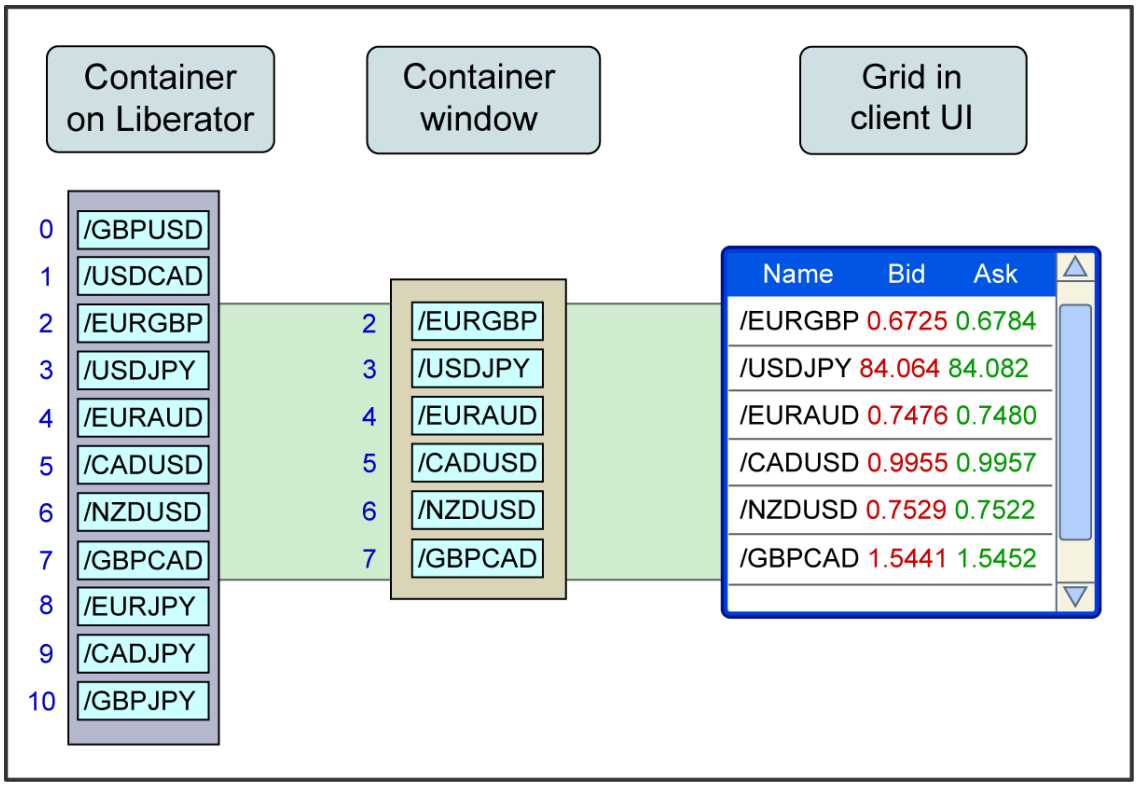
2.2 Container windowing

A simple subscription to a container of 200 currency pairs would cause the Liberator to send images and updates for all 200 instruments to the subscribing clients. This appears to offer little advantage over the client subscribing to each of the instruments individually and managing them locally. However, containers become really useful when you use their windowing mechanism.

The following example shows how container windowing works.

The client requests the Liberator to provide a windowed view of the elements in the container, say a window of just 6 elements. The Liberator subscribes to just these 6 elements on the relevant DataSource adapter(s), and sends the client images and updates for them.

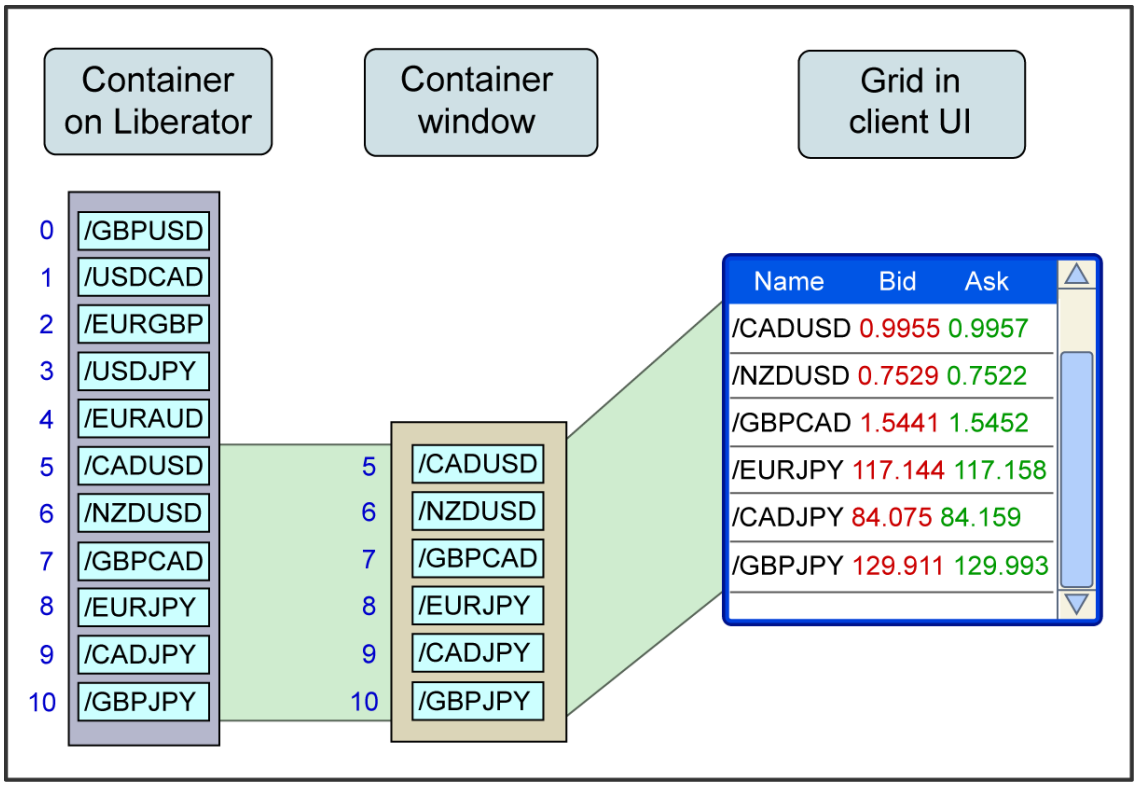
The client maps the container window to a grid window in the client UI, so the end-user can see the 6 instruments being updated in real time. The following diagram shows a container window on elements 2 to 7 of a container that holds (for simplicity of illustration) 11 elements:



A container window

When the end-user scrolls down to the end of the displayed grid, the client changes the range of the window to the last 6 container elements (numbers 5 to 10) and passes this information to the Liberator as a container structure update request. (For example, a client using **StreamLink for Browsers** to communicate with the Liberator would send the container structure update request by calling the `setContainerWindow()` method on an instance of `SL4B_AbstractRttpProvider`.)

The Liberator stops sending client updates for the instruments that are no longer in the window (elements 2, 3, and 4), but continues to send updates for the elements that have remained in the window (elements 5, 6, and 7). It subscribes to the instruments that are newly in the scope of the window (elements 8, 9, and 10), sending the client the initial images and subsequent updates for them.



Moving the container window

When a large number of instruments need to be managed, the benefits of using containers with windowing can be very significant.

2.3 When to use a container

Use containers to hold lists of items when:

- The items that qualify for inclusion in a list are dynamically determined by a system that is external to Caplin Xaqua, and the items can be obtained using a suitable DataSource adapter.

  AND

- Your Caplin Xaqua client needs to display a long list of items in a window.

  The container windowing 7 capability is ideal for managing large lists in this way. When using a windowed container, the client does not need to locally manage the complete list, and so does not waste resources in receiving and processing updates to the large number of items that are not in the container window.

Containers can also be used when your Caplin Xaqua client needs to filter and/or sort the contents of a list based on fields that do not update. The filtering and sorting capabilities of Caplin Refiner are ideal for this purpose. See Filtering and sorting containers using Caplin Refiner 23.

2.4 Summary of container features and benefits

Caplin Xaqua containers support the following features:

- Windowing
  (see Container windowing 7).
- Filtering of container elements using Caplin Refiner
  (see Filtering and sorting containers using Caplin Refiner 23).
- Sorting of container elements using Caplin Refiner
  (see Filtering and sorting containers using Caplin Refiner 23).
- Grouping of container elements using Caplin Refiner
  (see Grouping 36 in Filtering and sorting containers using Caplin Refiner 23).
- Container snapshots:
  Liberator can supply an image of a container's contents in **CSV** format, for export to spreadsheets and other data analysis software.

Other features are:

- Changes to individual items in the container are managed by a **DataSource application**.
  The updates are automatically fed to the client in the same way as if the client had explicitly subscribed to each item individually.
- The source of a container can be independent of the source of the elements in the container.
  (see Example: Multiple DataSource adapters 14).

Benefits

- **Auto subscription:**
  Once the client has subscribed to a container, the subjects referred to by the container's elements are automatically subscribed to as well.
- **Dynamic list management:**
  The dynamic list of container elements is managed on the server side.
- **Windowing benefits:**
  - The client does not have to manage large lists of instruments (or other items) when most of them are not seen by the end-user at any one time.
  - The amount of data being processed through Caplin Xaqua is minimized; auto subscription means that Liberator only subscribes to the items in the window.
  - The number of round-trips between client and Liberator can be very much reduced, because Liberator only needs to send images and updates for a subset of the elements in the container.
  - Client performance is improved (for the same reasons as above), so the client application is less likely to perform badly or lock up.
- **Filtering, sorting, and grouping:** Using **Caplin Refiner**, lists of instruments can easily be filtered, sorted, and grouped together, without needing to implement filtering and sorting algorithms in client applications.
- **No size limits:** There is no limit (other than system resources) on the size of a container from the client perspective.

> **Tip:** Some implementations of StreamLink provide an API for a record filtering capability that runs in Caplin Liberator and does not use the container mechanism or Caplin Refiner. This facility pre-dates Caplin Refiner and is not as flexible. It filters updates to records (including records in containers), but unlike Caplin Refiner, it does not add/remove container elements according to whether or not they match the filter.
>
> For more information, see the **StreamLink 5.0 Overview**.

2.5 Containers in the Caplin Xaqua architecture

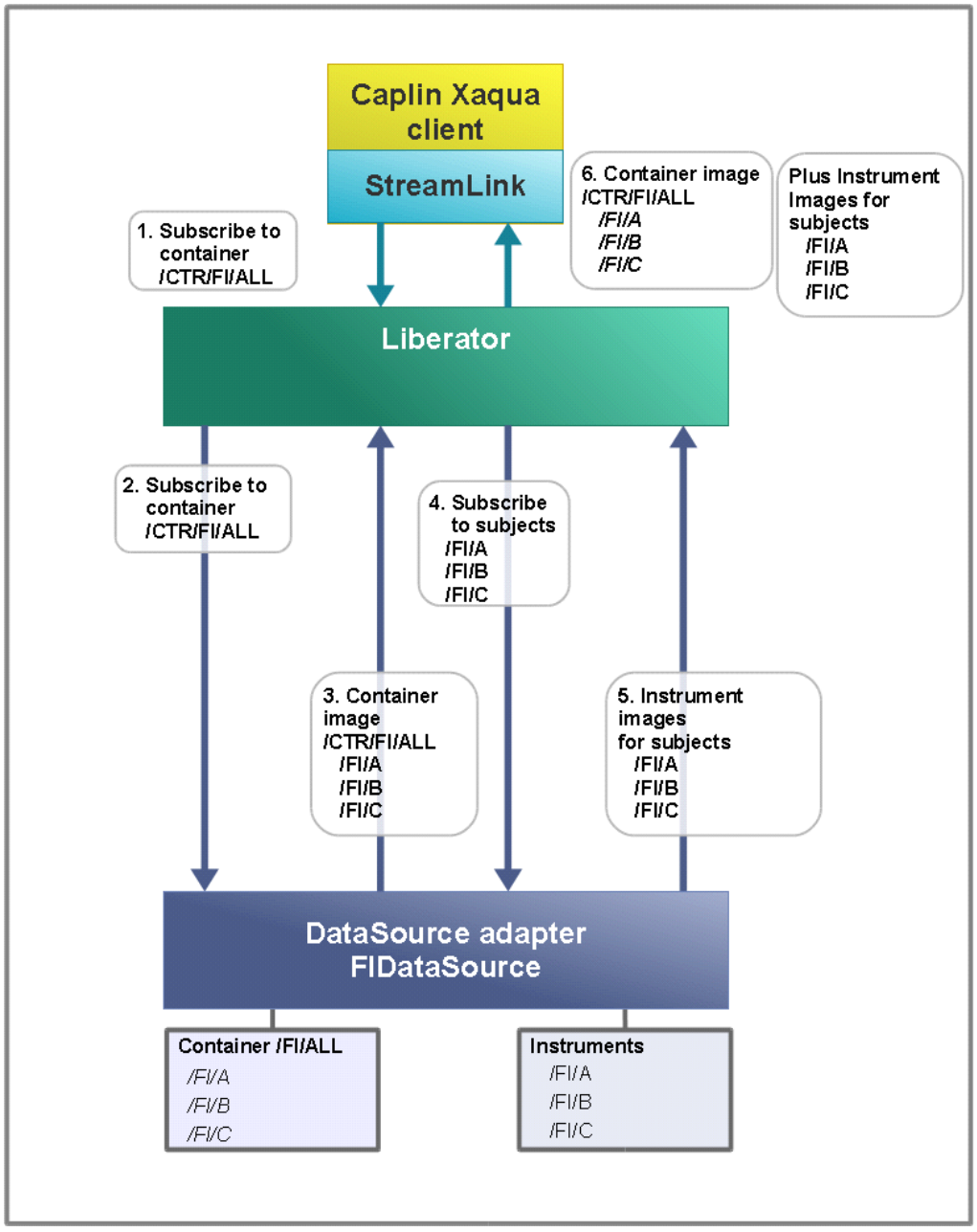
Several Caplin Xaqua components interact to provide container functionality:

- **Caplin Xaqua clients** (applications) request container subscriptions using a StreamLink API.
- **Caplin Liberator** manages containers on behalf of Caplin Xaqua client applications.
- The container elements are defined by other DataSource applications – typically **DataSource adapters** or a **Caplin Transformer**. These can be different DataSource adapters to the ones that supply the data referenced by the container elements.
- Transformer manages filtering, sorting, and grouping, through its **Caplin Refiner** module. For details refer to Filtering and sorting containers using Caplin Refiner 23.

The following sections give some examples of how container functionality is implemented in Caplin Xaqua.

Example: One DataSource adapter

The following diagram shows the flow of data and processing within Caplin Xaqua components when a client subscribes to a container. This is the simplest possible example, involving a **Caplin Xaqua client** application (with StreamLink), a Caplin Liberator, and a DataSource adapter that supplies both the container elements and the data that the elements refer to.



Container Architecture simple case

1. The Caplin Xaqua client subscribes, through StreamLink, to the container `/CTR/FI/ALL`, and this request is sent to the Liberator.
2. The Liberator does not currently have a subscription to this container in its cache, so it looks in its configuration for a data service that can supply the container elements.

   In this simple example, the FI container elements for `/CTR/FI/ALL` and the data that the elements refer to (the FI instruments) are all obtained from a single DataSource adapter called “FIDataSource” . (For the details of how the Liberator is configured to select this particular DataSource adapter, see One DataSource supplies the container and container data 18 in Configuring container usage in Liberator 17.)
3. FIDataSource receives the request for `/CTR/FI/ALL`, and recognizes from the `/CTR` prefix to the subject name that this is a request for a container. Since this particular container is not in its cache (not previously requested since the DataSource adapter was started up), FIDataSource obtains the list of elements (subjects) that this particular container refers to, and populates the container with the subject names – `/FI/A`, `/FI/B`, `/FI/C`, and so on.

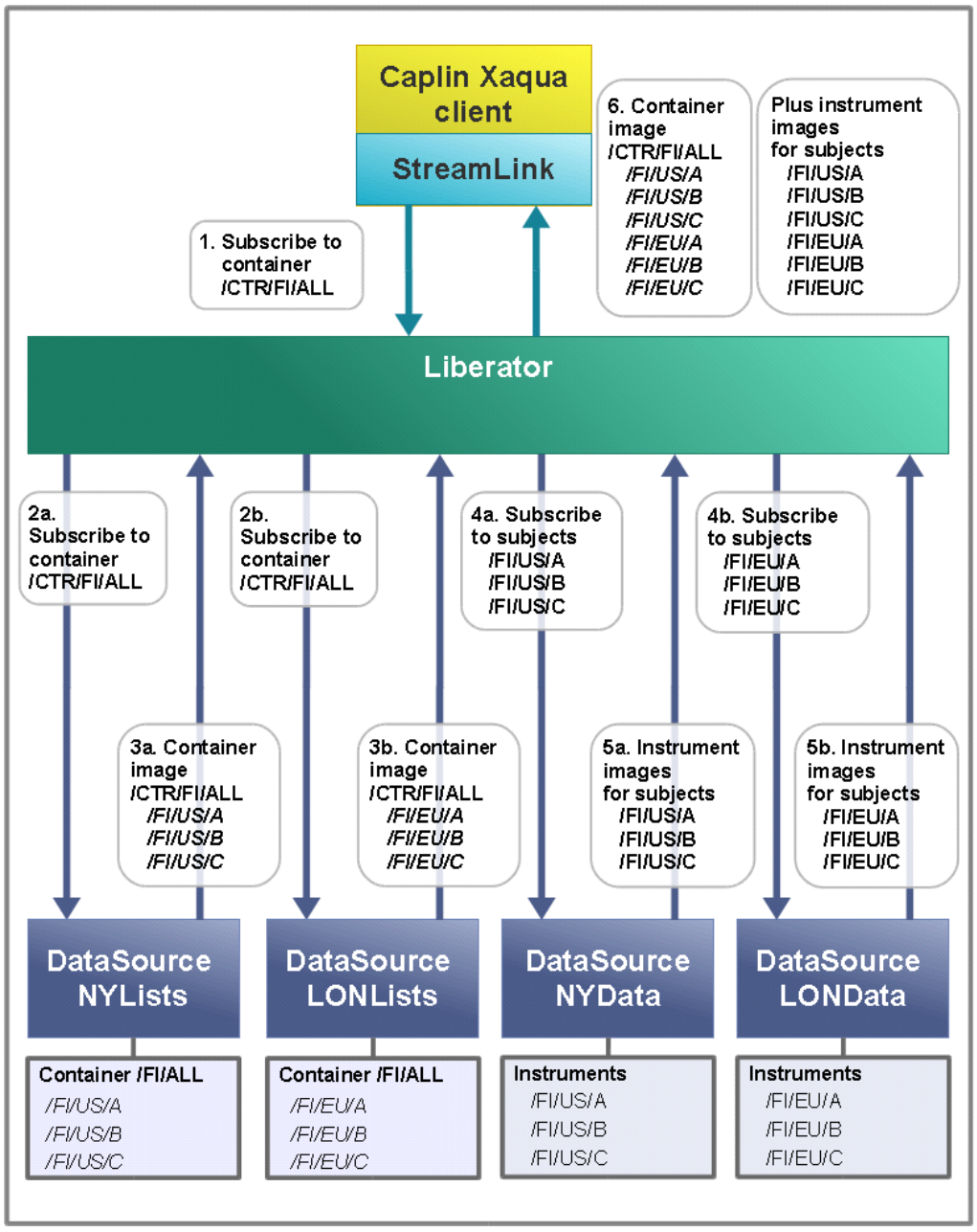
   The DataSource adapter returns the container elements to Liberator, where they are cached.
4. Liberator examines the container contents and subscribes to the subject of each container element that is not already in its cache. (In this simple example, the container is not windowed, so all its elements must be subscribed to.) For each unsubscribed subject it looks in its configuration for the data service that can supply the data. In this example, the data service is defined to use the same DataSource adapter FIDataSource as that which supplies the container itself. (For the details of the required configuration, see One DataSource supplies the container and container data 18 in Configuring container usage in Liberator 17.)
5. FIDataSource receives the requests for the subjects `/FI/A`, `/FI/B`, `/FI/C`, and so on. It returns the data images for the corresponding instruments, either from its cache (cached when the instrument has previously been subscribed to since the DataSource adapter was started up), or, if it is not already in the cache, by requesting the data from the relevant Bank system or external data feed. Liberator receives the data images for the subscribed container elements and caches them.
6. Liberator sends the client the container image, and the data images of all the instruments referred to by the container elements.
7. (This step is not shown in the diagram.) The Liberator sends the client all subsequent updates to the subscriptions. These can be:
   - Changes to the container structure; for example, new elements appearing in the container.
   - Updates to the subjects that the container elements refer to; for example, a change to the `Yield` field of instrument `/FI/C`.

Example: Multiple DataSource adapters

More complex setups are possible. For example, the contents of a container can be defined by more than one DataSource application, and the data that the elements refer to can be supplied by other DataSource applications, thus decoupling the source of the data from the source of the container. Such decoupling makes it easier to reconfigure a deployed Caplin Xaqua system to add additional sources of container elements and additional sources of data.

The following diagram shows the flow of data and processing within Caplin Xaqua components when a client subscribes to a container and there are several DataSources adapters supplying the data. In this example, the data about Fixed Income instruments is sourced from both New York and London, but must be made available to all end-users through a container called `/CTR/FI/ALL`. So the FI container and the elements (FI instruments) that it refers to are all obtained from four DataSource adapters:

- DataSource NYLists, located in New York, supplies the container elements for US instruments.
- DataSource NYData, located in New York, supplies the data for the US instruments.
- DataSource LONLists, located in London, supplies the container elements for European instruments.
- DataSource LONData, located in London, supplies the data for the European instruments.



Container Architecture multiple DataSource adapters

1. The Caplin Xaqua client subscribes, through StreamLink, to the container `/CTR/FI/ALL`, and this request is sent to the Liberator.

   The Liberator does not currently have a subscription to this container in its cache, so it looks in its configuration for a data service that can supply the container elements. The data service is configured so that the Liberator sends the subscription request for `/CTR/FI/ALL` to both the NYLists and LONLists DataSources. (For the details of how the Liberator is configured to select these particular DataSource adapters, see Multiple DataSources supply the container and container data 19 in Configuring container usage in Liberator 17.)

2a. The DataSource adapter NYLists receives the request for `/CTR/FI/ALL`. Since this particular container is not in its cache (not previously requested since the DataSource adapter was started up), NYLists obtains the list of elements (subjects) that this particular container refers to, and populates the container with just the subject names that it knows about – the US ones, `/FI/US/A`, `/FI/US/B`, `/FI/US/C`, and so on.

2b. The DataSource adapter LONLists also receives the request for `/CTR/FI/ALL`. As for NYLists, this particular container is not in its cache, so LONLists obtains the list of elements (subjects) that this particular container refers to, and populates the container with just the subject names that it knows about – the European ones, `/FI/EU/A`, `/FI/EU/B`, `/FI/EU/C`, and so on.

In reality, the requests for `/CTR/FI/ALL` are sent to both DataSource adapters at the same time, so the **natural order** of the elements in the returned container depends on which DataSource Adapter responds first.

3a. The NYLists DataSource returns the container elements for the US instruments to Liberator, where they are cached.

3b. The LONLists DataSource returns the container elements for the European instruments to Liberator, where they are cached.

4a/b. Liberator examines the container contents and subscribes to the subject of each container element that is not already in its cache. For each unsubscribed subject it looks in its configuration for data services that can supply the data.

In contrast to the example of one DataSource adapter 12, there are two data services defined for obtaining the instrument data, which result in the Liberator obtaining the data for subjects of the form `/FI/US/` from the New York DataSource adapter NYData, and data for subjects of the form `/FI/EU/` from the London DataSource adapter LONData. (For the details of how the Liberator is configured to select these particular DataSource adapters, see Multiple DataSources supply the container and container data 19 in Configuring container usage in Liberator 17.)

5a. DataSource NYData receives the requests for the US instruments (subjects `/FI/US/A`, `/FI/US/B`, `/FI/US/C`, and so on). It returns the data images for the corresponding instruments, either from its cache (cached when the instrument has previously been subscribed to since the DataSource adapter was started up), or by requesting the data from the relevant Bank system or external data feed if it is not already in the cache. Liberator receives the data images for the subscribed container elements and caches them.

5b. DataSource LONData receives the requests for the European instruments (subjects `/FI/EU/A`, `/FI/EU/B`, `/FI/EU/C`, and so on). It returns the data images for the corresponding instruments, in the same way as the NYData DataSource, and Liberator receives the data images and caches them.

6. Liberator sends the client the container image, and the data images of all the instruments referred to by the container elements.

7. (This step is not shown in the diagram.) The Liberator sends the client all subsequent updates to the subscriptions, as in the example of one DataSource adapter 12.

Also see DataSource adapter design guidelines 21.

3 Defining and using containers

To support container subscriptions, a number of code and configuration changes must be made to Caplin Xaqua components. Once this has been done, client code can use the relevant StreamLink API to subscribe to containers, receive updates to their structure and content, and define and manipulate container windows.

Tip: The exact way in which your Caplin Xaqua client application should subscribe to and manipulate containers depends on which StreamLink library you are using. For details, consult the API Reference document for your particular StreamLink implementation.

Note: Avoid implementing containers with very large numbers of elements (typically more than 50,000). Although Liberator is optimized for performance and uses multiple execution threads, intensive client access to containers of this size can increase the latency of data updates.

3.1 Configuring container usage in Liberator

When a Caplin Xaqua client requests a container, the Liberator connected to it receives the subscription request and must determine which DataSource application can provide the container, so it can pass the request on to that DataSource. This selection is determined through configuration, as shown in the following examples.

One DataSource supplies the container and container data

Here is the Liberator configuration that would route requests for the container `/CTR/FI/ALL`, as shown in the One DataSource adapter 12 example of Containers in the Caplin Xaqua architecture 11.

In this simple example, the FI container elements and data that the elements refer to (the FI instruments) are all obtained from a single DataSource adapter called “FIDataSource”.
The corresponding Liberator configuration looks like this:

```
add-peer
        remote-id               1
        remote-name             FIDataSource
        label                   FIDataSource
end-peer

add-data-service
        service-name            ContainersFI
        include-pattern         ^/CTR/FI/
        add-source-group
                required        true
                add-priority
                        label   FIDataSource
                end-priority
        end-source-group
end-data-service

        service-name            InstrumentsFI
        include-pattern         ^/FI/
        add-source-group
                required        true
                add-priority
                        label   FIDataSource
                end-priority
        end-source-group
end-data-service
```

The container subject `/CTR/FI/ALL` matches the include-pattern `^/CTR/FI/`, so the required data service is `ContainersFI`. This service definition has a single `add-priority` group with the label `FIDataSource`; which directs the Liberator to the DataSource called `FIDataSource`, defined in the `add-peer` group with the `remote-id` of `1`. The Liberator therefore sends a subscription request for the container `/CTR/FI/ALL` to the DataSource called `FIDataSource`.

Considering the container element `/FI/A`, and referring to the above configuration, the subject name `/FI/A` matches the include-pattern `^/FI/`, so the data service that can supply the data for the subject `/FI/A` is `InstrumentsFI`. This service definition has a single `add-priority` group with the label `FIDataSource`; which also directs the Liberator to the DataSource called `FIDataSource`.

Multiple DataSources supply the container and container data

In the Multiple DataSource adapters 14 example of Containers in the Caplin Xaqua architecture 11, *two* DataSource adapters provide the elements for a single container `/CTR/FI/ALL`, and two other DataSource adapters supply the data for the instruments in the container:

- DataSource NYLists, located in New York, supplies the container elements for US instruments.
- DataSource LONLists, located in London, supplies the container elements for European instruments.
- DataSource NYData, located in New York, supplies the data for the US instruments.
- DataSource LONData, located in London, supplies the data for the European instruments.

In this more complex example, the Liberator configuration that routes subscription requests for `/CTR/FI/ALL` to these DataSource adapters looks like this:

Liberator configuration for NYLists and LONLists

```
add-peer
        remote-id               1
        remote-name             NYLists
        label                   NYLists
end-peer

add-peer
        remote-id               2
        remote-name             LONLists
        label                   LONLists
end-peer

add-data-service
        service-name            ContainersFI
        include-pattern         ^/CTR/FI/
        add-source-group
                required        true
                add-priority
                        label   NYLists
                end-priority
        end-source-group
        add-source-group
                required        true
                add-priority
                        label   LONLists
                end-priority
        end-source-group
end-data-service
```

As in the configuration where one DataSource supplies the container and container data 18, the container subject `/CTR/FI/ALL` matches the include-pattern `^/CTR/FI/`, so the required data service is again `ContainersFI`. However, this service definition has *two* `add-source-group` entries, one specifying the NYLists DataSource, and the other specifying the LONLists DataSource. So the Liberator sends the subscription request for `/CTR/FI/ALL` to *both* DataSources.

Liberator configuration for NYData and LONData

```
add-peer
        remote-id               3
        remote-name             NYData
        label                   NYData
end-peer

add-peer
        remote-id               4
        remote-name             LONData
        label                   LONData
end-peer

add-data-service
        service-name            InstrumentsFIUS
        include-pattern         ^/FI/US/
        add-source-group
                required        true
                add-priority
                        label   NYData
                end-priority
        end-source-group
end-data-service

add-data-service
        service-name            InstrumentsFIEU
        include-pattern         ^/FI/EU/
        add-source-group
                required        true
                add-priority
                        label   LONData
                end-priority
        end-source-group
end-data-service
```

In contrast to the configuration where one DataSource supplies the container and container data 18, there are two data services defined for obtaining the instrument data. The subject name `/FI/US/A` matches the include-pattern `^/FI/US/` so the data service for this subject is `InstrumentsFIUS` and its data is therefore obtained from the New York DataSource adapter NYData. The subject name `/FI/EU/A` matches the include-pattern `^/FI/EU/` so the data service for this subject is `InstrumentsFIEU` and its data is therefore obtained from the London DataSource adapter LONData.

3.2 Mapping containers

The name of a container can be mapped within Liberator in the same way as other subject names.

For example, a client request for the container `/CTR/FI/ALL` could be mapped to a request for `/CTR/FI/ALL/<tier>`, where `<tier>` is the price tier for the requesting user. When the mapped request is passed to the DataSource application that supplies the container, the DataSource can respond by populating the container with just the instruments that can be traded within the indicated price tier.

In this particular case, the mapping would typically be done by a **Liberator Auth module** that determines users' price tiers. Simple mappings can be configured in Liberator using the **object-map** configuration item (for more information see the **Liberator Administration Guide**).

3.3 Defining containers in a DataSource

A container is defined (created) in a DataSource application. This is typically a DataSource adapter that interfaces to a bank system, or other external data feed, that supplies the list of items that go in the container. For example, the external system could provide a list of Fixed Income instruments that can be traded.

Container operations in DataSource APIs

The DataSource APIs allow you to perform various operations on containers.

You can:

- Add an element to a container. The element is appended to the end of the container.
- Insert an element into a container at a specified position.
- Remove an existing element from a container.
- Remove ("clear down") all elements from a container that match a specified subject prefix.

> **Tip:** The exact way in which your DataSource application should subscribe to and manipulate containers depends on which DataSource SDK you are using. For details, consult the API Reference document for your particular DataSource SDK.

DataSource adapter design guidelines

When coding DataSource adapters that supply containers, apply the following design guidelines as appropriate.

Responding to container subscription requests

When the DataSource adapter receives a subscription request for a container, it can be better to send the container elements back to Liberator / Transformer as an update, rather than as an initial image (as is usually the case when a DataSource responds to a subscription request).

Where you have more than one DataSource adapter supplying the container elements, the DataSources would normally be unaware of each other's role. When the container was first subscribed to, if the DataSources were to send the container elements as an image ("image" flag set on the DataSource message), the later images would overwrite the earlier ones.

For example, referring to the example with multiple DataSource adapters 14, DataSource NYLists sends Liberator a container image with elements `/FI/US/A`, `/FI/US/B`, and `/FI/US/C`. Then DataSource LONLists, which is unaware of NYLists, sends Liberator a container image with elements `/FI/EU/A`, `/FI/EU/B`, and `/FI/EU/C`. These elements would overwrite the elements `/FI/US/A`, `/FI/US/B` and `/FI/US/C` in the container, because receipt of an image implicitly causes a clear down of the container.

To avoid this problem, send the container elements as an *update*; DataSource LONLists then *appends* elements `/FI/EU/A`, `/FI/EU/B`, `/FI/EU/C` to the existing elements from NYLists `/FI/US/A`, `/FI/US/B`, `/FI/US/C`.

Forcing container clear down

In order to recover properly from error situations (lost connections, and so on), a DataSource adapter may need to clear down (empty) any containers for which it supplies elements. The obvious way to do this is just send the Liberator / Transformer an empty image of the container. However, this simple approach will not work when multiple DataSources populate the same container.

Referring to the example with multiple DataSource adapters 14, when DataSource NYLists needs to clear down the container, rather than sending an empty image of the container, it should send the client a container "clear down" response for `/FI/US`. This forces a clear down of just the container elements that particular DataSource is responsible for (`/FI/US/A`, `/FI/US/B`, `/FI/US/C` ...). Similarly, DataSource LONLists should a clear down response for `/FI/EU`, which forces a clear down of just the `/FI/EU/A`, `/FI/EU/B`, `/FI/EU/C` ... elements in the container.

4 Filtering and sorting containers using Caplin Refiner

The contents of a container can be filtered and sorted according to criteria supplied by the client application. Clients can specify both a filter and a sort in the same request. The filtering and sorting are done by the **Caplin Refiner** module of Caplin Xaqua's **Transformer** component.

Caplin Refiner can also group container records together, with extra records added to the container that act as headers for the blocks of grouped records (see Grouping 36).

Note: Avoid filtering or sorting on record fields that are subject to updates.
Frequent updates can have an adverse affect on the performance of Caplin Refiner, Liberator, and the requesting clients.

Examples

The bank may offer a wide range of bonds for customers to trade. The list of bonds is supplied in a container for the trading client application to display in a scrollable grid. However, an end-user is typically only interested in a subset of the available bonds, so they can apply a filter via the UI to the field values of the records within the container. Only information about the bonds selected by the filter is sent back to the client application.

Examples of filters are:

- All the bonds that mature in the forthcoming 2 year period.
- All the bonds with a coupon greater than a particular value.
- All the bonds with a rating greater than a particular value.

The end-user will usually want the filtered list to be sorted as well; for example, in descending order of the coupon.

Example filter

Assume that without filtering, the `/CTR/FI/ALL` container has the following six records:

Container /CTR/FI/ALL

Subject	Field `Description`	Field `CpnRate`	Field `MaturityDate`	Field `BidPrice`	Field `AskPrice`
/FI/US/A	US TREASURY	11.25	20141115	116 121/256	116 249/256
/FI/US/B	US TREASURY	11.75	20150215	150 117/256	150 127/256
/FI/US/C	US TREASURY	10.62	20150815	148 196/256	148 206/256
/FI/EU/A	AUSTRIA	4.35	20140715	103.948	103.984
/FI/EU/B	AUSTRIA	6.25	20270715	122.890	122.974
/FI/EU/C	GERMANY	4.92	20130328	101.859	101.858

Note that the `MaturityDate` field is in the format `YYYYMMDD`; for example the maturity date of `/FI/US/A` is `20141115`, meaning “15 Nov 2014”. This allows the records to be filtered and sorted on maturity date without needing to implement custom filter comparators 45 and custom sort comparators 42.

Typically some fields are static, meaning they have a set value that does not change, such as the `Description`, `CouponRate`, and `MaturityDate` fields. Other fields are dynamic, meaning they can change value. In this example, the values of the `BidPrice` and `AskPrice` fields can change throughout the day, perhaps as often as several times a second.

When the client application subscribes to this container, the application is automatically subscribed to the constituent records and receives updates for them each time any field values change.

Assuming the values of the `CpnRate` fields remain unchanged for the time being, a request for the `/CTR/FI/ALL` container with a filter of `"CpnRate>11"` would return a container with only two of the six records:

Container /CTR/FI/ALL filtered by `CpnRate>11`

Subject	Field `Description`	Field `CpnRate`	Field `MaturityDate`	Field `BidPrice`	Field `AskPrice`
/FI/US/A	US TREASURY	11.25	20141115	116 121/256	116 249/256
/FI/US/B	US TREASURY	11.75	20150215	150 117/256	150 127/256

In this example, `/FI/US/C`, `/FI/EU/A`, `FI/EU/B`, and `/FI/EU/C` have been removed from the container because the values of their `CpnRate` fields do not match the filter criteria.

Example sort

Clients can specify sort criteria to change the order of the records within a container.

For example, a client might request the `/CTR/FI/ALL` container with a sort on the numeric field called `CpnRate` in descending order. The result of the sort would look like this:

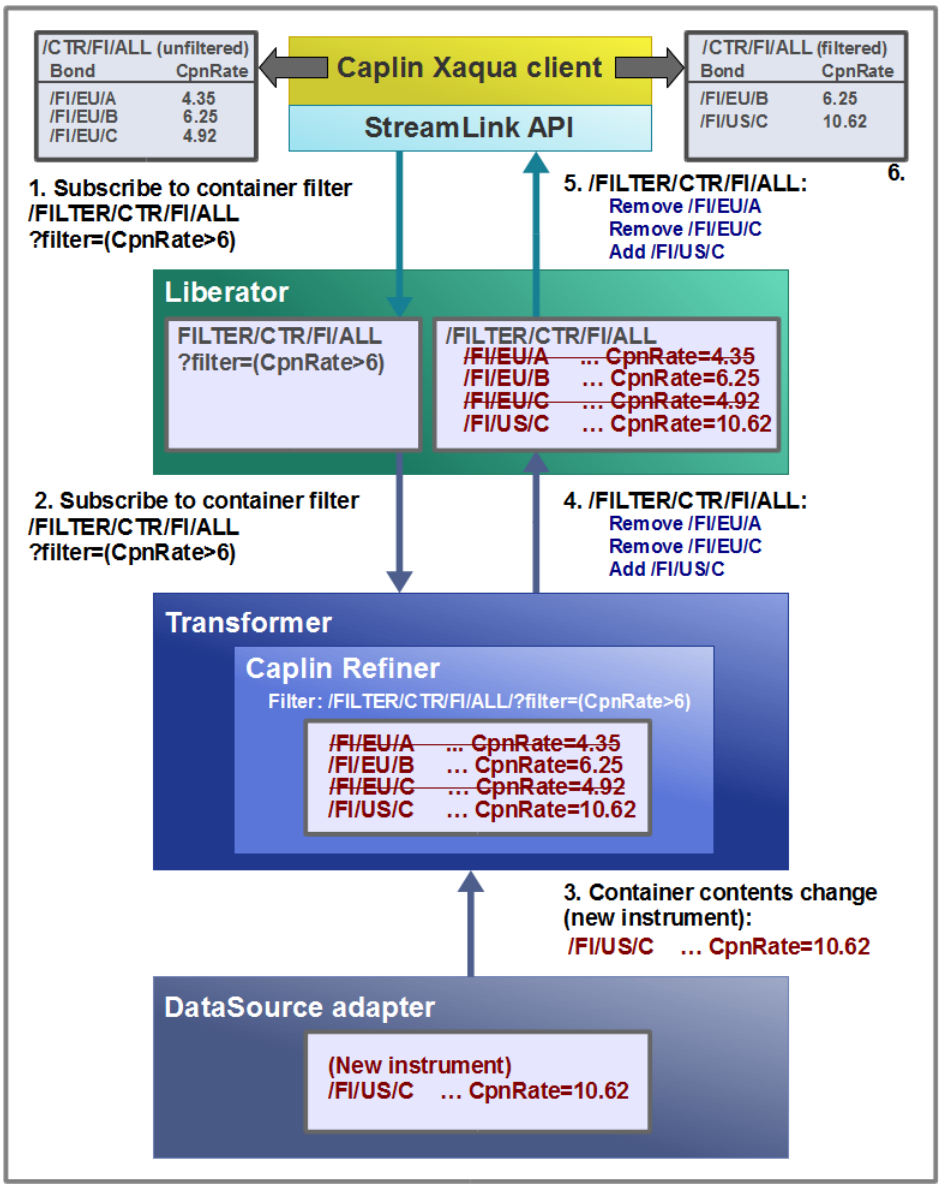
Container /CTR/FI/ALL sorted by descending CpnRate

Subject	Field `Description`	Field `CpnRate`	Field `MaturityDate`	Field `BidPrice`	Field `AskPrice`
/FI/US/B	US TREASURY	**11.75**	20150215	150 117/256	150 127/256
/FI/US/A	US TREASURY	**11.25**	20141115	116 121/256	116 249/256
/FI/US/C	US TREASURY	**10.62**	20150815	148 196/256	148 206/256
/FI/EU/B	AUSTRIA	**6.25**	20270715	122.890	122.974
/FI/EU/C	GERMANY	**4.92**	20130328	101.859	101.858
/FI/EU/A	AUSTRIA	**4.35**	20140715	103.948	103.984

The following sections show how these features can be used by client applications.
Also see Using advanced features of Caplin Refiner 41

4.1 How container filtering works

The following diagram illustrates how container filtering works. It shows the Caplin Xaqua components involved in filtering the contents of a container, and the flow of subscriptions and data through the system. The actual filtering and sorting of the container is performed by the Transformer's **Caplin Refiner** module, which is a **Java Transformer Module**. In this example, the Caplin Xaqua client application has already subscribed to the container `/CTR/FI/ALL`, and the instruments referred to in the container are displayed to the end-user in a grid (see the gray box at the top left of the diagram). Subsequently the end-user sets up a filter that restricts the instruments displayed in the container to those where the Coupon (`CpnRate` field) is greater than 6. For simplicity, the container is not windowed in this example.



Example of container filtering

1. The Caplin Xaqua client calls the StreamLink Container Filtering API to add the filter `(CpnRate>6)` to the subscription for the container `/CTR/FI/ALL`. Liberator receives the filter as a request for the subject `/FILTER/CTR/FI/ALL?(CpnRate>6)`.
2. Liberator passes the container filter subscription request on to Transformer, where it is retained by Caplin Refiner, which sets up the filter. (If the **base container –** in this case `/CTR/FI/ALL` **–** is not in the Transformer's cache, Transformer requests it from the relevant DataSource adapter.)
3. At the same time, the DataSource adapter that handles the subscriptions to the instruments in the container passes back to Transformer a new container element, referring to a new instrument `FI/US/C`.
4. In Transformer, Caplin Refiner applies the filter to the container according to the latest set of instruments. In this example, the value of the `CpnRate` field for `/FI/EU/A` and `/FI/EU/C` is less than `6`, so these instruments no longer meet the filter criteria, and must be removed from the filtered container. The new instrument /FI/US/C, with its `CpnRate` of `10.62`, matches the filter, so it is retained in the filtered container.

   Transformer sends a container structure change message to Liberator, specifying that the subject names `/FI/EU/A` and `/FI/EU/C` be removed from the container, and the subject name `/FI/US/C` be added to the container.
5. Liberator makes the changes to the container and passes the corresponding container structure change messages on to StreamLink. StreamLink then informs the **Caplin Xaqua client** about the change to the filtered container so that it can remove the instruments `/FI/EU/A` and `/FI/EU/C` from the filtered grid on the display, and add `/FI/US/C`.
6. Any updates to the instruments `/FI/EU/B`, and the image for the new instrument `/FI/US/C`, are also fed back to the client, by the usual Caplin Xaqua mechanism for propagating such updates via Liberator and StreamLink. The client updates the instruments in the filtered grid – see the gray box at the top right of the diagram.

This example shows how the filter criteria are applied when the filter is first defined, and how they are also dynamically applied to the container as changes are received to the instruments in the container.

4.2 Configuring container filtering

These sections explain how to set up Caplin Xaqua to enable container filtering using Caplin Refiner.

In summary:

- Install the Transformer's Caplin Refiner 26.
- Configure the Liberator 29.
- Configure Caplin Refiner 29 (optional).

Installing Caplin Refiner

1. Ensure the Transformer is licensed to use the Java Transformer Module.

The Transformer license file is in *<Transformer root>/etc/license-transformer.conf* and should contain the line:

```
module  jtm  transformer
```

If the license does not contain this line, please ***contact Caplin Support****.*

2. Verify that the Transformer is set up to use Java.

   There should be lines in
   <Transformer root>/etc/transformer.conf
   or in
   <Transformer root>/etc/java.conf
   that look like this:

```
# Set jvm-location as a fully qualified pathname to the preferred JVM
#
jvm-location <JVM-pathname>
```

   For example:

```
jvm-location /usr/local/java/jre/lib/i386/libjvm.so
```

3. Verify that the *transformermodule.jar* is available and configured on the classpath.

   There should be a line in
   <Transformer root>/etc/transformer.conf
   or in
   <Transformer root>/etc/java.conf
   that looks like this:

```
jvm-global-classpath %r/lib/java/transformermodule.jar
```

4. Caplin Refiner is deployed in the same way as all other Transformer Modules: put the file *container-filtering-module.jar* on the server that hosts Transformer.

   Typically it is placed in the directory

   <Transformer root>/lib-ext

5. Add the following configuration to
   <Transformer root>/etc/transformer.conf or *<Transformer root>/etc/java.conf*:

```
add-javaclass
   class-name com.caplin.transformer.refiner.Refiner
   class-id   jtm
   classpath  %r/lib-ext/refiner.jar
end-javaclass
```

6. Add a file *<Transformer root>/etc/java.conf* with the following content:

```
log-level     INFO
module-logfile  jtm.log
module-classid  jtm
```

7. Start Transformer.

- Navigate to the Transformer root directory.
- Run the command:

```
./etc/start transformer
```

8. Examine the log file *<Transformer root>/var/jtm.log*

If the Transformer is working correctly, this log contains the version number of the Transformer Module library:

```
2011/09/12-15:41:05.633 +0000: CRIT: Thread [main/1]:
Product: Java Transformer Module
Version         : 5.0.1
Build Date      : dd-Mon-yyyy
Build Time      : hh:mm
Build Number    : nnnnnn
Copyright       : Copyright 1995-2011 Caplin Systems Ltd
```

In the same log file, you should also see Caplin Refiner registering as a provider for the namespace it uses:

```
2011/12/16-15:41:05.703 +0000: INFO: Registering as a provider of </FILTER/*>
```

9. Examine the log file *Transformer root>/var/refiner.log*

If Caplin Refiner is working correctly, this log contains its version number:

```
[SEVERE|main|15:42:38.903]: Refiner 5.0.0-204042
```

Configuring filtering in Liberator

Liberator must be configured to support container filtering.

Assuming Liberator and Transformer are correctly configured to connect to each other, there will already be a section in the Liberator configuration file (*rttpd.conf*) that defines the **data service** for Transformer.

- Add the `^/FILTER/` pattern to the Liberator's Transformer data service, as in the following example:

```
add-data-service
   service-name      market-data
   include-pattern   ^/FI/
   include-pattern   ^/CONTAINER/
   include-pattern   ^/FILTER/

   add-source-group
       required true
       add-priority
           label transformer
       end-priority
   end-source-group
end-data-service
```

  This pattern ensures that when client applications make requests for subjects that begin with `/FILTER`, the Liberator requests them from Transformer and the requests are handled by the Transformer's Caplin Refiner module.

Configuring Caplin Refiner

Transformer's Caplin Refiner module has default configuration settings that allow it to work without further configuration. However, you can modify the configuration properties if required (for example, to improve filtering and sorting performance). To do this:

- Create an empty configuration properties file:

  *<Transformer root>/etc/**refiner.properties***

- Add configuration property values to the file from the following set, as required.

  The format is:

  `property-name=value`

Configuration properties

The following configuration properties are available in Caplin Refiner:

Property name	Default value	Description
`batch.time`	`0` milliseconds	The duration in milliseconds for which Caplin Refiner waits before publishing any changes it has received since the last time it published. The default value causes Caplin Refiner to immediately publish any new updates as soon as it has finished publishing the previous batch. If `batch.time` is negative or not a number, the default value is used. Run Caplin Refiner initially with the default setting for `batch.time` (no batching of updates). If in your system, the rate at which Caplin Refiner has to publish updates causes performance problems with Caplin Refiner itself or for clients, increase `batch.time` as required to give acceptable performance. For more information on tuning the performance of Caplin Refiner, see the document **Caplin Xaqua: Caplin Refiner Benchmarks**.

`filtering.algorithm.<filter-name>`	No default	Defines a custom filter comparator.
		
		`<filter-name>` is the name of the custom filter comparator.
		You can define multiple custom filter comparators; each `filtering.algorithm.<filter-name>` must have a unique `<filter-name>`.
		
		The property value is the fully qualified class name of the custom filter comparator class.
		For example:
		`filtering.algorithm.`**`BondRating`**`=`
		`com.example.BondRatingComparator`
		
		For more information, see Custom filter comparators 45.

Property name	Default value	Description
`log.file.count`	`1`	The number of Caplin Refiner log files to keep when `log.file.limit` is not `0`. Log files are cycled according to this count; when the number of log files matches the count, and the `log.file.limit` is reached, the current log file is closed and the oldest log file is reopened and overwritten with new log messages. Then the next oldest log file is reused, and so on. A value of `0` means there is no limit on the number of successive log files produced. The default value of `1`, causes Caplin Refiner to overwrite the current log file as soon as the `log.file.limit` is reached. If you set `log.file.limit` to a (positive) value other than `0`, set `log.file.count` to at least `2`, otherwise the contents of the current log file will be overwritten as soon as the `log.file.count` is reached.
`log.file.level`	`CONFIG`	Defines the log level for messages written to the Caplin Refiner log file. The log levels are the standard Java ones: ◆ `SEVERE` (highest level) ◆ `WARNING` ◆ `INFO` ◆ `CONFIG` ◆ `FINE` ◆ `FINER` ◆ `FINEST` (lowest level)
`log.file.limit`	`0`	The maximum size of a Caplin Refiner log file in bytes. When the current log file reaches this limit, it is closed and a new log file is started (see `log.file.count`). The default value of `0` means the same log file is always used. If you set `log.file.limit` to a (positive) value other than `0`, set `log.file.count` to at least `2`, otherwise the contents of the current log file will be overwritten as soon as the `log.file.count` is reached.

Property name	Default value	Description
`max.update.size`	`1000` updates	As Transformer receives updates to the records in a container, it creates (via Caplin Refiner) individual container record removal and record insertion messages according to how the updates match the filter and sort criteria. If the resulting number of record insertions is high, this can load the Liberator excessively. The `max.update.size` property helps to reduce such loading on the Liberator. It defines the maximum number of container record inserts that Caplin Refiner sends in a single batch period (as defined by `batch.time`). If the number of such updates exceeds `max.update.size`, the changes are sent as a single container image instead. If `max.update.size` is negative, zero or not a number, this feature is turned off and all updates are sent as inserts.
`nodata.records.rerequest.timer`	`60,000` milliseconds	The period between re-requests made by Caplin Refiner for any records that were not previously successfully subscribed to on a supplying DataSource. If this value is negative, `0`, or not a number, this feature is turned off.
`placeholder.timeout`	`1,000` milliseconds	Defines how long Caplin Refiner waits for results before removing the placeholder record. See The container placeholder 36.

`sorting.algorithm.<sort-name>`	No default	Defines a custom sort list or custom sort comparator.
		
		`<sort-name>` is the name of the custom sort.
		You can define multiple custom sort lists and sort comparators; each `sorting.algorithm.<sort-name>` must have a unique `<sort-name>`.
		
		For custom sort lists, the property value is a list of sort attributes of the form
		`list: <item1>,<item2>,...`
		in ascending sort order.
		For example:
		`sorting.algorithm.`**`TenorSort`**`=list:SPOT,ON,TN,SN,1W,2W,1M,2M,6M,1Y`
		For more information, see Custom sort lists 42.
		
		For custom sort comparators, the property value is of the form
		`class:<comparator-class>`
		where `<comparator-class>` is the fully qualified class name of the custom comparator class.
		For example:
		`sorting.algorithm.`**`BondRating`**`=class:com.example.BondRatingComparator`
		For more information see Custom sort comparators 42.

4.3 Using Caplin Refiner

To use Caplin Refiner, a client application subscribes to (requests) a subject, where the subject name indicates to the Transformer that a container is to be filtered and/or sorted. The Transformer passes the subscription request to its Caplin Refiner module, which then parses information from the subject in order to perform its functions.

The client does not create such subscription requests directly, but instead calls dedicated API methods to construct filter and sort directives, which are then passed on to Caplin Refiner.

- To use Caplin Refiner from a **Caplin Trader application**, call the Caplin Trader API methods on particular **display components** (such as **grids**) that allow you to filter and sort the data displayed. For details, see the document **Caplin Trader: API Reference**.
- To use Caplin Refiner from a client that is not a Caplin Trader application, call the API methods provided in the StreamLink library that your application uses. These methods allow you to build up the request in an object oriented way, and the request string itself is then generated by the StreamLink library.

> **Tip:** For a list of the StreamLink implementations have an API for using Caplin Refiner, see Appendix A: StreamLink support for container filtering 48.

4.4 Filtering rules

To filter container elements, you use API calls to build up a **filter expression** that is then sent to Caplin Refiner. A simple filter expression would be:

```
CpnRate>6
```

where:

- `CpnRate` is the record field to be filtered.
- `6` is the value to compare against.
- `>` is the operator (in this case “greater than”).

This filter expression therefore means: “Supply only those container records where the `CpnRate` field has a value greater than `6`.”

Filters can be applied to both text and numeric fields.

The following operators can be used in a filter expression:

Filter expression operators

Operator	Meaning	Applies to
=	equals (field value must match exactly)	text or numeric fields (usually used for text)
!=	not equals	text or numeric fields (usually used for text)
==	numeric equals	numeric fields only
!==	numeric not equals	numeric fields only
<	less than	numeric fields only
>	greater than	numeric fields only
<=	less than or equal to	numeric fields only
>=	greater than or equal to	numeric fields only
~	regular expression match	text or numeric, usually text
#	case insensitive regular expression match	text or numeric, usually text

Filters can contain any number of expressions, joined with `AND` or `OR`.
For example:

AND and OR in a filter expression

```
MaturityDate=20130328 OR (CpnRate>6 AND CpnRate<11)
```

The `MaturityDate` field is `"28 March 2013"` in the format `YYYDDMM`.

4.5 Sort rules

To sort container elements, you use API calls to build up a **sort rule** (the set of criteria for the sort) that is then sent to Caplin Refiner. Container sort rules consist of three parts:

- The name of the field to sort by.
- The type of the field, which can be `text`, `number`, or a custom type (see Custom sorting 41).
  Text sorts are case-insensitive.
- The sort sequence, either `ascending` or `descending`.

An example of a sort rule is:

```
sort=MaturityDate:number:ascending
```

This sort rule specifies that the filtered records are to be sorted in ascending numeric order of the `MaturityDate` field.

You must specify the type of the field so that Caplin Refiner can correctly sort the records in the intended order . For example, assume there are three records A, B, C to be sorted on a particular field, and this field has values `10`, `20`, and `100` respectively. If the field type is specified as numeric, an ascending sort returns the records in the following (numeric) order:

```
A: 10
B: 20
C: 100
```

However if the field type is specified as `text`, the records are sorted *alphabetically*. An ascending sort therefore returns the above records in a different order:

```
A: 10
C: 100
B: 20
```

Containers can only be sorted by one field, so it is not possible to apply a secondary sort. In cases where two or more records have the same value in the sort field, those records are sorted relatively into the **natural order** as defined by the DataSource adapter that supplies that container (or part of the container). This is the order in which the DataSource would return records with this field value to the client if no sort were specified.

Consider the container `/CTR/FI/ALL` discussed in Filtering and sorting containers using Caplin Refiner 23. Sorting this container by the `Description` text field results in the following ordering, where the order of the records within the same Description value is the natural order:

Container /CTR/FI/ALL sorted by Description

Subject	Field `Description`	Field `CpnRate`	Field `MaturityDate`	Field `BidPrice`	Field `AskPrice`
/FI/EU/A	AUSTRIA	4.35	20140715	103.948	103.984
/FI/EU/B	AUSTRIA	6.25	20270715	122.890	122.974
/FI/EU/C	GERMANY	4.92	20130328	101.859	101.858
/FI/US/A	US TREASURY	11.25	20141115	116 121/256	116 249/256
/FI/US/B	US TREASURY	11.75	20150215	150 117/256	150 127/256
/FI/US/C	US TREASURY	10.62	20150815	148 196/256	148 206/256

4.6 Grouping

Grouping adds extra records to the container that act as headers for blocks of records with the same field value. For example, referring to the FI container described in Filtering and sorting containers using Caplin Refiner 23, if you sort by the `Description` field and also group by the `Description` field, the returned container looks like this:

```
Container: /FILTER/FI/ALL?sort=Description:text:ascending&groupBy=Description

/FILTER/META/CTR/FI/ALL/GROUPHEADER/groupby=Description/AUSTRIA
SubHeaderText=AUSTRIA
/FI/EU/A Description=AUSTRIA CpnRate=4.35 MaturityDate=20140715 BidPrice=103.948 ...
/FI/EU/B Description=AUSTRIA CpnRate=6.25 MaturityDate=20270715 BidPrice=122.890 ...

/FILTER/META/CTR/FI/ALL/GROUPHEADER/groupby=Description/GERMANY
SubHeaderText=GERMANY
Description=GERMANY /FI/EU/C CpnRate=4.92 MaturityDate=20130328 BidPrice=101.859 ...

/FILTER/META/CTR/FI/ALL/GROUPHEADER/groupby=Description/US TREASURY
SubHeaderText=US TREASURY
/FI/US/A Description=US TREASURY CpnRate=11.25 MaturityDate=20141115 BidPrice=...
/FI/US/B Description=US TREASURY CpnRate=11.75 MaturityDate=20150215 BidPrice=...
/FI/US/C Description=US TREASURY CpnRate=10.62 MaturityDate=20150815 BidPrice=...
```

Each of the header records (`/FILTER/META/CTR/FI/ALL/GROUPHEADER/` and so on) contains one field, `SubHeaderText`, that contains the value of the field that you are grouping by. When the client application receives the sorted and grouped container, it typically displays the header rows in the grid with different styling to the data rows.

Tip: The exact way in which record grouping is added to a container subscription depends on which StreamLink library you are using. For details consult the API Reference document for your particular StreamLink implementation.

4.7 The container placeholder

The process of subscribing to a container, subscribing to all the records in it, and then Caplin Refiner processing each of the records can take longer than the active request timeout configured in Liberator. If Caplin Refiner were to wait until it had processed all the results before sending a response, the active request timeout might trigger first and Liberator would then discard the container.

To prevent this happening, as soon as Caplin Refiner receives a filter or sort request, it sends back a container response with a single special subject name in it: `"/FILTER/META/PLACEHOLDER"`. This is called the container placeholder record. It tells the Liberator that Caplin Refiner has received the request, preventing the container subscription from being discarded.

Handling container placeholders in the client

Typically a client application displays a loading indicator while waiting for a response to a sort or filter request, and it removes the loading indicator when a response is received. If the first response received is a container of size 1 that contains a placeholder record, the client should ignore the response and leave the loading indicator in place. The next container update received will either be a container image of the results, or an explicit removal of the placeholder record that indicates there are no matching results. The client should respond by removing the loading indicator.

Caplin Refiner explicitly removes the placeholder record when the time taken to filter the container exceeds the timeout defined in the configuration property `placeholder.timeout` (see Configuration properties 30 in Configuring Caplin Refiner 29). In this case, the client can still subsequently receive results from the filter request until such time as it discards the filter.

5 Permissions and subject mappings for filtered containers

In Caplin Xaqua deployments that use a Permissioning DataSource for user permissions and subject mappings, additional permissions and subject mappings need to be set up if Caplin Refiner is used to filter or sort container subscriptions.

5.1 Setting user permissions

When Caplin Refiner receives a request for a filtered or sorted container, it requests the base container from the providing DataSource application. For example, if Caplin Refiner receives a request for:

```
/FILTER/CTR/FI/ALL?filter=(CpnRate>5)
```

it requests the following base container from the providing DataSource application:

```
/CTR/FI/ALL
```

When the base container is returned, Caplin Refiner requests the constituent records of the container, and then applies the filter (in this example `CpnRate>5`) before responding to the container request.

This means that users must be permissioned for:

- The constituent records of the base container.
- The unfiltered container subject `/CTR/FI/ALL`.

  This permission is required for unfiltered container requests.
- The filtered container subject prefix `/FILTER/*`

  This permission is required for filtered container requests. The `/*` at the end of `/FILTER` means that users are permissioned for all filtered container subjects.

> **Tip:** In this example, users could be permissioned for the filtered container subject `/FILTER/CTR/FI/ALL` instead of the subject prefix `/FILTER/*`.
> The permissions are equivalent, allowing the same container to be filtered.

5.2 Setting subject mappings

Subject mappings allow Liberator to change the subject of a record before the record is requested from the providing DataSource. Subject mappings are typically used to support price tiering. For example, if the container `/CTR/FI/ALL` has the constituent record:

```
/FI/US/A
```

Liberator could map the subject of this record to:

```
/FI/US/A/TIER1
```

before it requests the record from the providing DataSource.

Price tiering allows users in different price tiers to receive different prices for the same instrument. In this example, the user receives the price for `/FI/US/A/TIER1` and not `/FI/US/A` when the container `/CTR/FI/ALL` is requested. A different user could receive the price for `/FI/US/A/TIER2`.

Subject mappings for the constituent records of a container are normally applied by the Auth Module at Liberator, but Caplin Refiner must also apply these mappings if it filters or sorts the container.

This means that the following subject mappings must be set up:

- The subject of the container to be filtered or sorted (including the filter or sort string) must be mapped to `<container-subject>;mapsuffix=<suffix-to-map>`.

  In this example, `<container-subject>` is `/FILTER/CTR/FI/ALL?filter=(CpnRate>5)` and `<suffix-to-map>` is `/TIER1`. The mapped subject is therefore:

  ```
  /FILTER/CTR/FI/ALL?filter=(CpnRate>5);mapsuffix=/TIER1.
  ```

  This subject mapping informs Caplin Refiner that it must map the subjects of constituent records to `/TIER1` before it requests the records from the providing DataSource.

- Subject mappings for the constituent records of the container.

  These mappings are applied at the Liberator Auth Module. In this example, the container has only one record (with subject `/FI/US/A`) that must be mapped to `/FI/US/A/TIER1`.

Subject mappings for each user are normally set up in the Permissioning DataSource, and the default subject mapper provided with the permissioning software provides a `User.setSubjectMapping()` method for doing this.

The following code examples show how this method is used to set up subject mappings for the container `/FILTER/CTR/FI/ALL` and for the constituent record `/FI/US/A` described above.

Setting the subject mapping for the container

```
myUserInstance.setSubjectMapping("/FILTER/CTR/FI/ALL.*", ";mapsuffix/TIER1")
```

Tip: The `.*` at the end of the regular expression `/FILTER/CTR/FI/ALL.*` matches any filter or sort string. In the example shown above, it matches the filter string `?filter=(CpnRate=5)`.

Setting the subject mapping for the constituent record

```
myUserInstance.setSubjectMapping("/FI/US/A", "/TIER1")
```

Note: While Caplin Refiner can apply suffix subject mappings for the constituent records of containers that it filters, it cannot apply any other kind of container or record subject mapping.

Note: Users must be permissioned for the mapped subject of constituent records. In the example above, the container has only one constituent record and the mapped subject is `/FI/US/A/TIER1`. For further information about subject mapping and permissioning, please refer to the Permissioning documents 40.

5.3 Permissioning documents

The following documents provide additional information about user permissions and subject mapping.

- **Caplin Xaqua: Permissioning Overview And Concepts**

  Introduces permissioning concepts and terms, and provides an overview of subject mapping.

- **Caplin Xaqua: How to Create a Permissioning DataSource Adapter**

  Describes how you can use the Permissioning DataSource API to create a Permissioning DataSource adapter, and describes how to set up subject mappings for a user.

- **Permissioning DataSource: API Reference**

  Documents the Java™ classes and interfaces that allow you to integrate Caplin Xaqua with a Permissioning System.

6 Using advanced features of Caplin Refiner

Caplin Refiner offers the following advanced features:

6.1 Custom sorting

When sorting a container, you can specify whether the field type is numeric or text. However in some cases you might want to use your own logic to order field values, rather than a simple alphabetic or numeric comparison.

One example would be sorting a container of bonds by their credit ratings. The ordering used by Standard & Poor's for upper-medium grade bonds and higher is:

`A`	Lowest rating
`A+`	
`AA-`	
`AA`	
`AA+`	
`AAA`	Highest rating

However, if you sort these ratings with a normal ascending alphabetic sort, the resulting order is quite different:

`A`	Lowest rating
`A+`	
`AA`	
`AA+`	
`AA-`	
`AAA`	Highest rating

To deal with problems such as this, you can customize the sort order. There are two ways to do this:

- For simple cases, use custom sort lists 42.
- For more complex sorts, you can write your own custom sort comparators 42.

6.2 Custom sort lists

Custom sort lists provide a simple way to specify a custom sort order.

For example, assume you need to sort container records on the value of the `Tenor` (settlement date) field, where the ordered list of tenor values is (earliest date first):

```
SPOT,ON,TN,SN,1W,2W,1M,2M,6M,1Y
```

To define a custom sort based on an ordered list of values:

- Create the *<Transformer root>/etc/refiner.properties* configuration file if it does not exist already.
- Add a line of the form:

```
sorting.algorithm.<sort-name>=list:<item1>, <item2>,...
```

For the tenor example:

```
sorting.algorithm.TenorSort=list:SPOT,ON,TN,SN,1W,2W,1M,2M,6M,1Y
```

Tip: For the full definition of the `sorting.algorithm.<sort-name>` property, see Configuration properties 30 in Configuring Caplin Refiner 29.

6.3 Custom sort comparators

When you need to sort records where the logic defining the sort order is more complex than just a simple text / numeric comparison or ordered list of values, you can write your own custom sort comparator for Caplin Refiner.

Writing a custom sort comparator

The custom sort comparator is a Java class that implements the **java.util.Comparator<String>** interface. For detailed information on how to implement such a class, see the API reference documentation for the Java Platform Standard Edition.

During the sorting process, Caplin Refiner calls the **compare()** method of the custom sort comparator class:

```
int compare(String object1, String object2)
```

The method compares `object1` against `object2`, and returns:

- A negative integer if `object1` is deemed to be less than `object2`.
- Zero if `object1` is deemed to be equal to `object2`.
- A positive integer if `object1` is deemed to be greater than `object2`.

When Caplin Refiner calls **compare()**, `object1` and `object2` are the field values of two records to be compared.

As an example, consider filtering a container of bonds by their credit ratings, where the Standard & Poor's grading applies for upper-medium grade bonds and higher (see Custom Sorting 41):

`A`	Lowest rating
`A+`	
`AA-`	
`AA`	
`AA+`	
`AAA`	Highest rating

You could implement a custom sort comparator for this scheme, where the implemented **compare()** method would behave according to the following examples:

`compare("AA", "AAA")`	returns `-1` (`AA` is less than `AAA`)
`compare("AA", "AA+")`	returns `-1` (`AA` is less than `AA+`)
`compare("AA", "AA")`	returns `0` (`AA` is equal to `AA`)
`compare("AA", "AA-")`	returns `1` (`AA` is greater than `AA-`)
`compare("AA", "A+")`	returns `1` (`AA` is greater than `A+`)
`compare("AA", "A")`	returns `1` (`AA` is greater than `A`)
`compare("AA-", "AA")`	returns `-1` (`AA-` is less than `AA`)
`compare("AA+", "A+")`	returns `1` (`AA+` is greater than `A+`)

For example purposes, the following sections assume the custom sort comparator class that implements this scheme is called **com.example.BondRatingComparator**.

> **Tip:** If correctly implemented according to the specification of the **java.util.Comparator<String>** interface, a custom sort comparator can also be used as a *custom filter* comparator for the same record field (see Custom filter comparators 45).

Deploying the custom sort comparator

The custom comparator class must be available on the classpath for Caplin Refiner. The easiest way to achieve this is to compile the class and put it in a JAR file, then deploy it to the *<Transformer root>/lib-ext* directory:

1. Compile *BondRatingComparator.java* to *BondRatingComparator.class*
2. Place *BondRatingComparator.class* in a JAR called *custom-comparators.jar*
3. Move the jar file to *<Transformer root>/lib-ext*

Adding the JAR to the classpath

Add configuration to make the new custom comparator class known to Caplin Refiner.

- Installing Caplin Refiner 26 describes how to add to the Transformer configuration file a section that defines Caplin Refiner. Add to this configuration another `classpath` line specifying the location of the JAR for the new custom comparator class:

```
add-javaclass
   class-name  com.caplin.transformer.refiner.Refiner
   class-id    jtm
   classpath   %r/lib-ext/refiner.jar
   classpath   %r/lib-ext/custom-comparators.jar
end-javaclass
```

Configuring the custom sort comparator

Assign the custom comparator a unique name that Caplin Refiner uses to determine when to use it for sorting:

- Create the *<Transformer root>/etc/refiner.properties* configuration file if it does not exist already.
- Add a line of the form:

```
sorting.algorithm.<sort-name>=class:<comparator-class>
```

where `<comparator-class>` is the fully qualified class name of your custom comparator.

For example:

```
sorting.algorithm.BondRating=class:com.example.BondRatingComparator
```

Tip: For the full definition of the `sorting.algorithm.<sort-name>` property, see Configuration properties 30 in Configuring Caplin Refiner 29.

Using a custom sort in client code

To use a custom sort in a client application, call the appropriate API methods provided in the StreamLink library that your application uses.

Tip: For a list of the StreamLink implementations have an API for using Caplin Refiner, see Appendix A: StreamLink support for container filtering 48.

6.4 Custom filter comparators

You may need to filter records where the logic defining the how the records match the filter is more complex than just a simple text / numeric comparison. In this case, you can write your own custom filter comparator for Caplin Refiner.

Writing a custom filter comparator

The custom filter comparator is a Java class that implements the **java.util.Comparator<String>** interface. For information on how to implement such a class, see the API reference documentation for the Java Platform Standard Edition.

During the filtering process, Caplin Refiner calls the **compare()** method of the custom filter comparator class, passing the value of the criterion you are filtering against and the field value of the record to be compared. Your custom code must return a value which represents how the record value compares to the filter criterion:

```
int compare(String object1, String object2)
```

The method compares `object1` against `object2`, and returns:

- A negative integer if `object1` is deemed to be less than `object2`.
- Zero if `object1` is deemed to be equal to `object2`.
- A positive integer if `object1` is deemed to be greater than `object2`.

When Caplin Refiner calls **compare()**, `object1` is the value of the filter criterion, and `object2` is the corresponding field value of the record being filtered.

As an example, consider filtering a container of bonds by their credit ratings, where the Standard & Poor's grading applies for upper-medium grade bonds and higher (see Custom Sorting 41):

`A`	Lowest rating
`A+`	
`AA-`	
`AA`	
`AA+`	
`AAA`	Highest rating

The implemented **compare()** method should behave according to the following examples:

`compare("AA", "AAA")`	returns `-1` (`AA` is less than `AAA`)
`compare("AA", "AA+")`	returns `-1` (`AA` is less than `AA+`)
`compare("AA", "AA")`	returns `0` (`AA` is equal to `AA`)
`compare("AA", "AA-")`	returns `1` (`AA` is greater than `AA-`)
`compare("AA", "A+")`	returns `1` (`AA` is greater than `A+`)
`compare("AA", "A")`	returns `1` (`AA` is greater than `A`)
`compare("AA-", "AA")`	returns `-1` (`AA-` is less than `AA`)
`compare("AA+", "A+")`	returns `1` (`AA+` is greater than `A+`)

At run-time, Caplin Refiner's filtering software calls the **compare()** method as:

```
compare(value, record-field);
```

So, if at run-time the filter is `(rating<AA)`, the method is called as

```
compare("AA", rating);
```

and all comparisons returning `1` pass the filter.

Therefore only bonds with rating `AA-`, `A+`, or `A` are returned to the client.

If instead the filter is `(rating>=AA)`, all comparisons returning `-1` or `0` pass the filter, so only bonds with rating `AA`, `AA+`, or `AAA` are returned to the client.

> **Tip:** If correctly implemented according to the specification of the **java.util.Comparator<String>** interface, a custom filter comparator can also be used as a *custom sort* comparator for the same record field (see Custom sort comparators 42).

For example purposes, the following sections assume the custom filter comparator class is called **com.example.BondRatingComparator**.

Deploying the custom filter comparator

The custom comparator class must be available on the classpath for Caplin Refiner. The easiest way to achieve this is to compile the class and put it in a JAR file, then deploy it to the *<Transformer root>/lib-ext* directory:

1. Compile *BondRatingComparator.java* to *BondRatingComparator.class*
2. Place *BondRatingComparator.class* in a JAR called *custom-comparators.jar*
3. Move the jar file to *<Transformer root>/lib-ext*

Adding the JAR to the classpath

Add configuration to make the new custom comparator class known to Caplin Refiner.

- Installing Caplin Refiner 26 describes how to add to the Transformer configuration file a section that defines Caplin Refiner. Add to this configuration another `classpath` line specifying the location of the JAR for the new custom comparator class:

```
add-javaclass
   class-name  com.caplin.transformer.refiner.Refiner
   class-id    jtm
   classpath   %r/lib-ext/refiner.jar
   classpath   %r/lib-ext/custom-comparators.jar
end-javaclass
```

Configuring the custom filter comparator

Assign the custom comparator a unique name that Caplin Refiner uses to determine when to use it for filtering:

- Create the *<Transformer root>/etc/refiner.properties* configuration file if it does not exist already.
- Add a line of the form:

```
filtering.algorithm.<filter-name>=<comparator-class>
```

where `<comparator-class>` is the fully qualified class name of your custom comparator.

For example:

```
filtering.algorithm.BondRating=com.example.BondRatingComparator
```

Tip: For the full definition of the `filtering.algorithm.<filter-name>` property, see Configuration properties 30 in Configuring Caplin Refiner 29.

Using a custom filter in client code

To use a custom filter in a client application, call the appropriate API methods provided in the StreamLink library that your application uses.

Tip: For a list of the StreamLink implementations have an API for using Caplin Refiner, see Appendix A: StreamLink support for container filtering 48.

7 Appendix A: StreamLink support for container filtering

The following table indicates which StreamLink For Browsers implementations have a specific API for filtering and sorting using Caplin Refiner. This list was correct at the time of publication, but for the latest version, please **contact Caplin Support**.

StreamLink Implementation	Filtering and Sorting API?
StreamLink.NET	✔
StreamLink for Java	✔
StreamLink for Silverlight	✔
StreamLink for Browsers (see **Tip** below)	✘
StreamLink for Flex	✘
StreamLink for iOS	✘

Tip: Caplin Trader (which uses StreamLink For Browsers) has an API for container filtering and sorting using Caplin Refiner.

8 Glossary of terms and acronyms

This section contains a glossary of terms, abbreviations, and acronyms relating to the use of containers in Caplin Xaqua.

Term	Definition
CSV	Comma Separated Values. A set of file formats used to store tabular data, in which numbers and text are stored in plain-text form that can be easily written and read in a text editor. (Source: Wikipedia http://en.wikipedia.org/wiki/Comma-separated_values)
Base container	The name of the container as specified in a **Caplin Refiner** filter or sort request.
Blotter	A record of the details of transactions made by an end-user, such as instrument trades in a trade blotter.
Caplin Liberator	A real-time financial internet hub that delivers trade messages and market data to and from subscribers over any network.
Caplin Trader	A web application framework for financial trading.
Caplin Trader application	A **Caplin Xaqua client** that has been built using **Caplin Trader**.
Caplin Xaqua	A framework for building single-dealer platforms that enables banks to deliver multi-product trading direct to client desktops.
Caplin Xaqua client	A client desktop or web application that interfaces with **Caplin Xaqua** to deliver multi-product trading to end users.
Caplin Refiner	A module of **Caplin Transformer** that filters, sorts, and groups the contents of a container on behalf of **Caplin Xaqua clients**. Caplin Refiner is a **Java Transformer Module**. See Filtering and sorting containers using Caplin Refiner 23.
Caplin Trader	A web application framework for financial trading. An application constructed with Caplin Trader is a **Caplin Xaqua client**.
Caplin Transformer	An event-driven real-time business rules engine.
Container	A data structure in **Caplin Xaqua** that allows **Caplin Liberator** to manage lists of data (such as lists of financial instruments) on behalf of **Caplin Xaqua clients**. See What is a container? 4
Container order	The order of the elements within a **container** is defined by the DataSource that provides the container. This ordering is known as “container order” or “**natural order**”.
Container windowing	See the section Container windowing 7.
Data service	A **Caplin Liberator** configuration facility that allows you to define where Liberator data comes from, based on its subject name, and allowing for priority, failover, and load balancing.
DataSource	**DataSource** is the internal communications infrastructure used by Caplin Xaqua's server components such as **Caplin Liberator**, **Caplin Transformer**, and **DataSource adapters**.

Term	Definition
DataSource adapter	A **DataSource application** that integrates with an external (non-Caplin) system, exchanging data and/or messages with that system.
DataSource application	A **Caplin Xaqua** application that uses the Caplin **DataSource** APIs to communicate with other Caplin Xaqua applications via the DataSource protocol.
Display component	A GUI component of **Caplin Trader** that can be rendered in a page on the screen. The term also refers to the JavaScript code that generates the component and handles its user interaction.
Filter expression	A mathematical expression that defines how **Caplin Refiner** is to filter the elements of a **container**. For example `CpnRate>6 AND CpnRate<11` See Filtering rules 33.
Grid	A **display component** that displays data in a tabular format.
Java Transformer Module	A **Transformer module** that is implemented in the Java language..
Liberator Auth module	A **Caplin Liberator** module that performs authentication and authorization.
Natural order	Alternative term for **container order**.
Permissioning Auth Module	A **Liberator auth module** that allows Liberator to integrate with the **Caplin Xaqua** permissioning system. For more information, see the document **Caplin Xaqua: Permissioning Overview and Concepts**.
Permissioning DataSource	A **DataSource application** that provides permissioning information for **Caplin Xaqua** that conforms to the permissioning model described in **Caplin Xaqua: Permissioning Overview and Concepts**.
RTTP	Real Time Text Protocol. Caplin's protocol for streaming real-time financial data from **Caplin Liberator** servers to **Caplin Xaqua clients**, and for transmitting trade messages and other messages between clients and Liberator in both directions.
StreamLink	The StreamLink libraries connect client applications to **Caplin Liberator** via the **RTTP** protocol. They provide an object oriented API that gives access to RTTP functionality.
StreamLink for Browsers	StreamLink for Browsers is a JavaScript implementation of **StreamLink** that runs in Web browsers. It allows Ajax applications, such as **Caplin Trader** applications, to communicate with **Caplin Liberator**.
Sort rule	The set of criteria for a container sort in **Caplin Refiner**. See Sort rules 34.
Transformer module	A processing module in **Caplin Transformer**.

Contact Us

Caplin Systems Ltd
Cutlers Court
115 Houndsditch
London EC3A 7BR
Telephone: +44 20 7826 9600
www.caplin.com