CAPLIN

Caplin Xaqua 1.0

Permissioning Overview And Concepts

March 2012

Contents

1 Preface

1.1 What this document contains

This document introduces permissioning concepts and terms, and shows the permissioning components of the Caplin Xaqua architecture.

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 Technical Managers, Enterprise Architects, System Architects, System Administrators, and Software Developers who want to integrate Caplin Xaqua with a Permissioning System.

1.3 Related documents

- **Caplin Xaqua: Overview**

  Provides a business and technical overview of Caplin Xaqua and includes an explanation of its architecture.

- **Caplin Liberator: Administration Guide**

  Describes how to install and configure Caplin Liberator and discusses the authentication modules that are provided with the server.

- **Caplin Xaqua: Installing Permissioning Components**

  Describes how to install and configure the Permissioning Auth Module and Permissioning DataSource in an existing Caplin Xaqua installation. You only need to install these components if your installation of Caplin Trader is earlier than release 1.2.8, as later releases include these permissioning components.

  Also describes the Permissioning Auth Module properties that determine how **TOBO** is controlled by Liberator's Permissioning Module.

- **Caplin Xaqua: How To Create A Permissioning DataSource Adapter**

  Describes how to create a Permissioning DataSource adapter using the Permissioning DataSource API. A Permissioning DataSource adapter is required to integrate Caplin Xaqua with a Permissioning System. The document also discusses the Demo Permissioning DataSource provided with the reference implementation of Caplin Trader from release 1.2.8.

- **Caplin Trader: How To Add Permissioning At The Client**

  Describes how to add permissioning to Caplin Trader.

- **Permissioning DataSource: API Reference**

  The API reference documentation provided with the Permissioning DataSource SDK (Software Development Kit). The classes and interfaces presented by this API allow you to write a Java application that will integrate a Permissioning System with Caplin Xaqua.

- **Caplin Trader: API Reference**

  The API reference documentation provided with Caplin Trader. The classes and interfaces of the `caplin.security.permissioning` package allow you to write JavaScript classes that extend the live permissioning capabilities of Caplin Trader.

1.4 Typographical conventions

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

Type	*Uses*
aMethod	Function or method name
aParameter	Parameter or variable name
/AFolder/Afile.txt	File names, folders and directories
`Some code;`	Program output and code examples
The `value=10` attribute is...	Code fragment in line with normal text
Some text in a dialog box	Dialog box output
`Something typed in`	User input – things you type at the computer keyboard
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

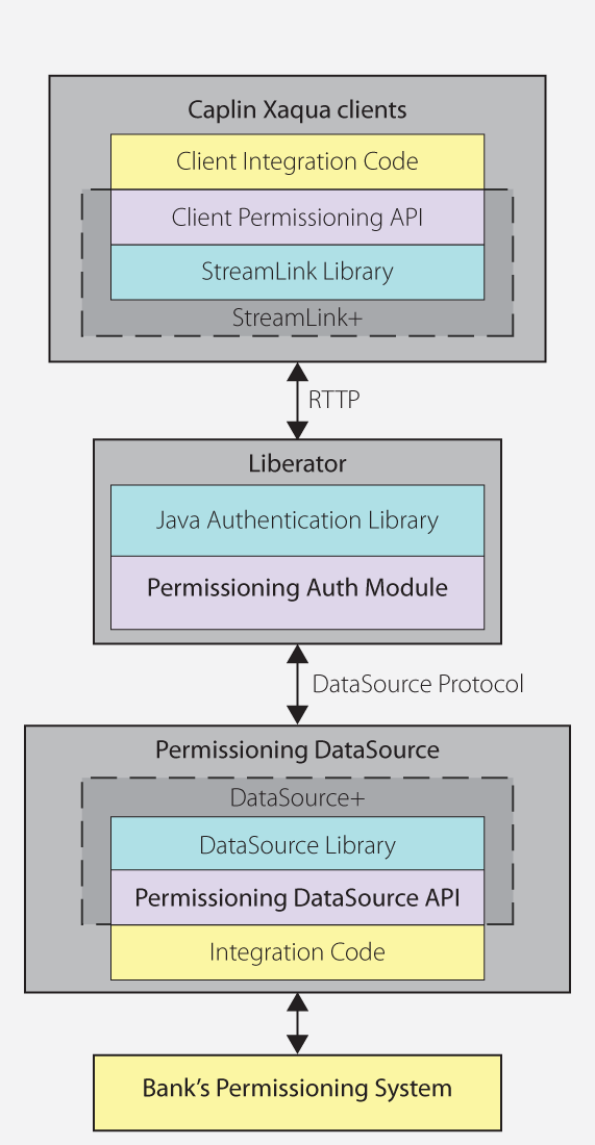
Customer feedback can only improve the quality of our product documentation, and we would welcome any comments, criticisms or suggestions you may have regarding this document.

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

1.6 Acknowledgments

2 Permissioning components

Caplin Xaqua consists of a number of components that interact to provide end-users with a web-based financial trading application (see the document **Caplin Xaqua: Overview**). The components that are used to integrate a **Permissioning System** with Caplin Xaqua are shown in the diagram below.



Simplified Caplin Xaqua architecture showing only permissioning components

When a **Caplin Xaqua client** application such as **Caplin Trader** interacts with **Liberator**, for example when an end-user logs in or attempts to view or trade financial information, Liberator uses the services of a **Permissioning Auth Module** to decide if the interaction with Liberator is permitted or not. The Permissioning Auth Module enforces security constraints based on permissioning data provided to Liberator by the **Permissioning DataSource**.

Permissioning System

The Permissioning System is the source of the permissioning data that you want to integrate with Caplin Xaqua. Permissioning data identifies the end-users that are authorized to interact with Liberator, and the financial **products** that these end-users are permitted to view and trade.

Permissioning DataSource

The Permissioning DataSource is a **DataSource Adapter** that acts as the interface between Caplin Xaqua and your Permissioning System. Its purpose is to provide Liberator with permissioning data. The Permissioning Auth Module in Liberator uses this data to decide whether or not an interaction with Liberator is permitted.

To create a Permissioning DataSource, you write and compile a Java application that uses the Permissioning DataSource **API**. This simple API is built on top of the Caplin **DataSource for Java** API, allowing your application to send permissioning data to Liberator using the DataSource protocol, but without the need for your code to explicitly use the DataSource for Java API.

Liberator can be configured to connect to one or more Permissioning DataSources.

Permissioning Auth Module

When a Permissioning DataSource is used to integrate Caplin Xaqua with a Permissioning System, the Liberator server must be configured to use the Permissioning Auth Module, which makes decisions about who is permitted to access the server and the type of access permitted. The Permissioning Auth Module is one of several authentication modules that are supplied with Caplin Xaqua, and uses the Java Authentication API to communicate with Liberator.

You will find further information about the authentication modules supplied with Caplin Xaqua in the **Caplin Liberator: Administration Guide**.

Permissioning at the Client

The components of the Graphical User Interface (GUI) at the client can be tailored to match the **permissions** of the **user** who is currently logged in. An example would be to display information that the user is allowed to view, and to hide information that the user is not allowed to view. This information could be anything from pricing data for currency-pairs, to menu items, data grids, trade tiles, and tenors.

To add permissioning to a client application that is based on the Caplin Trader framework, you must modify the client application by adding JavaScript code that uses the Permissioning API of Caplin Trader. This simple API can retrieve permissioning data from Liberator and is built on top of the StreamLink for Browsers library. StreamLink for Browsers provides communication between Caplin Trader and the Liberator server using the RTTP protocol.

3 Permissioning concepts

We will now look at some of the permissioning concepts that allow Liberator to enforce security constraints on the end-users of a Caplin Xaqua client application.

3.1 Products

A product is an entity on which end-users may be assigned permissions.

> **Note:** In other Caplin Xaqua documentation, a "product" is a term that refers only to a financial instrument. In permissioning documentation (including this document) a "product" is any entity on which an end-user may be assigned permissions (including financial instruments).

Examples of products are:

- "All FI Instruments"
- "FX Instrument GBPUSD"
- "LIBOR based swaps"
- "The Blotter"

You define products when you write the Permissioning DataSource application.

3.2 Actions

An **action** defines the interaction that a **user** can have with a product.

Examples of product actions are:

- "Viewing"
- "RFQ trading"
- "One-Click trading"

You define product actions when you write the Permissioning DataSource application.

3.3 Permissions

A **permission** determines whether an action on a product will be allowed or denied.

When you define a permission you can also define a namespace. A namespace allows client applications to query related permissions, such as all permissions in the "tenor" namespace. If you do not define a namespace, then the permission will reside in the *default* namespace.

Examples of product permissions (each row in the table defines a permission):

Action	Product	Namespace	Authorization
Viewing	LIBOR based swaps		Allow
RFQ Trading	All FI Instruments		Allow

Action	Product	Namespace	Authorization
One-Click Trading	FX Instrument GBPUSD	Quick Trades	Deny
One month settlement	All Instruments	Tenor	Allow

Permissions are assigned to Users 8 and Groups 8 when you write the Permissioning DataSource application.

3.4 Users

A user represents an end-user of a Caplin Xaqua client application. A user can only log in to a Caplin Xaqua client application if permissioning data to authenticate the user has been supplied to Liberator. Users can also be assigned permissions for the products streamed by Liberator.

Users are defined when you write the Permissioning DataSource application. Data for constructing each user would typically be read from a configuration file or persistent Permissioning System. The **Demo Permissioning DataSource** that is supplied with the reference implementation of Caplin Trader reads this data from an XML file (see **Caplin Xaqua: How To Create A Permissioning DataSource**).

3.5 Groups

Users can be arranged in **groups**, where every member of a group has the same permissions. A user can be a member of more than one group, and groups can be members of other groups. This allows an inheritance tree of permissions to be created.

In the example below, user X is a member of two groups:

- Group "FI Traders". This group is permitted "RFQ trading" on "All FI Instruments"
- Group "FX Traders". This group is permitted "RFQ trading" on "All FX Instruments"

Since user X is a member of "FI Traders" and "FX Traders", they will be able to trade both FX and FI instruments.

You will find another example of an inheritance tree in Example permissioning hierarchy 15.

When permissions in an inheritance tree are in conflict, conventions are used to determine the permission that is assigned to the user. These conventions are described in Permissioning hierarchy conventions 12.

Groups are defined when you write the Permissioning DataSource application. Data for constructing each group would typically be read from a configuration file or persistent Permissioning System. The Demo Permissioning DataSource that is supplied with the reference implementation of Caplin Trader reads this data from an XML file (see **Caplin Xaqua: How To Create A Permissioning DataSource**).

3.6 Rules

In addition to defining permissions, you must also define permissioning **rules**. Rules link permissions to user interactions, and are used by Liberator to decide which of the many permissions that have been defined will apply when a user attempts to interact with a product.

When a Caplin Xaqua client requests data from Liberator, or tries to publish data to Liberator, the Permissioning Auth Module inspects the content of the associated RTTP message and tries to match it with one or more of the defined rules. If a match is found then the rule will identify the permission that must be checked to determine whether this interaction with Liberator is to be allowed or denied.

If more than one rule matches an RTTP message, then every one of the identified permissions will be checked. If any one of these permissions denies access, then access to the product will be denied.

Rules *must* be defined for every interaction that involves a Caplin Xaqua client publishing data to Liberator, such as when a user attempts to trade a product. If a rule has not been defined for such an interaction, then the default permission is to *deny* the interaction.

Rules *cannot* be defined for interactions that involve a Caplin Xaqua client requesting data, such as when a user attempts to view a product. With this type of interaction a **default rule** is implemented, and the permission that has been defined for the "VIEW" action in the *default* namespace will be checked to see if the interaction is to be allowed or denied.

Simple example of a matching rule

The following example shows a typical RTTP message that would be sent to Liberator when a user tries to spot trade a product.

RTTP Message

Message Subject	Message Fields			
	MsgType	*Trading-Type*	*Amount*	*Instrument*
/FT/TRADE	Execute	SPOT	1000000	/FX/GBPUSD

An example of a rule that would match this type of trade is:

- Check the user permission for the action "spot-trade" when the subject of the RTTP message to Liberator is "/FT/TRADE" and the "Trading-Type" is "SPOT". The product name to be checked is given by the "Instrument" field of the RTTP message.

Matching rule

Subject Match	Field Match Criteria	Product Reference Field	Action	Namespace
/FT/TRADE	Trading-Type=SPOT	Instrument	spot-trade	

In this case the permission that has been defined for the action "spot-trade" in the *default* (undefined) namespace will be checked to see if the user is to be allowed or denied access to the product.

The permission must also match the product that is identified by the "Instrument" field of the RTTP message, which in this case is "/FX/GBPUSD".

An example of a permission that would apply to this type of trade is shown below.

Permission that would apply

Action	Product	Namespace	Authorization
spot-trade	/FX/GBP.*		Allow

The permission in the *default* (undefined) namespace is checked since a namespace was not defined by the matching rule.

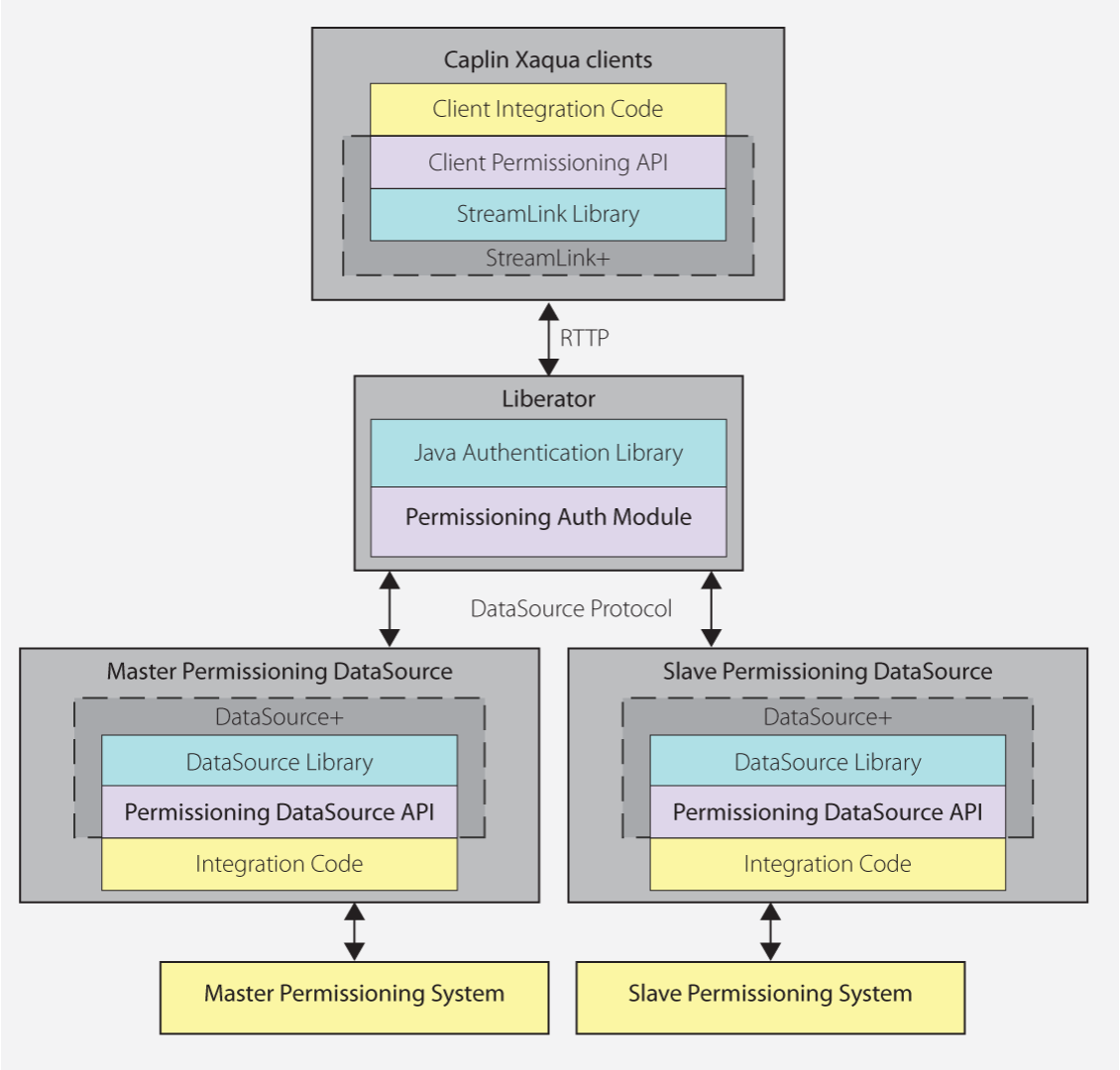
In this case "spot-trade" action on all "/FX/GBP" products will be allowed. The ".*" characters following "GBP" are metacharacters in a Java regular expression, such that "/FX/GBP.*" will match all GBP currency pairs.

Tip: You will find further information about Java regular expressions in the API documentation for the **java.util.regex** package, as supplied by Oracle® Corporation.

Rules are defined when you write the Permissioning DataSource application. Data for constructing each rule would typically be read from a configuration file or persistent Permissioning System. The Demo Permissioning DataSource that is supplied with the reference implementation of Caplin Trader reads this data from an XML file (see **Caplin Xaqua: How To Create A Permissioning DataSource**).

3.7 Roles

Roles determine whether a Permissioning DataSource has been designated as the **master** or **slave**.



Multiple Permissioning DataSource Adapters connected to Liberator (showing one master and one slave)

Liberator can be configured to accept permissioning data from multiple Permissioning DataSources. This allows permissions from different permissioning systems to be provided by different Permissioning DataSources. For example, one Permissioning DataSource could provide permissions for FX products and another permissions for FI products, with each Permissioning DataSource being administered by a different department.

When permissioning data is sent to Liberator from more than one Permissioning DataSource, one Permissioning DataSource is designated the master and each of the other Permissioning DataSources are designated as slaves. A slave Permissioning DataSource can only send a limited a set of permissioning data to Liberator and there can only be one master.

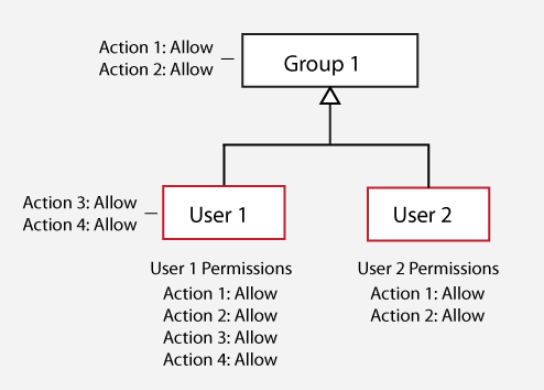
The document **Caplin Xaqua: How To Create A Permissioning DataSource Adapter** describes how to set the master and slave roles, and defines the limited set of permissioning data that a slave can send to Liberator.

3.8 Permissioning hierarchy conventions

We will now look at permissioning hierarchies and how permissioning conflicts are resolved.

Single Inheritance

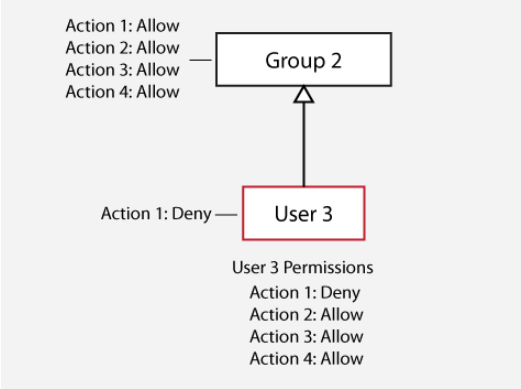
A user can be assigned their own permissions in addition to inheriting group permissions.



In this case, User 1 is assigned permissions and also inherits the permissions of Group 1, while User 2 is not assigned any permissions but inherits the permissions of Group 1.

Inheritance Masking

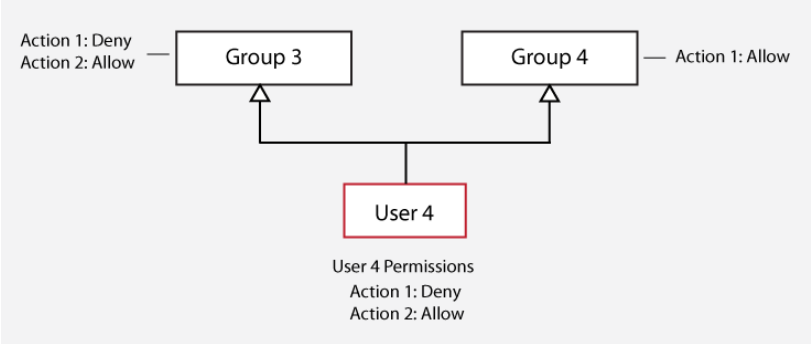
A permission assigned to a user or group masks permissions for the same action and product in parent groups.



In this case, the "Deny" permission assigned to User 3 for Action 1 masks the "Allow" permission for Action 1 in Group 2. The matching permission closest to the user in the permissioning hierarchy is always used to determine whether an action will be allowed or denied.

Multiple Inheritance Conflicts

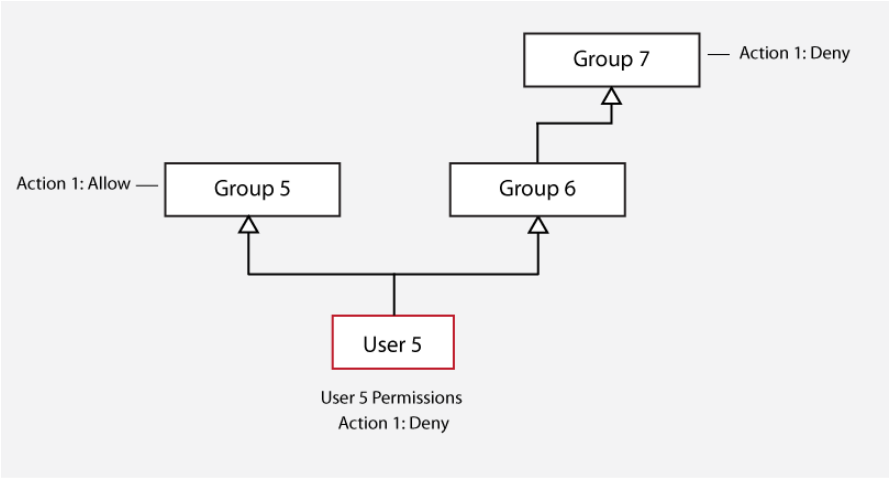
If a user is a member of more than one group and any of the inherited permissions deny an action, then this permission will override any inherited permissions that allow the action.



In this case, the "Deny" permission for Action 1 that User 4 inherits from Group 3 overrides the "Allow" permission for Action 1 that User 4 inherits from Group 4.

Multiple Inheritance Conflicts in a Complex Hierarchy

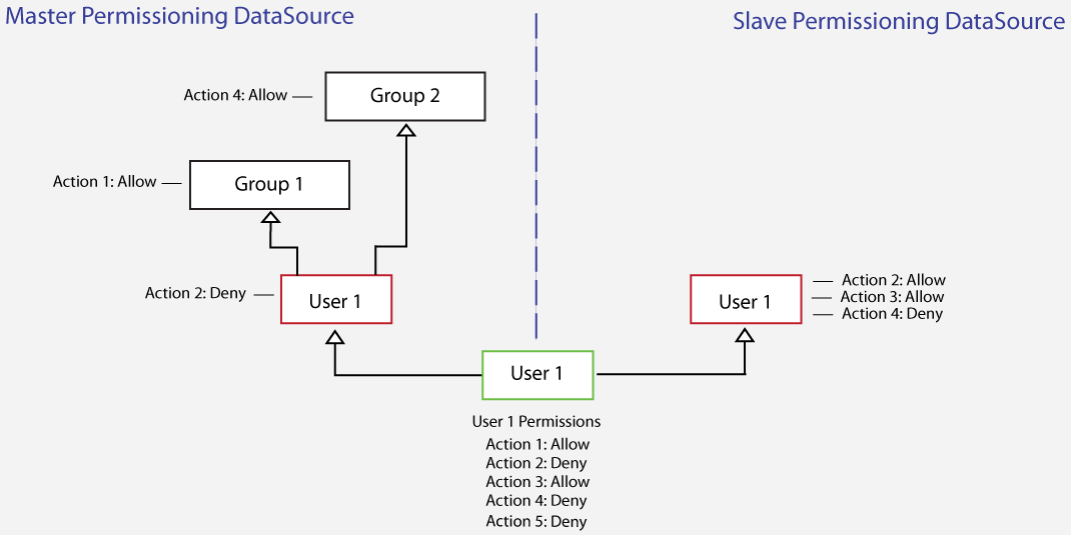
"Deny" permissions also override "Allow" permissions in more complex hierarchies.



In this case, the "Deny" permission that User 5 inherits from Group 7 overrides the "Allow" permission that User 5 inherits from Group 5. Group 5 was compared to Group 7 because Group 6 had no permission to compare.

Permissions from Multiple Permissioning DataSources

"Deny" permissions also override "Allow" permissions when permissions are set in the master and slave Permissioning DataSources.



In this case:

- The "Deny" permission assigned to User 1 for Action 2 in the master overrides the "Allow" permission assigned to User 1 for Action 2 in the slave.
- The "Deny permission assigned to User 1 for Action 4 in the slave overrides the "Allow" permission for Action 4 that User 1 inherits from Group 2 in the master.
- Because permissions for Action 5 are not defined in the master or slave, the permission for Action 5 is, by default, "Deny".

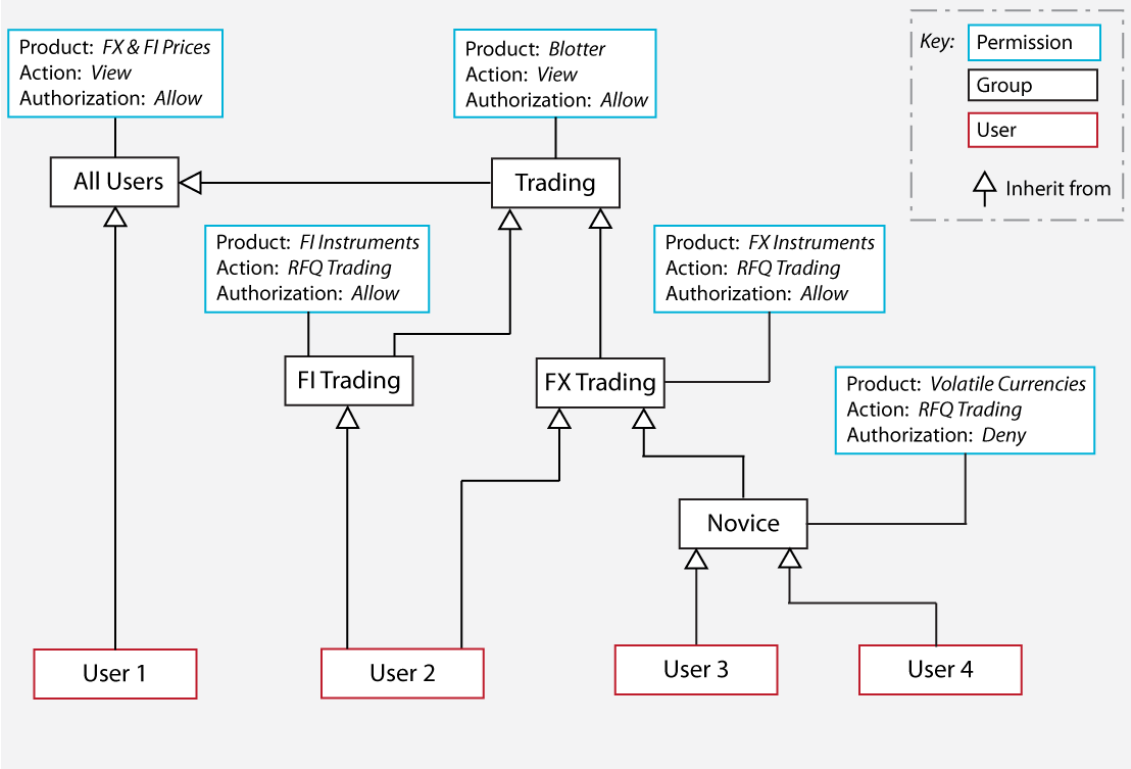
To summarize, when permissions for an action are sent to Liberator from master and slave Permissioning DataSources, permissioning conflicts are resolved as shown in the following table.

Master/Slave permissioning conflicts

Master permission	Slave permission	Combined permission
"Allow"	"Allow"	"Allow"
"Allow"	Undefined	"Allow"
Undefined	"Allow"	"Allow"
"Allow"	"Deny"	"Deny"
"Deny"	"Allow"	"Deny"
Undefined	Undefined	"Deny"

3.9 Example permissioning hierarchy

The diagram below shows a permissioning hierarchy that uses the permissioning concepts introduced in this section. In this example, all permissions reside in the *default* namespace.



Example permissioning hierarchy

From this diagram you will see that users have been assigned permissions through inheritance.

User 1 is a view only user

- Allowed to view all FX and FI prices (inherited from "All Users")

User 2 is an experienced trader

- Allowed to view all FX and FI prices (inherited from "All Users")
- Allowed to view a blotter of previous trades (inherited from "Trading")
- Allowed to RFQ trade FI Instruments (inherited from "FI Trading")
- Allowed to RFQ trade FX Instruments (inherited from "FX Trading")

User 3 and User 4 are novice traders with the same product permissions

- Allowed to view all FX and FI prices (inherited from "All Users")
- Allowed to view a blotter of previous trades (inherited from "Trading")
- Allowed to RFQ trade FX Instruments *that are not Volatile Currencies* (FX Instruments inherited from "FX Trading", but Volatile Currencies denied by "Novice")

4 Complex permissioning rules

Simple permissioning rules 9 were introduced when we discussed permissioning concepts 7. We will now look at how to construct more complex permissioning rules.

4.1 Matching the RTTP message subject

Java regular expressions and Caplin substitution tokens can be used when you define a rule or permission.

- Java regular expressions can be used in the Subject Match expression of a rule, and in the Product expression of a permission. For example "/F." matches "/FT" and "/FI", since the "." metacharacter matches any single character.

> **Tip:** You will find further information about Java regular expressions in the API documentation for the **java.util.regex** package, as supplied by Oracle® Corporation.

- Substitution tokens can also be used in the Subject Match expression of a rule, and in the Product expression of a permission. They are used to prevent one user from accessing another user's data. The substitution tokens that are understood by the Permissioning system are, %u, %U, and %t
  - %u matches the username of the logged in user.

    For example, "/PRIVATE/%u/FX" matches "/PRIVATE/Bob/FX" if the user logged in as "Bob".
  - %U matches the session name allocated by Liberator for a user session.

    For example, "/PRIVATE/%U/FX" matches "/PRIVATE/Bob-0/FX" if the user's allocated session name is "Bob-0".

    A user can log in more than once, and is allocated a different session name each time they log in.
  - %t matches the username of the logged in user, or any user that the logged in user can trade on behalf of. Its main use is to support permissioning for **TOBO**; see Using the %t substitution token 38 in Permissioning for TOBO 25.

> **Note:** If substitution tokens are used in the definition of rules or permissions, corresponding object mappings must also be set up in the Liberator configuration (see **object-map** in the document **Caplin Liberator Administration Guide**).

> **Note:** **Only use %t in permission definitions** (see Using the %t substitution token 38).

> **Tip:** To place one of the literal values %u, %U, or %t in a regular expression, escape the string with a "\" like this: \%u, so that it is not interpreted as a substitution token.

Using a substitution token in a rule

The following example shows an RTTP message that has a username ("Bob") in the subject of the message.

RTTP Message

Message Subject	Message Fields			
	MsgType	*Side*	*Amount*	*Instrument*
/PRIVATE/Bob/FX/ONECLICK	Execute	Buy	500000	/FX/GBPUSD

An example of a rule that matches this request is:

Matching rule

Subject Match	Field Match Criteria	Product Reference Field	Action	Namespace
/PRIVATE/%u/FX/ONECLICK		Instrument	ONE-CLICK	

Because the matching rule contains the `%u` substitution token, the Permissioning Auth Module compares the username of the logged in user with the username in the requested subject.

If the logged in user is "Bob", the rule is applied and the permission defined for the action "ONE-CLICK" in the default namespace is checked to see if the request from Bob is to be allowed or denied.

If the logged in user is "John" and not "Bob", the Permissioning Auth Module immediately denies the request.

Using a substitution token in a permission

The substitution tokens %u, %U, and %t can also be used when you define a permission.

When a user attempts to view a product a default rule is implemented, and the permission that has been defined for "VIEW" action in the *default* namespace is checked to see if the interaction is to be allowed or denied.

The following example shows a typical RTTP message for this type of request, in which the user attempts to view a product, and the username of the logged in user ("Bob") is in the subject of the message.

RTTP Message

Message Subject
/PRIVATE/Bob/FX/USDGBP

Because the user is attempting to view a product, the **default rule** is implemented. An example of a permission that would apply to this type of request is shown below (it is for the VIEW action and is in the default namespace, so it matches the default rule).

Permission that would apply

Action	Product	Namespace	Authorization
VIEW	/PRIVATE/%u/FX/USDGBP		Allow

In this example, the Permissioning Auth module replaces the %u token in Product with the username of the logged in user, and compares this modified Product with the requested subject.

If the logged in user is "Bob", the modified Product in the permission is /PRIVATE/Bob/FX/USDGBP, which matches the requested subject, so the request from Bob is allowed, provided no other VIEW permissions with Deny authorization match the subject.

If the logged in user is "John" and not "Bob", the modified Product in the permission does not match the requested subject, so it has no effect on the permission for the requested subject, as the request from "John" is neither allowed nor denied by this permission.

Tip: Substitution tokens are most effective when applied to permissions in Groups that multiple users inherit from. Without the token you would have to define a specific permission for each user, containing the user's name.

For information about using the %t substitution token in permissions, see Using the %t substitution token 38.

4.2 Field match criteria

A rule can have zero or more Field Match Criteria that map RTTP message fields and values. All field mappings described by Field Match Criteria must be present in the RTTP message, otherwise the rule will not match the message. In the example below, the rule will only match an RTTP message if the message field "Trading-Type" has the value "SPOT", and the message field "SIDE" has the value "Buy".

Example rule

Subject Match	Field Match Criteria	Product Reference Field	Action	Namespace
/FT/TRADE	Trading-Type=SPOT, SIDE=Buy	Instrument	spot trade	

If the rule does not define any Field Match Criteria, then the RTTP message fields will be ignored when the rule is being matched to the message.

You will notice from the above example that it is not necessary to include the "Instrument" value of Product Reference Field in the Field Match Criteria when you define the rule. Product matching is described in The Product Reference Field 19.

4.3 The Product Reference Field

The Product Reference Field of a rule identifies the field in the RTTP message that contains the name of the product. If the rule matches the message, the permission for the identified product is checked to see if the action is allowed or denied (see Permissions 7).

Matching All Products

The Product Reference Field can be given the special value "ALL_PRODUCTS" (equivalent to the Java regular expression ".*"), in which case all product permissions for the action are checked to see if the action is allowed or denied. We will now look at an example RTTP message and a matching rule that uses this special value.

RTTP Message

	Message Fields			
Message Subject	*MsgType*	*SIDE*	*Amount*	*Instrument*
/FX/ONECLICK	Execute	Buy	500000	/FX/USDGBP

An example of a rule that matches this type of trade is shown below.

Matching rule

Subject Match	Field Match Criteria	Product Reference Field	Action	Namespace
/FX/ONECLICK		ALL_PRODUCTS	ONE-CLICK	

In this case all product permissions for "ONE-CLICK" action in the *default* namespace are checked to see if the user is to be allowed or denied access. If any of these permissions deny "ONE-CLICK" action, access to "/FX/EURGBP" is denied.

An example of a permission that applies to this type of trade is shown below.

Permission that would apply

Action	Product	Namespace	Authorization
ONE-CLICK	/FX/EURGBP		Allow

Although this permission is for "/FX/EURGBP", it also applies to "/FX/USDGBP", since the rule specifies that all product permissions for "ONE-CLICK" action in the *default* (undefined) namespace must be checked to see if the action is allowed or denied.

Using Java Regular Expressions

The Product Reference Field can be a Java regular expression, in which case every field of the RTTP message that matches the regular expression will be considered to contain the name of a product. Java regular expressions can be used to define rules for multi-leg trading, as shown in the following example.

RTTP Message

	Message Fields			
Message Subject	*L1_*	*L2_*		
/TRADE/FX	/FX/GBPUSD	/FX/USDJPY		

An example of a rule that would match this type of trade is shown below.

Matching rule

Subject Match	Field Match Criteria	Product Reference Field	Action	Namespace
/TRADE/FX		L\d_	TRADE	TRADER

The "\d" in the regular expression of the Product Reference Field will match any digit. In this case the permission that has been defined for the "TRADE" action in the "TRADER" namespace will be checked to see if the user is to be allowed or denied access to the products identified by fields "L1_" and "L2_".

An example of the permissions that would apply to this type of multi-leg trade is shown below.

Permissions that would apply

Action	Product	Namespace	Authorization
TRADE	/FX/GBPUSD	TRADER	Allow
TRADE	/FX/USDJPY	TRADER	Allow

In this case "TRADE" action on the products "/FX/GBPUSD" and "/FX/USDJPY" will be allowed, since the rule states that the permission applies to each of these products. If the user did not have "Allow" permission for each of these products, then the multi-leg trade would be denied.

4.4 The Action Reference Field

We have already looked at how you can define the action for a rule in the Action field of the rule 9. Rules can also be defined that derive the action from a field in the RTTP message.

In the following example, we look at a rule that uses the Action Reference Field to identify the field of the RTTP message that defines the action.

RTTP Message

Message Subject	Message Fields		
	Tenor	*Trading-Type*	*Instrument*
/TRADE/FX	1Month	RFQ	/FX/GBPUSD

An example of a rule that would match this type of trade is shown below.

Matching rule

Subject Match	Field Match Criteria	Product Reference Field	Action	Action Reference Field	Namespace
/.*		Instrument		Tenor	TenorPermissions

The rule matches since the subject of the RTTP message is "/TRADE/FX", which matches the regular expression "/.*" in the Subject Match field of the rule.

An example of the permission that would apply to this type of trade is shown below.

Permissions that would apply

Action	Product	Namespace	Authorization
1Month	.*	TenorPermissions	Allow

In this case "RFQ" trading with a tenor of one month will be allowed. The permission matches because:

- The Action Reference Field of the rule identifies "Tenor" as the RTTP field that defines the action.
- The "Tenor" field of the RTTP message has the value "1Month".
- The permission allows "1Month" action for all products in the "TenorPermissions" namespace.

5 Additional permissioning capabilities

We will now look at some additional permissioning capabilities.

5.1 Subject mapping

When a user attempts to view data, a default rule is implemented (see Rules 9). This default rule states that the Permissioning Auth Module must check the user's permission in the *default* namespace for "VIEW" action on the product identified by the subject of the RTTP message. If the action is permitted, then Liberator will stream data to the user for the requested product.

Subject mapping allows the subject of an RTTP message to be modified by Liberator. Subject mapping is transparent to the user and could be used, for example, to provide preferential data to selected users.

The Default Subject Mapper

A default **subject mapper** is provided with the Permissioning software that allows you to add a subject mapping for a user. This mapping specifies a subject pattern and a subject suffix. If the user attempts to view data where the subject of the RTTP message matches the subject pattern, then the subject suffix will be appended to the subject of the RTTP message when Liberator requests the data from the pricing DataSource.

To view the data, the user must be permitted "VIEW" action in the *default* namespace for the product identified by the *modified* subject of the RTTP message.

When Liberator streams the data to the user, the data will be streamed using the *original* RTTP message subject, but for the product identified by the *modified* RTTP message subject.

Only one subject mapping can be added for each user by a Permissioning DataSource using the default subject mapper. The subject mapping is added by calling the `User.setSubjectMapping()` method of the **Permissioning DataSource API**.

Adding Multiple Subject Mappings for a User

The `RegexSuffixSubjectMapper` class of the Permissioning DataSource API allows multiple subject mappings to be added for each user. This class is set as the subject mapper for a user by calling the `User.setSubjectMapper()` method of the Permissioning DataSource API, and subject mappings are added by calling `User.addSubjectMapping()`.

> **Tip:** For a complete description of the Permissioning DataSource API, please refer to the **Permissioning DataSource: API Reference** documentation.

Creating a Custom Subject Mapper

If you want to calculate a subject mapping based on customized mapping logic, then you must create a custom subject mapper. The custom subject mapper must implement the `SubjectMapper` interface of the Permissioning DataSource API and be deployed in the Permissioning Auth Module.

For more information on how to implement and deploy a custom subject mapper, see **Caplin Xaqua: How to Create a Permissioning DataSource Adapter**.

Simple Example of Subject Mapping using the Default Subject Mapper

The following example shows how subject mapping can be applied to a user when Liberator supports **price tiering**. In this example the subject pattern is the regular expression "/PRICES/FX/.*" and the subject suffix is "-tier2".

Applied subject mapping

Subject Pattern	Subject Suffix
/PRICES/FX/.*	-tier2

If the user attempts to view "/PRICES/FX/GBPUSD", Liberator will request the data from the DataSource provider for "/PRICES/FX/GBPUSD-tier2". When Liberator streams the data to the user, it will be streamed from "/PRICES/FX/GBPUSD-tier2" but using the original RTTP message subject "/PRICES/FX/GBPUSD".

An example of a permission that would apply to this type of data request is shown below.

Permission that would apply

Action	Message Subject	Namespace	Authorization
VIEW	/PRICES/FX/.*-tier2		Allow

In this case VIEW action on "/PRICES/FX/GBPUSD-tier2" will be allowed, since "/PRICES/FX/.*-tier2" matches all "-tier2" currency pairs in "/PRICES/FX" and the authorization is "Allow". Note that the permission matches the *modified* message subject, and not the *original* message subject.

Global context data

The **global context** is an object at the Permissioning Auth module. This object allows custom subject mappers to access data that is common to all subject mappers and users. In this way a custom subject mapper can map subjects using logic based on this common data, and not just on subject mappings defined for the user.

For example, if a custom subject mapper uses FX rates to map a subject, it is more efficient to add these rates to the global context than to the subject mappings of every user.

A default global context class is provided with the Permissioning software, but you can also create a custom global context class that provides specialized methods to subject mappers.

The document **Caplin Xaqua: How to Create a Permissioning DataSource Adapter** describes how to create a custom global context class. The document also describes how a Permissioning DataSource can add data to the global context, and how a custom subject mapper can access this data.

5.2 User attributes

A user can be assigned any number of attributes in the form of name/value pairs. User attributes are not processed by the Permissioning Auth Module 6, and therefore do not affect permissioning directly, but they can be accessed by the client application to provide miscellaneous contextual information about the currently logged in user.

A typical use would be to send information to a Caplin Xaqua client about the maximum tradable amount that users are permitted to trade.

Defining and Retrieving User Attributes

In the Permissioning DataSource Adapter, user attributes are defined in Java using the `com.caplin.permissioning.User.setAttribute()` method of the Permissioning DataSource API. In the following example, the maximum tradable amount for a user is set to 5 million USD.

```
/* create new user with login name trader1 */
User maxTradeUser = new User(trader1);

...

/* set the max tradable USD (millions) */
maxTradeUser.setAttribute(maxTradeUSD, 5);
```

Tip: The document **Caplin Xaqua: How To Create A Permissioning DataSource** describes how to create a custom Permissioning DataSource adapter.

In a client application that is based on Caplin Trader, user attributes are retrieved in JavaScript using the `caplin.security.permissioning.PermissionService.getUserAttribute()` method of the Caplin Trader Permissioning API. In the following example, the maximum tradable amount in USD is retrieved for the currently logged in user.

```
/* get the max tradable USD (millions) */
maxTradableUSD =
    caplin.security.permissioning.PermissionService.getUserAttribute(maxTradeUSD);
```

Tip: The document **Caplin Trader: How to Add Permissioning At The Client** describes how to modify the appearance and behaviour of Caplin Trader to match the permissions of the currently logged in user.

5.3 Permissioning for TOBO

TOBO stands for “trading on behalf of”. TOBO allows a user who is logged in to a Caplin Xaqua client to execute trades on instruction from a customer (for example, the customer may give instructions by telephone). The logged in user is called a **sales-user**, and the customer is called a **customer-user** (because they are known to Caplin Xaqua as an end-user, and are a customer of the organization that employs the sales-user). The sales-user trades *on behalf of* the customer-user.

An important aspect of this arrangement is that each customer-user can be configured to receive tailored prices for any given traded instrument (**price tiering**). The prices offered to the customer-user should be the same, regardless of whether that customer-user trades directly through their own login, or a sales-user trades on their behalf.

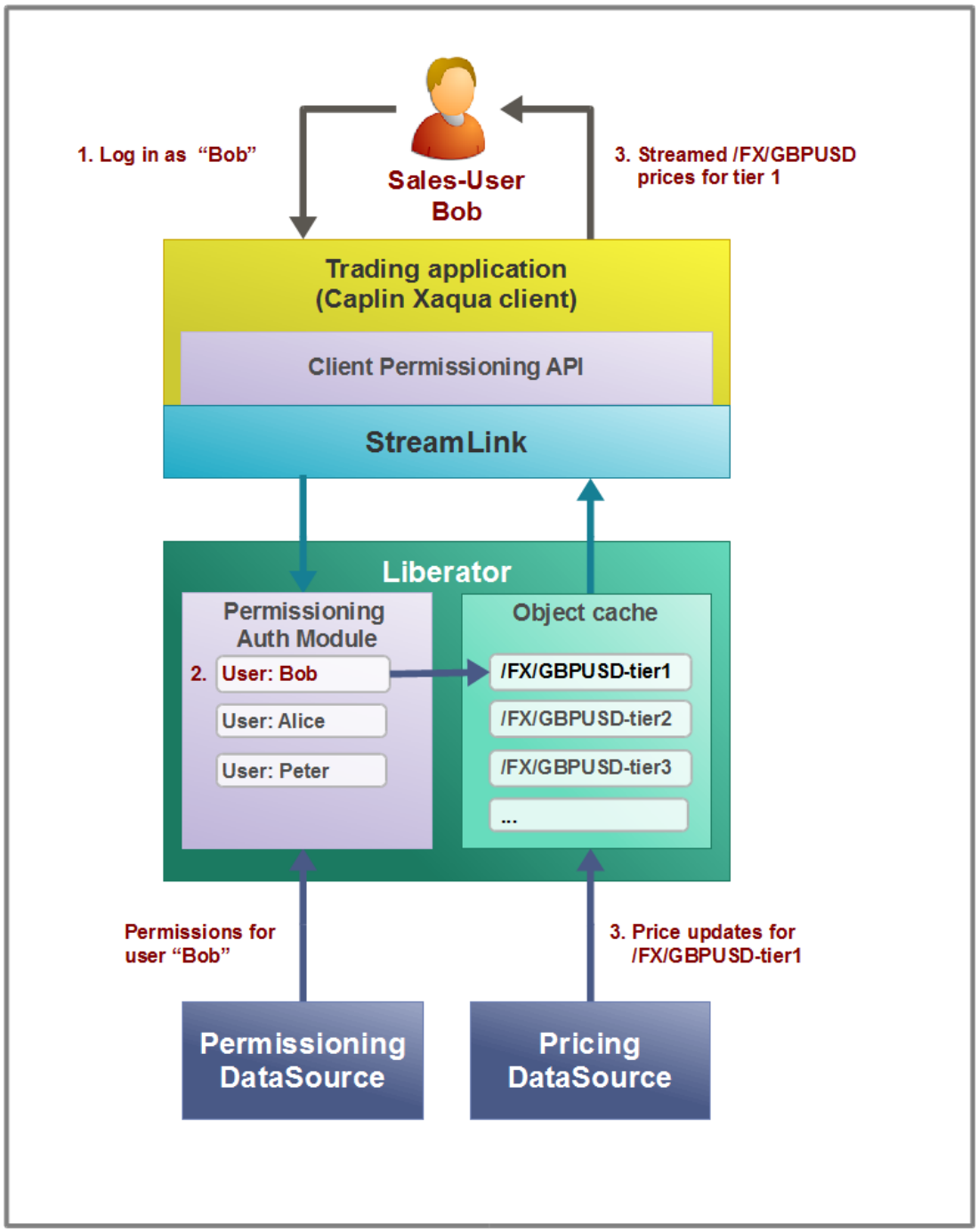
Caplin Xaqua's permissioning system supports TOBO functionality by:

- Granting or denying a sales-user permission to trade on behalf of a particular customer-user.
- Ensuring the sales-user has the right permissions to view and trade instruments on behalf of a customer-user.
- Ensuring the sales-user sees the same tiered prices the customer-user would see.

Note: If your Caplin Xaqua application is based on Caplin Trader, please contact Caplin Support to find out whether the TOBO facility described here is available in the Caplin Trader release that you are using, and for information on which aspects of TOBO are pre-configured in the Caplin Xaqua release that was delivered with Caplin Trader.

Typical TOBO processing

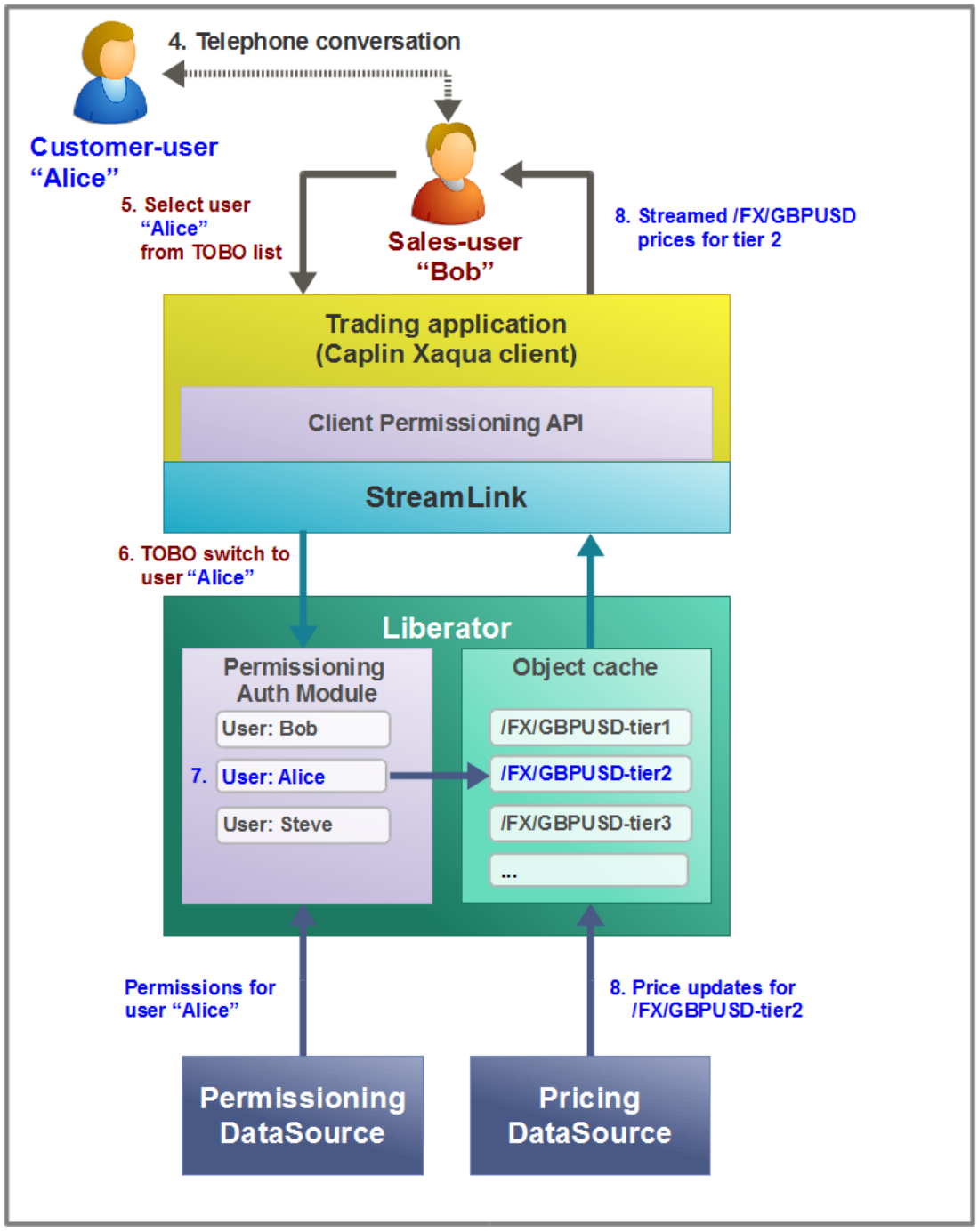
Here is what happens when a sales-user logs in and receives prices, when he is not trading on behalf of another customer:



Sales-user logs in and receives prices

1. A sales-user logs in to the trading application (a Caplin Xaqua client) as himself; his username is "Bob". The trading application asks the Liberator, via the Client Permissioning API, for the permissions that apply to user Bob. The Liberator obtains these permissions from its Permissioning Auth Module. (The Permissioning DataSource had previously sent these permissions to the Permissioning Auth Module, as shown in the diagram.)
2. The sales-user wishes to look at the prices for the FX currency pair GBPUSD. The Permissioning Auth Module verifies that he is allowed to do this. For this sales-user, the /FX subject namespace is mapped to tier1 prices, so the Liberator subscribes to /FX/GBPUSD-tier1, and requests this object from the Pricing DataSource that supplies /FX/GBPUSD-tier1.
3. The Liberator streams the price updates to the trading application. The sales-user therefore sees the prices for GBPUSD that reflect the price tier defined for him as a user.

Now consider the situation where the sales-trader trades on behalf of a customer:



Sales-trader trades on behalf of a customer

4. The sales-user Bob receives a telephone call from a customer-user, Alice, asking him to quote prices for /FX/GBPUSD and possibly trade this currency on her behalf.
5. When Alice trades for herself, she logs in to the trading application with username "Alice". However in this case, Bob, who is already logged in as himself, merely selects Alice's username from a drop down list in the trading application's UI and clicks on a "trade on behalf of" button.

6. This action sends a **TOBO switch message** to the Liberator's Permissioning Auth Module, requesting that Bob be allowed to trade on behalf of the customer-user whose login username is "Alice".
7. The Permissioning Auth Module checks that the sales-user Bob is permitted to trade on behalf of the customer-user Alice. (The Permissioning DataSource had previously sent these permissions to Permissioning Auth Module.) The Permissioning Auth Module also checks that Alice actually exists in the permissioning system. If one of these conditions is not met, Bob is not allowed to trade on behalf of Alice; the trading application receives notification of this and the prices remain tiered for Bob, just as they were before the attempted TOBO switch.

   If the Permissioning Auth Module's checks both succeed, the /FX subject namespace is mapped to tier2, which is the price tier that was allocated to Alice when her trading account was set up. The Liberator therefore subscribes to /FX/GBPUSD-tier2.
8. The Liberator streams the price updates for /FX/GBPUSD-tier2 to the trading application.
   Bob therefore sees the prices for GBPUSD that reflect the price tier defined for Alice, on whose behalf he is trading.
9. (Not shown in the diagram.) Bob could subsequently switch to trading on behalf of the customer-user "Peter", whose prices for /FX/GBPUSD are in yet another tier, tier3.
10. (Not shown in the diagram.) When Bob has finished executing trades for customer-users, he stops the TOBO session. This restores his streamed prices to those he would normally see when trading for himself (the prices for tier1).

Note that customer-users do not have to be set up to log in and trade directly; they could be set up as Liberator users but not as users of the trading application, so they must always trade through a sales-user.

TOBO permission modes

There are two modes of TOBO permissioning behavior. The Permissioning Auth Module may be configured, through the **TOBOPermissionMode** property in its *javaauth.properties* file, to exclusively use one mode or the other. In each mode, the sales-user can trade as a customer-user; the difference between the modes lies in how the permissions are determined when trading on behalf of the customer-user. Before you deploy your trading system for live operation, you must decide which mode to use and set up permissions accordingly, as explained in more detail in subsequent sections.

The TOBO modes are **SalesUser** and **SalesIntersectCustomerUser**.

SalesUser mode

In this TOBO mode, the sales-user's permissions are applied. That is, the permissions normally granted to the sales-user's Liberator session, which apply when he is not using TOBO, also apply when he is using TOBO. This typically means that in order to trade on behalf of different customer-users, the sales-user must be given the aggregate of all the customer-users' permissions.

Set this mode if it is acceptable to your business that few constraints should be applied to sales-users when they trade on behalf of customer-users.

Note that in this mode, because a sales-user has all the permissions of the set of customer-users, the sales-user could trade an instrument on behalf of a customer-user that the customer-user would not be permitted to trade. For example:

Customer-user-A can trade FX/GBP.* only.

Customer-user_B can trade FX/AUD.* only.

When the TOBO mode is **SalesUser**, a sales-user must be allowed to trade both FX/GBP.* and /FX/AUD.* so that he can trade on behalf of Customer-user-A and on behalf of Customer-user-B. But this also allows the sales-user to trade /FX/AUD.* on behalf of Customer-user-A, which this customer-user cannot do on their own.

> **Tip**: The **SalesUser** mode of TOBO is available in versions of the permissioning API from 4.5.9 onwards.

SalesIntersectCustomerUser mode

In this TOBO mode, permissions granted to both the sales-user and the customer-user apply. If either or both of the permission checks result in a Deny, the operation is denied. For the action to be allowed, both the sales-user *and* customer-user must be permitted. This is summarized in the following table:

SalesIntersectCustomerUser mode permission checks

Customer-user permission	Sales-user permission	Resulting TOBO permission is intersection (logical AND)
Deny	Deny	**Deny**
Permit	Deny	**Deny**
Deny	Permit	**Deny**
Permit	Permit	**Permit**

This mode of TOBO permissioning is most suited to situations where you wish to constrain what instruments a sales-user can trade on behalf of customer-users, in addition to independently constraining the instruments each customer-user can trade. For example, when there is a large number of customer-users, a particular sales-user cannot easily be aware of what trading restrictions apply to which customer-users. In this case, you can configure the permissions system in **SalesIntersectCustomerUser** mode, so the sales-user is only allowed to trade the instruments that the customer-user can trade.

You can also use this mode to constrain what a particular sales-user can do on behalf of *all* customers. For example, you could prevent a less experienced sales-user from trading in volatile instruments by setting a deny permission for the sales-user. When this sales-user trades on behalf of a customer-user who is permitted to trade volatile instruments, the intersection of permissions prevents the sales-user from trading such an instrument.

> **Tip:** The **SalesIntersectCustomerUser** mode of TOBO is available in the permissioning API from version 4.5.20 onwards.

Setting the TOBO permission mode

To set the TOBO permission mode you define a property called **TOBOPermissionMode** in the *javaauth.properties* file of the Permissioning Auth Module. For details, see the document **Caplin Xaqua: Installing Permissioning Components**.

Example of TOBO rules and permissions

Here is an example of how rules and permissions work when a sales-user Bob trades on behalf of a customer-user Alice. It shows how Bob is allowed to switch over to trade on Alice's behalf. The example applies to both TOBO modes.

In his trading application, Bob selects Alice's name from a list of customer-users and selects a button that requests permission to trade on her behalf. The application responds by sending a **TOBO switch message** to the Liberator:

TOBO switch message (RTTP Message)

Message Subject	Field: Username
/TOBOCHANGEUSER	Alice

By convention, the message subject is `/TOBOCHANGEUSER`, although you can specify a different name through configuration (for details, see “Permissioning Auth Module configuration for TOBO” in the document **Caplin Xaqua: Installing Permissioning Components**). The message contains a `UserName` field that identifies the customer-user on whose behalf the sales-user wants to trade; in this case, "Alice".

Object-map configuration in Liberator maps the username of the currently logged in user (the sales-user's Liberator login name) to the subject of the TOBO switch message. The required configuration is:

```
object-map  /TOBOCHANGEUSER  /TOBOCHANGEUSER/%u
```

For sales-user Bob, this changes the message subject to:

/TOBOCHANGEUSER/Bob

The Permissioning Auth Module then looks for a rule that matches the subject of the TOBO switch message (/TOBOCHANGEUSER/Bob). If there is no such rule, Bob will not be allowed to perform a TOBO switch.

A matching rule looks like this:

Rule matching the TOBO switch message

Subject Match	Field Match Criteria	Product Reference Field	Action	Namespace
/TOBOCHANGEUSER/%u		UserName	**ChangeTradeOnBehalfOfUser**	TradeOnBehalfOf

The Subject Match is /TOBOCHANGEUSER/Bob (the Permissioning Auth Module automatically substitutes the Liberator username "Bob" in the %u token). This matches the subject of the TOBO switch message, so the rule applies to the sales-user "Bob".

The Product Reference Field of the rule is the field in the TOBO switch message (UserName) containing the name of the customer-user on whose behalf the sales-user wishes to trade. So in this case, the Product Reference field refers to the customer-user "Alice".

The Permissioning Auth Module looks for a permission that

- is in the namespace defined by the rule (TradeOnBehalfOf),
- has the action defined by the rule (ChangeTradeOnBehalfOfUser),
- and is for the product called "Alice" (as defined by the Username field of the TOBO switch message).

The corresponding permission that allows the TOBO switch is:

Permission identified by the rule

Action	Product	Namespace	Authorization
ChangeTradeOnBehalfOfUser	Alice	TradeOnBehalfOf	**Allow**

This permission has the authorization "Allow", so Bob is allowed to trade on behalf of Alice. If the permission had the Authorization "Deny", or there was no permission matching the rule, Bob would not be allowed to trade on behalf of Alice.

Note that the sales-user must have only one permission for the action ChangeTradeOnBehalfOfUser for each customer-user they can trade on behalf of.

In addition to having permission to TOBO switch, Bob must also have sufficient permissions to actually view and trade the same instruments as Alice can.

When Bob subsequently trades on behalf of Alice the subject mappings that would apply to Alice if she traded for herself are applied to Bob, so Bob sees the same prices that Alice would. For more information on Subject Mapping see the Subject mapping 22 section.

Switching out of TOBO mode

When a sales-user has finished trading on behalf of a customer-user and does not intend to trade on behalf of yet another customer-user, the permissioning system must be returned to a state where the sales-user can trade on their own behalf, according to their own permissions and subject mappings.

This changeover is done by sending a TOBO switch message that specifies a username of "null":

TOBO switch message (mapped)

Message Subject	Field: Username
/TOBOCHANGEUSER/Bob	null

The same rule is triggered as in Example of TOBO rules and permissions 31:

Rule matching the TOBO switch message

Subject Match	Field Match Criteria	Product Reference Field	Action	Namespace
/TOBOCHANGEUSER/%u		Username	**ChangeTradeOnBehalfOfUser**	TradeOnBehalfOf

This time the rule selects a permission for the Product called null:

Permission identified by the rule

Action	Product	Namespace	Authorization
ChangeTradeOnBehalfOfUser	null	TradeOnBehalfOf	**Allow**

This permission has the authorization "Allow", which causes the Permissioning Auth Module to apply sales-user Bob's permissions and subject mappings.

ALL_ACTIONS permission

A permission with the special Action name ALL_ACTIONS allows a user to be permitted/denied all actions relating to the specified namespace and product/product set. This type of permission can be used to reduce the number of permissions that a sales-user must have in order to trade on behalf of a customer-user, as the following example shows

Example

Assume customer-users have one or more trading accounts. There may be many different accounts spread across a large number of customers, and each customer-user must be permissioned to trade on just their own accounts.

When customer-user "Alice" trades the instrument /FX/GBPUSD on the account "Account_1", the following RTTP message is sent to Liberator:

RTTP Message when Alice trades or Bob trades on behalf of Alice

Message Subject	Message Fields			
	MsgType	*Instrument*	*Account*	*{other fields}*
/FT/TRADE	Execute	/FX/GBPUSD	Account_1	...

The rule that matches this trade message is:

Matching rule

Subject Match	Field Match Criteria	Product Reference Field	Action Reference Field	Namespace
/FT/TRADE		Instrument	Account	Accounts

The permission applying to "Alice" that matches this rule is:

Alice's permission

Action	Product	Namespace	Authorization
Account_1	/FX/GBP.*	Accounts	Allow

When Alice trades the same instrument on a different account, "Account_2", the RTTP message is:

RTTP Message

Message Subject	Message Fields			
	MsgType	*Instrument*	*Account*	*{other fields}*
/FT/TRADE	Execute	/FX/GBPUSD	Account_2	...

The same rule matches this trade message as did previously:

Matching rule

Subject Match	Field Match Criteria	Product Reference Field	Action Reference Field	Namespace
/FT/TRADE		Instrument	Account	Accounts

The permission applying to Alice that this rule matches is:

Alice's permission

Action	Product	Namespace	Authorization
Account_2	/FX/GBP.*	Accounts	Allow

Note that Alice has a separate permission for each account. This gives the system administrators the flexibility to control which accounts Alice can trade on, by changing individual permissions.

Now if sales-user "Bob" is to be allowed to trade /FX/GBPUSD on Alice's behalf for these accounts, he needs the same permissions as Alice – one per account that Alice is allowed to trade on. But in a deployed system, the number of permissions needed for the sales-traders is (number-of-sales-users) x (number-of-accounts) x (number-of-products). Also, every time we add or remove a permission for a customer-user, we have to add or remove the equivalent permission for all sales-users.

Instead of defining another complete set of permissions just for Bob, we can give him a single ALL_ACTIONS permission:

Bob's ALL_ACTIONS permission

Action	Product	Namespace	Authorization
ALL_ACTIONS	/FX/GBP.*	Accounts	Allow

This permission allows Bob to perform all actions on products that match /FX/GBP.* in the Accounts namespace.

When the TOBO permission mode is **SalesIntersectCustomerUser**, and Bob trades on behalf of Alice, the intersection of Bob's ALL_ACTIONS permission with Alice's permissions (actions Account_1, Account_2), allows him to trade on both of Alice's accounts.

Using ALL_ACTIONS permissions in this way makes it easier to maintain permissioning data. The number of ALL_ACTIONS permissions required for the sales-users is just (number-of-sales-users) x (number-of-products), and whenever a permission is added or removed for a customer-user, there is no need to modify the sales-users' permissions.

Note: Only use the Action name ALL_ACTIONS in permissions.
Do not specify ALL_ACTIONS in the Action field of rules.

Tip: Explicitly defined permissions override the ALL_ACTIONS permission for the same combination of user, product, and namespace. So, for example, you could still deny Bob the ability to trade /FX/GBPUSD on Alice's behalf, by giving him Deny permission on the Product /FX/GBPUSD for the action Account, in the namespace Accounts.

ALL_ACTIONS when sales-users trade for themselves

Unless your sales-users are highly trusted, you should not give them ALL_ACTIONS permission when they trade for themselves. Set up separate rules and permissions to those used for trading on behalf of customer-users, (not using any ALL_ACTIONS permissions). Assign a different namespace to these rules and permissions, and make sure that different rules are matched by different fields in the RTTP message to those used in the customer-user messages. The following example illustrates this in the case where sales-user Bob trades /FX/GBPxxx instruments on his own behalf:

RTTP Message when Bob trades for himself

Message Subject	Message Fields			
	MsgType	*Instrument*	*SU_Account*	*{other fields}*
/FT/TRADE	Execute	/FX/GBPUSD	Account_1	...

Note that in this message, the field containing the account name is called *SU_Account*, not *Account* as in customer-user Alice's RTTP message.

The rule that matches this trade message is:

Matching rule

Subject Match	Field Match Criteria	Product Reference Field	Action Reference Field	Namespace
/FT/TRADE		Instrument	SU_Account	SU_Accounts

The Namespace for this rule is SU_Accounts (“Sales Users' Accounts”), not Accounts as in the rules applying to trades on behalf of customer-users.

The permission applying to "Bob" that matches this rule is:

Bob's permission

Action	Product	Namespace	Authorization
Account_1	/FX/GBP.*	SU_Accounts	Allow

Because Bob's RTTP message does not contain a *Account* field, the ALL_ACTIONS permission that Bob has to trade on behalf of other users does not apply when Bob trades for himself. The permission for Action Account_1 in namespace SU_Accounts allows Bob to trade /FX/GBPUSD, but he can only trade other products, such as /FX/AUD.* if he has the relevant other permissions.

ALL_ACTIONS in permission mode SalesUser

A sales-trader should not be given an ALL_ACTIONS permission when the TOBO permission mode is **SalesUser**, because in this mode, the permission would allow the sales-trader to do anything on behalf of any user.

Enabling/disabling TOBO for a sales-user

It may be desirable to turn TOBO permissions on or off quickly for a given sales-user, without removing the permissions that name the customer-users the sales-user can trade on behalf of (so these permissions don't have to be remembered and added again when TOBO is turned back on). This can be done by creating an extra rule that is triggered when the Liberator receives a TOBO switch message. Here is an example of how to do this.

Assume the sales-user is called Bob. Bob wishes to trade on behalf of customer-user Alice.

As in Example of TOBO rules and permissions 31, the TOBO switch message received by the Permissioning Auth Module has the (mapped) subject /TOBOCHANGEUSER/Bob.

TOBO switch message with mapped subject

Message Subject	Field: Username
/TOBOCHANGEUSER/Bob	Alice

In addition to the rule shown in Example of TOBO rules and permissions 31, there is another rule that matches this message subject:

Rule matching the TOBO switch message

Subject Match	Field Match Criteria	Product Reference Field	Action	Namespace
/TOBOCHANGEUSER/%u		ALL_PRODUCTS	**ToboOn**	ToboEnabled

This rule is in a different namespace (see the Tip below) and refers to ALL_PRODUCTS, so the Permission Auth Module searches all permissions for the Action ToboOn in the ToboEnabled namespace. The (single) matching permission is:

Permission identified by the rule

Action	Product	Namespace	Authorization
ToboOn	ALL_PRODUCTS	ToboEnabled	**Allow**

This permission has the authorization "Allow", so Bob is allowed to trade on behalf of sales-users, including Alice.

If the permission had the Authorization "Deny", or there was no permission matching the rule, Bob would not be allowed to trade on behalf of anyone.

Note that the TOBO switch message triggers *two* rules; the rule shown here that permits or denies trading on behalf of anyone, and the rule shown in Example of TOBO rules and permissions 31 that permits or denies trading on behalf of a specific customer-user. The sales-user is only allowed to trade on behalf of the customer-user if *both* rules refer to permissions that have Allow authorization.

> **Tip:** The permission namespace (ToboEnabled) should be different to that used for permissioning the customer-users a sales-user can trade on behalf of (ToboSwitchPermissions). This is because permissions used for different purposes should be isolated from each other by namespace to prevent unintended interactions between them.

Enabling/disabling TOBO for all sales-users

The scenario previously described assumes there is a separate ToboOn permission for each sales-user in the system.

A more sophisticated way to manage enabling and disabling TOBO would be to put all sales-users in a permissioning group, and define a permission at the group level that enables TOBO for all these sales-users. To then disable TOBO for an individual sales-user, you can define for that user a ToboOn permission that has Deny authorization – this overrides the group level permission. Furthermore, you can disable TOBO for all sales-users by changing the group level ToboOn permission to Deny.

Using the %t substitution token

The %t substitution token can be used in the Product expression of a permission. It matches the username of the logged in user, or the username of any user that the logged in user can trade on behalf of, and is therefore intended for use in TOBO related permissions.

The following example shows how %t is used in a permission, when a sales-user attempts to view a product on behalf of a customer-user. (Compare this to the simple example using %u shown in Using a substitution token in a permission 17.) Because the user is attempting to view a product, the default rule is applied. The following permission matches the default rule; it is for the action called VIEW in the default namespace:

Permission that applies

Action	Product	Namespace	Authorization
VIEW	/PRIVATE/**%t**/FX/USDGBP		Allow

If sales-user Bob is authorized to trade on behalf of customer-users Steve and Alice, this permission applies to the following RTTP message subjects:

/PRIVATE/Bob/FX/USDGBP

/PRIVATE/Steve/FX/USDGBP

/PRIVATE/Alice/FX/USDGBP

Bob can therefore view USDGBP prices for himself, and on behalf of Steve and Alice.

If Bob is not authorized to trade on behalf of Paul (that is, he cannot TOBO switch to Paul), Bob cannot view data with the subject

/PRIVATE/**Paul**/FX/USDGBP

This is because the subject does not match the VIEW permission (the %t in the permission's Product string will never be replaced by the string "Paul").

Note that while the use of the %t token in permissions permits Bob to act on behalf of Steve and Alice, it does not permit Steve or Alice to act on behalf of Bob, nor does it permit Steve to act on behalf of Alice or vice-versa.

For a user who is *not* permitted to trade on behalf of any other users, %t behaves like %u; it just matches the user's Liberator login username.

A typical use-case

The %t token could be used in permissions to grant a sales-user access to a customer-user's trade history. This allows the sales-user to see the trade history data securely. Using `%t` also avoids having to explicitly grant the sales-user a separate VIEW permission for each customer-user.

Note: For the %t token to work, you must set the configuration properties **ToboSwitchAction** and **ToboSwitchNamespace** in the Permissioning Auth Module's *javaauth.properties* file.
For details, see the document **Caplin Xaqua: Installing Permissioning Components**.

Note: Never use %t in the Subject Match definition of a permissioning Rule, or to map an object using **object-map** in the Liberator configuration. This is because Liberator does not recognize the %t token, and can only map objects for the logged in user and not the user they are trading on behalf of.

It is only the Permissioning Auth Module that recognizes the %t token when it is placed in a product permission, where it matches the token to the logged in user, or to the user they are trading on behalf of.

6 Further reading

If you would like to know how to create a custom Permissioning DataSource adapter, or how to add Permissioning to Caplin Trader, then the following documents provide this information. You may also be interested in reading some of the other Related documents.

How To Create A Permissioning DataSource Adapter

A Permissioning DataSource adapter is required to integrate Caplin Xaqua with a Permissioning System. The document **Caplin Xaqua: How To Create A Permissioning DataSource Adapter** describes how to create a custom Permissioning DataSource adapter by writing an application that uses the Permissioning DataSource API. The document also discusses the Demo Permissioning DataSource that is provided with the reference implementation of Caplin Trader from release 1.2.8.

How To Add Permissioning At The Client

The appearance and behaviour of Caplin Trader can be tailored to match the permissions of the currently logged in user. You will find further information about how to do this in the document **Caplin Trader: How To Add Permissioning At The Client**.

6.1 More about permissioning for TOBO

For more information about setting up permissioning to support TOBO, refer to the following documents:

- **Caplin Xaqua: How To Create A Permissioning DataSource Adapter**

  Contains examples of how to set up TOBO-related permissions in a Permissioning DataSource Adapter.

- **Caplin Xaqua: Installing Permissioning Components**

  Describes the javaauth properties that determine how TOBO is controlled by Liberator's Permissioning Module.

Note: If you are not using a Caplin Xaqua client that is based on Caplin Trader, you may need to change some of these properties to make TOBO work correctly in your client environment.

- **Caplin Trader: How To Add Permissioning At The Client**

  Explains how to use TOBO permissioning in a Caplin Xaqua client that is based on Caplin Trader.

- **Caplin Trader: Permissioning Configuration XML Reference**

  Describes XML-based TOBO configuration that is needed for Caplin Trader applications.

7 Glossary of terms and acronyms

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

Term	Definition
Action	The interaction that a user can have with a **product**.
API	Application Programming Interface
Caplin Trader	A web application framework for constructing browser-based financial trading applications (**Caplin Trader applications**).
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 can also be short for a **Caplin Xaqua system**.
Caplin Xaqua client	A client desktop or web application that interfaces with **Caplin Xaqua** to deliver multi-product trading to end-users.
Caplin Xaqua system	A single-dealer platform that is built using **Caplin Xaqua**.
Customer-user	An end-user of a **Caplin Xaqua client** who instructs a **sales-user** to trade on their behalf.
DataSource API	An **API** and underlying code library that allows **DataSource applications** to communicate with each other.
DataSource adapter	A **DataSource application** that acts as the interface between **Caplin Xaqua** and an external (non-Caplin) system, exchanging data and/or messages with that system.
DataSource application	A **Caplin Xaqua** application that uses the **DataSource API** and code library to communicate with other Caplin Xaqua applications.
DataSource for Java	A **DataSource API** written in Java.
Default rule	The built-in rule that is used to determine permissions to view data. See Rules 9 in Permissioning Concepts 7.
Demo Permissioning DataSource	The Demo Permissioning DataSource is an example of a **Permissioning DataSource** that gets its permissioning data from an XML file.
Global context	An object at the **Permissioning Auth Module**. The global context allows custom **subject mappers** to access data that is common to all subject mappers and **users**.
Group	A logical grouping of zero or more **users** and other groups, such that each group can be assigned zero or more **permissions**.
Liberator	A real-time financial internet hub that delivers trade messages and market data to and from subscribers over any network that supports web traffic.
Master	When permissioning data is sent to Liberator from multiple **Permissioning DataSource** adapters, one of the Permissioning DataSource adapters is designated the master, and the others are designated as **slaves**.
Permission	Determines whether an **action** on a **product** will be allowed or denied.

Term	Definition
Permissioning Auth Module	One of several authentication modules that are supplied with **Caplin Xaqua**.
Permissioning DataSource	A **DataSource adapter** that acts as the interface between **Caplin Xaqua** and your **Permissioning System**.
Permissioning DataSource API	The **API** that a **Permissioning DataSource** uses to send permissioning data to **Liberator**.
Permissioning System	The source of the permissioning data that you want to integrate with **Caplin Xaqua**.
Price tiering	A feature of trading systems whereby different customers see different prices. For example, better prices could be given to customers who trade high volumes, or to customers who trade more regularly. Each user of the system is allocated to a price tier that determines what prices they will see.
Product	In permissioning documentation (including this document) a "product" is any entity on which a **User** may be assigned **permissions** (including financial instruments). In other **Caplin Xaqua** and **Caplin Trader** documentation a "product" is a term that refers only to a financial instrument.
Role	Roles determine whether a **Permissioning DataSource** is designated as a **master** or **slave** Permissioning DataSource.
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.
RTTP message	A message sent between a **Caplin Xaqua client** and **Liberator**, using **RTTP**.
Rule	Rules link **permissions** to user interactions, and are used by **Liberator** to decide which of the many permissions that have been defined will apply when a **user** attempts to interact with a **product**.
Sales-user	An end-user of a **Caplin Xaqua client** who takes instructions from a **customer-user** to trade on their behalf.
SDK	Software Development Kit
Slave	When permissioning data is sent to Liberator from multiple **Permissioning DataSource** adapters, one of the Permissioning DataSource adapters is designated the **master**, and the others are designated as slaves.
Subject mapper	A subject mapper is a Java class that resides at the **Permissioning Auth Module**. A subject mapper can modify the message that Liberator receives when an end-user attempts to view or trade a product, and can be used to provide preferential data to selected **users**.
TOBO	**Trading On Behalf Of**
TOBO switch message	An **RTTP message** that is sent from a **Caplin Xaqua client** to a **Liberator**, requesting that the current user (a **sales-user**) be allowed to trade on behalf of a different user (a **customer-user**). Also see **Trading On Behalf Of**, Typical TOBO processing 26, and Example of TOBO rules and permissions 31.

Term	Definition
Trading On Behalf Of	This is a facility that allows a user who is logged in to a **Caplin Xaqua client** to execute trades on instruction from a customer (for example, the customer may give instructions by telephone). The logged-in user (**sales-user**) trades on behalf of the customer (**customer-user**). Also see Permissioning for TOBO 25.
UI	User Interface
User	An end-user of a **Caplin Xaqua client** application such as **Caplin Trader**.

Contact Us

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