CAPLIN

Transformer 4.4

Administration Guide

July 2010

1 Preface

1.1 What this document contains

This document describes the Caplin Transformer and its place in the Caplin real-time data architecture. It includes instructions on how to install and configure the Caplin Transformer.

For information on how to create custom modules to add value to incoming market data, please refer to the associated **Caplin Transformer Module SDK** documents.

1.2 Who should read this document

This document is intended for people who need to install, configure and maintain the Caplin Transformer. Administrators are assumed to have a working knowledge of Solaris or Linux procedures.

1.3 Typographical conventions

This document uses the following typographical conventions to identify particular elements within the text.

Type	Use
Arial Bold	Function names and methods.
	Other sections and chapters within this document.
Arial Italic	Parameter names and other variables.
Times Italic	File names, folders and directories.
`Courier`	Program output and code examples.
❖	Information bullet point
■	Instruction

1.4 Acronyms and glossary

Active DataSource	A DataSource application that is configured to respond to DataSource Request messages received from one or more of its DataSource peers, by keeping track of which objects have been requested, and sending to the requesting peers updates just for those objects. This improves performance by reducing network bandwidth.
API	Application Program Interface, the method by which a programmer writing an application program can make requests of the operating system or another application.
Applet	A small program that can be sent along with a Web page to a user. Java applets can perform interactive animations, immediate calculations, or other simple tasks without having to send a user request back to the server.
Authentication	Permitting a particular user to log in to Caplin Liberator in order to access streaming data. Also known as entitlement.
Authorisation	Permitting particular data to be viewed.
Base64	An encoding scheme which allows arbitrary binary data to be transferred over a text protocol, such as an HTTP request.
Caplin Liberator	A suite of software applications and components for publishing real-time information using RTTP protocol over IP networks. Caplin Liberator collects real-time data from one or more sources and redistributes it to suitably permissioned RTTP subscribers.
Caplin WebSheet	WebSheet combines an RTTP client with advanced DHTML scripting to display live market data in browsers, turning a web page into a dynamic, real-time quote terminal. It provides a configurable container that emulates many of the characteristics of a professional trader workstation.
ComStock	The Standard and Poor's market data platform.

CRC	Cyclic Redundancy Checksum, a method for detecting transmission errors.
CSP	Standard and Poor's ComStock datafeed handler.
CUSIP	A nine-character number that uniquely identifies a particular security. CUSIP is an acronym for the Committee on Uniform Securities and Identification Procedures, the standards body which created and maintains the classification system.
DACS	A system for administering market data and permissioning trading floor, enterprise and Internet users to data, transactions, and other Reuters services.
DataSource	A network protocol that enables most Caplin and RTTP-related products to communicate with each other.
DataSource API	A transmission path which links the architectural elements that use the DataSource protocol.
DataSource Peer	Any remote application that can connect to the DataSource API. All Caplin Transformers are DataSource peers.
DataSource SDK	The DataSource SDK (Software Development Kit) is a library of functions that enables developers to create their own DataSource peer.
DDE	Dynamic Data Exchange, a set of commands and message formats that allows information to be shared or communicated between DDE-aware Windows applications.
DER	A file format for encryption keys, used by Caplin Liberator for public keys.
DSN	A name for an ODBC data source.
DTD	Document Type Definition, a specification that accompanies a document and identifies what the tags are that separate paragraphs and how each is to be processed.

Entity	Permission for a particular field to be viewed. An entity can also define values that a particular field must hold before it can be viewed.
ETX	End of Transmission.
FD0	ComStock's weekly symbol information feed.
FID	A Field Identifier (see the entry for Field below), specific to Reuters data feeds.
Field	A data element of an object, identified by a field name or field number and with a data value of a string.
Firewall	A firewall is a set of related programs on a network gateway server that protects the resources of a private network from users from other networks. Firewalls can screen requests to make sure they come from acceptable addresses and allow mobile users to gain remote access to the private network.
GMT	Greenwich Mean Time.
HNAS	Reuters Historical News Application Server-a session service that supplies historical news.
HTTP	Hypertext Transfer Protocol, the set of rules for exchanging files on the World Wide Web.
HTTPS	Otherwise known as Secure Sockets Layer or SSL, HTTPS is a version of HTTP used for managing the security of a message transmission on the Web. HTTPS uses the public-and-private key encryption system, which also includes the use of a digital certificate.
IIS	Internet Information Services, a Windows-based web server.
ILA	IDN Logical Address, a number uniquely identifying a Reuters SF+ server or keystation.
Image	A complete set of data currently held on a data source.

Java	An object-oriented programming language designed for use in the distributed and heterogenous environment of the Internet.
Javadoc	A tool that parses the declarations and documentation comments in a set of Java source files and produces a set of HTML pages describing the classes, inner classes, interfaces, constructors, methods, and fields.
JDBC	An API that enables developers to access most tabular data sources from the Java programming language. It can connect to SQL databases, spreadsheets and flat files.
JDK	Java Development Kit, containing tools, runtimes and APIs for developers writing, deploying, and running applets and applications in Java.
LAN	Local Area Network, a group of computers and associated devices that share a common communications line and typically share the resources of a single processor with applications and data storage that are shared by multiple computer users.
MarketFeed	A high performance system delivering real time market data, news, options, fundamental data, market alerts and charting information to back office system through a simple API.
MarketLink	Reuters' feed protocol for distribution of real-time market data.
Object	There are several types of RTTP object: Directory, Page, Record, News headline, News story and Chat object. Each type is identified by a three digit number.
ODBC	Open Database Connectivity (ODBC) is an open standard application programming interface for accessing a database. By using ODBC statements in a program, you can access files in a number of different databases, including Access, dBase, DB2, Excel, and Text.
Page	A fixed format text array used by older financial systems, descended from dumb terminal displays of 80 x 25 characters.
Parameter	A data element of an object, identified by a field name or field number and with a data value of a string. Same as "field".

Parser	A program that receives input in the form of program instructions, markup tags or some other interface and breaks them up into parts (for example, objects, methods and their attributes) that can then be managed by other programs.
PDF	Portable Document Format. PDF documents preserve the exact look and content of the originals, complete with fonts and graphics.
PriceMaster	PriceMaster controls, converts, permissions and distributes real-time market data. It collects information from a variety of data feeds for onward publishing to all the traditional quote vendors.
Protocol	A standard that defines the way in which data is passed between two or more pieces of computer equipment over a telephone line or other communications link. Two pieces of equipment must be using the same protocol in order to communicate.
Proxy server	A server that acts as an intermediary between a workstation user and the Internet to ensure network security and give improved administrative control.
RIC	Reuters Instrument Code, a way of uniquely identifying all instruments in all asset classes upon which Reuters publishes prices
RSA	A public-key encryption technology developed by RSA Data Security, Inc. The acronym stands for Rivest, Shamir, and Adelman, the inventors of the technique. RSA uses an algorithm to convert input data into something unrecognisable (encryption), and then convert the unrecognisable data back to its original form (decryption).
RTS	Caplin Sentinel Script. RTS files hold details regarding what events report and who to report it to..
RTTP	RTTP (Real Time Text Protocol) is a web protocol developed by Caplin Systems Ltd that implements advanced real-time streaming for almost all types of textual information, including logical records, news and free-format pages.

Sink Distributor	Software responsible for responding to sink requests, maintaining communications with source services and distributing SSL messages to sink clients.
SQL	SQL (Structured Query Language) is a standard interactive and programming language for getting information from and updating a database. Queries take the form of a command language that lets you select, insert, update, find out the location of data, and so on.
SSL	The Secure Sockets Layer (SSL) is a commonly-used protocol for managing the security of a message transmission on the Internet. SSL uses the public-and-private key encryption system, which also includes the use of a digital certificate.
SSL (2)	Source Sink Library (SSL), Reuters interface for Triarch.
StreamLink for Excel	A mechanism to communicate with Caplin Liberators using the RTTP protocol. The user enters a StreamLink for Excel function in a cell which identifies what particular piece of data should be displayed. Whenever the value changes, Caplin Liberator sends an updated value to be displayed within that cell.
STX	Start of Transmission.
Symbols	The letters used to uniquely identify a financial instrument (e.g. the symbol for Microsoft's common stock on the Nasdaq market is MSFT). Many symbol naming conventions (symbologies) exist, but each is applied consistently in each market, and one symbology is used across all North American equity markets.
TCP/IP	Transmission Control Protocol/Internet Protocol is the basic communication language or protocol of the Internet. It can also be used as a communications protocol in a private network (either an intranet or an extranet).
Thread	A concurrent process running in a single instance of a program.
Triarch	The Reuters market data platform.

TS1	"Time Series" data provided from Reuters, used to create charts. TS1 data is historical price information stored in packed-page format.
UDP	UDP (User Datagram Protocol) is a communications protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol.
UMDP	Universal Market Data Protocol, a protocol which enables distributors of real time market data to provide a common interface for data vendors who wish to obtain this data.
XML	XML (Extensible Markup Language) contains markup symbols to describe the contents of a page or file. An XML file can be processed purely as data by a program or it can be stored with similar data on another computer or displayed. XML is "extensible" because, unlike HTML, the markup symbols are unlimited and self-defining.
XML tags	The sequence of characters or other symbols that you insert at certain places in a file to indicate how the file should look when it is printed or displayed or to describe the document's logical structure. Tags that have other tags within them are called parent tags; those within are called child tags.

1.5 Feedback

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

Please email your thoughts to **documentation@caplin.com**.

2 Overview

2.1 Overview

The Caplin Transformer receives large volumes of raw real-time market data and republishes it as value-added data in real time, either to a database or the DataSource API (a transmission path which enables most Caplin and RTTP-related products to communicate with each other), or to generic output such as ODBC, XML or message queues. As it is an active data source, Caplin Transformer is also capable of receiving requests for data over the DataSource API.

This is achieved by a set of business modules that implement the specific algorithms required. Examples of the calculations and other processing Caplin Transformer can perform include:

- ❖ a module can calculate the top 10 shares in terms of price, yield, number of transactions per hour and so on;
- ❖ another module can log incoming data to a database, and if a corrected or marked trade appears in a feed determine whether that is passed on or not;
- ❖ it can simplify updates so that if multiple updates are contained within the update packet, only the last (and hence most significant) is sent out;
- ❖ it can perform predetermined tasks according to market conditions, such as sending initialisation messages when markets open and end-of-day calculations when they close;
- ❖ Caplin Transformer can also process suspensions in trading and determine what to do with the data in alignment with market rules.

These modules are configured by editing plain text *.conf* configuration files.

2.2 System architecture

Figure 2-1 shows how Caplin Transformer fits into the Caplin real-time data architecture.

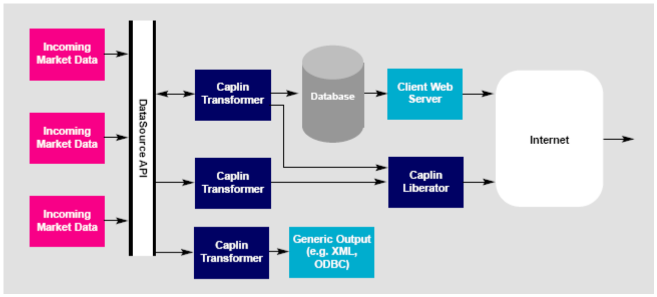

Figure 2-1: Caplin Transformer system architecture

Several market data sources contribute information onto the DataSource API. This data is accepted by the Caplin Transformer, which process it, and can then output it in three ways:

- back to the DataSource API, where it can be picked up by any suitably configured DataSource-enabled application.
- to a database, where information can be kept for historical analysis and interrogated by a client web server.
- to a Caplin Liberator, which can then publish the information over the Internet.

2.3 Internal architecture

Figure 2-2 shows the internal structure of the Caplin Transformer.

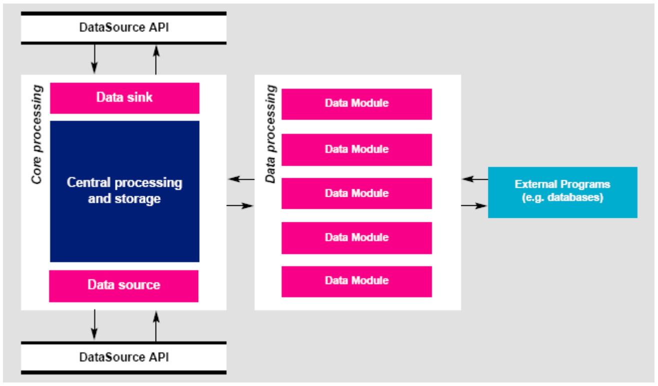

Figure 2-2: Caplin Transformer internal architecture

- The data sink element of Caplin Transformer extracts relevant data from the DataSource API.
- Central processing sends the data to the modules, which perform the specific algorithms that adjust the data.
- Amended data is then either output to external applications, such as databases for storing historical records, or back to the core for publishing back onto the DataSource API through the data source.

2.4 Caplin Transformer and DataSource peers

As well as being sources of data, products attached to the DataSource API can be destinations for data sent from Caplin Transformer, as illustrated in Figure 2-3 below.

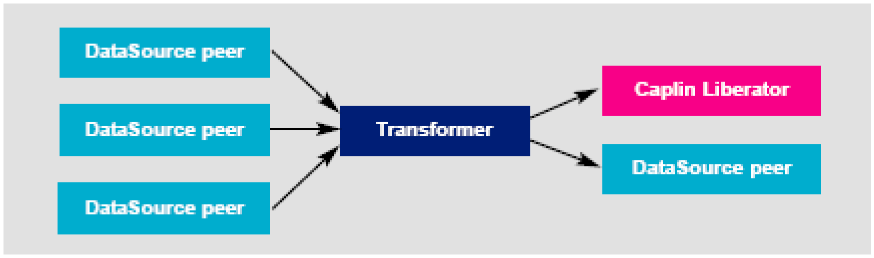

Figure 2-3: Caplin acting as a data source and data sink

A DataSource peer is a remote application that uses Caplin's DataSource protocol to attach to the DataSource API and both send and receive data.

As this means the link between DataSource peers is bidirectional, the true relationship between the elements is shown in Figure 2-4 below

Note: *All Caplin Liberators include DataSource functionality, and can operate in the same way as DataSource peers..*

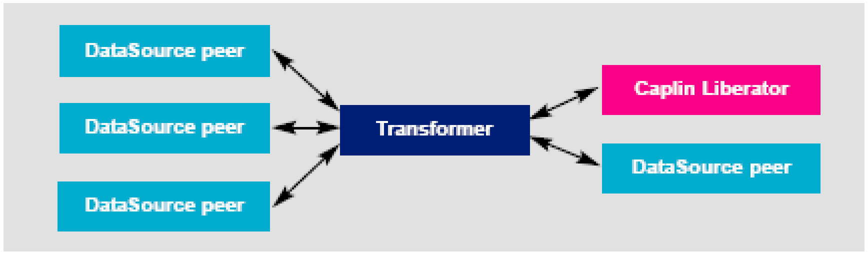

Figure 2-4: Bidirectional links between DataSources

Objects can be requested from individual DataSource peers or groups of peers.

2.5 What's new in Transformer?

New since version 4.1

This section summarizes the additions and improvements to Transformer since version 4.1.

Active DataSource

Transformer is now always an active DataSource.

This means that it will respond to DataSource Request messages received from one or more of its DataSource peers, by keeping track of which objects have been requested, and sending to the requesting peers updates just for those objects. This improves performance by reducing network bandwidth.

As a consequence of this enhancement the ***rtas-active*** configuration option, that enabled Caplin Transformer to act as an active data source, has been withdrawn.

New Transformer modules

Several new processing modules are available for Transformer:

❖ A **pipeline module** that allows business rules to be written in the LUA scripting language.

❖ A **data formatting module** that allows other modules to call standard data formatting functions.

❖ A **clustering module** that permits the clustering of Transformers. Clustering allows data to be replicated across Transformers, both through configuration and programmatically.

❖ An **autopublishing module** that allows the Transformer to automatically broadcast updates to data, even though it is an active DataSource.

New in version 4.4

This section summarizes the additions and improvements to Transformer specifically in version 4.4.

Performance enhancements

The performance of Transformer has been improved.

DataSource load balancing

The load balancing algorithm used to spread requests and updates across multiple DataSource peers has been improved. Each request is now directed to the peer with the smallest number of existing subscriptions. See "Specifying alternative DataSource peers" on page 30.

New subscription API

One of the performance improvements allows updates to be processed by multiple threads. A new subscription management API has been written to take explicit advantage of this performance enhancement. The new API is documented in the doxygen reference documentation supplied with the Transformer installation kit. The example Transformer module supplied by Caplin illustrates how to use the new API.

Legacy subscription API is deprecated

The previous Transformer subscription API is now deprecated, and is marked as such in the reference documentation. It will continue to be supported, but using it may cause performance degradation.

Note: *If you have existing custom Transformer modules that use the legacy subscription API and their performance becomes of concern, it is recommended that you adapt them using the new API.*

Logging configuration item add-log-cycle now add-log

The configuration item for configuring Transformer log files, ***add-log-cycle***, is now called ***add-log***. Existing configuration files that refer to ***add-log-cycle*** will continue to work, but since ***add-log-cycle*** is deprecated, any configuration changes and new configuration entries should use ***add-log***. See page 41.

Logging configuration item debug-level now log-level

The ***debug-level*** configuration item, which determines the errors and events that are reported to the log files when Transformer is operating, is now called ***log-level***. Existing configuration files that refer to ***debug-level*** will continue to work, but since ***debug-level*** is deprecated, any configuration changes and new configuration entries should use ***log-level***. See page 75.

3 Installing and running Transformer

3.1 System requirements

The Caplin Transformer runs on the following operating systems:

- Linux;
- Solaris.

Precise hardware requirements are dependent on your specific data volumes and processing loads. Functional analysis of your needs must be carried out before hardware recommendations can be made.

Two machines are required for failsafe purposes.

3.2 Installing Caplin Transformer

Perform the following steps to install Caplin Transformer on a Linux or Solaris platform:

- Unpack the kit into a suitable directory (for example */opt* or */usr/local*) and create a link to this new directory.

Examples:

Linux:

```
$ cd /usr/local
$ tar xzf /tmp/TRANSFORMER-[VERSIONNUMBER]-i686-pc-linux-gnu.tar.gz
$ ln -s TRANSFORMER-[VERSIONNUMBER] TRANSFORMER
```

Solaris:

```
$ cd /opt
$ uncompress /tmp/TRANSFORMER-[VERSIONNUMBER]-sparc-sun-solaris2.8.tar.Z
$ tar xf /tmp/TRANSFORMER-[VERSIONNUMBER]-sparc-sun-solaris2.8.tar
$ ln -s TRANSFORMER-[VERSIONNUMBER] TRANSFORMER
```

You should now have a directory structure something like this:

```
/opt/TRANSFORMER-[VERSIONNUMBER]/bin(for binary programs)
/opt/TRANSFORMER-[VERSIONNUMBER]/doc(for documents and examples)
/opt/TRANSFORMER-[VERSIONNUMBER]/etc(for startup and configuration)
/opt/TRANSFORMER-[VERSIONNUMBER]/var(for log files)
/opt/TRANSFORMER-[VERSIONNUMBER]/lib(for modules)
/opt/TRANSFORMER-[VERSIONNUMBER]/src(for sample source code )
/opt/TRANSFORMER-[VERSIONNUMBER]/include(for include files used for
compiling code)
/opt/TRANSFORMER -> TRANSFORMER-[VERSIONNUMBER]
```

- Edit the file *etc/transformer* and change the line TRANSFORMER_ROOT to point to the directory in which you installed Caplin Transformer (for example */usr/local/TRANSFORMER*).

- If you want Caplin Transformer to start automatically on boot-up then create a link from your startup directory.

Example (Red Hat Linux):

```
$ cd /etc/rc.d/rc3.d
$ ln -s /usr/local/TRANSFORMER/etc/transformer S99transformer
```

On other systems using SYSV startup scripts this process should be similar.

Note: *The parameter S99transformer tells the system to run the script last. The transformer part of the parameter name must match the application binary.*

3.3 Defining Transformer as a DataSource peer

You need to give Transformer an identifier in order for any connected peers to know which updates should be sent to it.

- Use the following parameters in the configuration file *transformer.conf* to give a unique identifier for your Transformer.

datasrc-name	The name of the Transformer, and how DataSource peers will identify it. See page 52. This name can be overridden by putting a value in the ***local-name*** option of the ***add-pee***r entry (see page 54). %a represents the application name, %h the name of the host machine.

Example:

```
datasrc-name      testsrchost8
```

datasrc-id	ID number of this Transformer. See page 52 This ID can be overridden by putting a value in the ***local-id*** option of the ***add-peer*** entry (see page 54), in which case it must match the ***remote-id*** given in the ***add-peer*** entry in the remote DataSource's configuration.

3.4 Connecting to DataSource peers

- Use the following parameters in the configuration file *transformer.conf* to identify peers and configure how they connect.

datasrc-interface	Network interfaces to listen for connections from DataSource peers. See page 53.
datasrc-port	Network port to listen for connections from DataSource peers. The default of 0 means that no connections can be made to the Transformer. See page 53.
datasrc-sslport	Network port to listen for SSL connections from DataSource peers. The default of 0 means that no SSL connections can be made to the Transformer. See page 53 and also "Making SSL connections with DataSources" on page 34.
add-peer	Identifies a DataSource peer which can be communicated with. This entry includes the ID number and name of the DataSource peer, and the ID number and name of Transformer, which is sent to the DataSource peer in order to identify your Transformer. See page 54.

Defining datasource peer connections

For each DataSource peer that communicates with the Transformer specify an **add-peer** entry in *transformer.conf*. If the DataSource initiates the connection (so the Transformer accepts the connection request), the entry must include a **remote-id** option and optionally a **remote-name** option, as in the following example.

```
add-peer
      remote-name       DataSource_1
      remote-id         1
      ...
end-peer
```

The DataSource peer's configuration should include:

- A **datasrc-id** that matches the **remote-id** in the Transformer's **add-peer** configuration entry, and an optional **datasrc-name**.

- An **add-peer** entry containing **addr** and **port** options. These define the Transformer address and port to which the DataSource peer should send connection requests.

```
datasrc-name            DataSource_1
datasrc-id              1
...
add-peer
      addr              <<Transformer addr>>
      port              <<Transformer port>>
      ...
end-peer
```

When DataSource connects to the Transformer, the **datasrc-name** defined for the DataSource will override the **remote-name** defined in the Transformer's **add-peer** section.

If the Transformer initiates the connection to the DataSource, then specify the configuration the other way round. The **addr** and **port** options must be in the Transformer configuration and specify the connection address and port for the DataSource. The DataSource configuration contains **remote-id** and **remote-name** settings corresponding to the Transformer's **datasrc-id** and **datasrc-name**.

Changing the Transformer's identity in peer connections

When a connection is made between a DataSource peer and a Transformer, they exchange ids and names. The Transformer's id and name, as defined in **datasrc-id** and **datasrc-name**, are sent to the DataSource peer. Using the **local-id** and **local-name** options of the **add-peer** entry you can override the Transformer's id and name for that particular peer, as in the following example.

```
datasrc-name            Transformer_A
datasrc-id              2
...
add-peer
      local-name        Transformer_A1
      local-id          3
      remote-name       DataSource_1
      remote-id         1
      ...
end-peer
```

When a connection is made to the DataSource peer, it is sent the **local-id** and **local-name** rather than the Transformer's **datasrc-id** and **datasrc-name**. This allows you to give the Transformer different identities as seen by different DataSource peers.

Multiple connections to a DataSource

You may want to configure more than one connection to a single DataSource, for example to improve performance by utilizing multiple DataSource threads (see “Improving performance using DataSource threads” on page 36). To do this you must modify both the Transformer configuration in *transformer.conf* and the DataSource configuration.

Assuming the DataSource initiates the connection to the Transformer (this is usually the case):

DataSource configuration

For each connection to the Transformer specify an **add-peer** entry with a **local-id** option and an optional **local-name** option. The **local-id** setting must be different for each entry.

```
add-peer
      local-name        MyDataSource_connx_1
      local-id          1
      addr              <<Transformer addr>>
      port              <<Transformer port>>
      ...
end-peer

add-peer
      local-name        MyDataSource_connx_2
      local-id          2
      addr              <<Transformer addr>>
      port              <<Transformer port>>
      ...
end-peer
```

Note: *The **addr** and **port** options are the same in each add-peer entry, since they are the address and port on which the Transformer listens for connection requests.*

Transformer configuration

For each connection to the DataSource specify an **add-peer** entry with a **remote-id** option. The **remote-id** settings should correspond to those of the **local-id** options in the DataSource configuration.

```
add-peer
       remote-name       MyDataSource_connx_1
       remote-id         1
       ...
end-peer

add-peer
       remote-name       MyDataSource_connx_2
       remote-id         2
       ...
end-peer
```

As far as the Transformer is concerned this configuration is the same as that for accepting connections from two different DataSource peers. At run time the Transformer will accept connections from peers with ids 1 and 2, and will be unaware that it is the same DataSource at the other end of the two connections.

If the Transformer initiates the connection to the DataSource then specify the configuration the other way round; the **local-id** settings must be in the Transformer configuration and the **remote-id** settings must be in the DataSource configuration. In this case the values in **local-id** will override the Transformer's global id number defined in **datasrc-id**, and the **local-name** settings will override the Transformer name defined in **datasrc-name**.

Enabling failover

Transformer knows a peer is down when it loses its network connection to the peer or it fails to receive heartbeat signals from that peer.

The **add-peer** entries can be used to set up the Transformer to allow a data source failover and enable Transformer to connect to alternative data sources when required. A single **add-peer** section can configure a set of alternative peers to connect to using the addr and port options.

This can be configured by commenting out all the ***add-peer*** options and using the default settings, with the exception of the following options:

addr	must have at least one data source identified to failover to. If more are specified, then the Transformer will try the first source, and if that fails too, it will try the second and so on.
port	each data source identified in the addr option must have a port specified.

Transformer will connect to the first ***addr*** and ***port*** in the list and failover to the others in order if it cannot connect to the preceding peer in the list. Having established a connection with another source, it will continue to request data from it until that connection fails and it attempts to connect to the other sources in order again.

The following example allows failover to 4 data sources; the Transformer will try each identified source in turn.

```
add-peer
      addr  192.168.201.245 192.168.201.245 192.168.201.245 192.168.201.245
      port  25110           25111           25112           25113
end-peer
```

Using data services, multiple peers can be configured for failover without Transformer needing to swap connections. See “Data services” on page 28.

- Use the following parameters in the configuration file *transformer.conf* to determine whether Transformer ignores extra connection attempts by a user.

datasrc-reject-new-peers	If a DataSource peer tries to connect to the Transformer but there is already one connected with the same id (for example, if a peer's firewall has been down and the peer is registered as connected but in fact is not), the current peer will be disconnected and the new one is allowed to connect. ***datasrc-reject-new-peers*** turns off this default behaviour so the new DataSource peer is not allowed to connect. See page 52.

3.5 Reconnecting peers using the UDP interface

Transformer includes a UDP command interface that enables you to send a UDP message to reset peer connections after failover.

- Include the following options in the file *transformer.conf* in order to use the UDP interface.

udp-port	Port to listen on for UDP messages. If not specified then udp signals are disabled. See “Configuring the UDP interface” on page 76.
udp-interface	Network interface to listen on for UDP messages. See “Configuring the UDP interface” on page 76.

The following UDP command can be sent over a Transformer’s UDP interface.

peer-reconnect

An instruction to attempt to reconnect with the specified peers. If several DataSource peers have been configured to be used as alternative or failover sources, this enables your application to reconnect to previously failed peers if they are now online. By default , the first failover address is reconnected to, if no number is given:

Syntax: ***peer-reconnect peers addr-num***

Parameter:

Name	Type	Description
peer	int	Datasource peer index which should be reconnected to after failover. ***Note:*** *These are not DataSource IDs (specified by datasrc_id parameter in the configuration file), but correspond to the order of the peers' add-peer entries in the configuration file. The first add-peer is for peer 0, the next peer 1 and so on.*
addr-num	int	Which address in the failover list to reconnect to. Defaults to the first in the list.

- For how to issue the UDP command, see the section “UDP commands” on page 77.

3.6 Data services

Note: *Data services replace the old Source Mapping feature.*

You must use data services in order for Transformer to request a particular object from a particular DataSource or to define where broadcast data can come from. They allow you to define where data comes from, based on its subject name. They also allow the definition of groups of peers in a way that allows priority, failover and load balancing.

A data service defines the following:

- a name, which is the identifier for the service;
- a regular expression pattern match on the object name, or a number of patterns - this defines which objects will come from this service;
- a DataSource peer or set of peers that the request for the object will be forwarded to.

The DataSource peers defined for a service allow a number of different structures. Each service can have a number of ‘source groups’. Within a source group a number of priority groups can be defined, and within those priority groups, lists of peers can be defined.

When an object needs to be requested from a service and Transformer first looks at the service groups, it will make a request to a peer from each group at the same time. This may be useful if you do not know which peer has the data, or if a peer is serving a different set of fields and the data needs to be merged together.

Within a source group Transformer will look at the first priority group and request from a peer in that priority. If there are multiple peers in the priority group, Transformer will send the request to the peer with the smallest number of existing subscriptions – this achieves load balancing across peers. If no peer is connected in that priority, or if the peers in that priority did not have that object, the Transformer will try the next priority group - this achieves failover.

Active data services are identified within the data service section of the *transformer.conf* configuration file. How these are configured is detailed in “Configuring data services” on page 62.

Specifying the object or objects

Examples of different applications of mappings are given below.

For example:

```
include-pattern "^/NA/"
```

would request any object starting with the characters /NA/

```
include-pattern "^/[A-M]"
```

would request any object starting with the characters /A to /M

```
include-pattern "ABC"
```

would request any object containing “ABC” in any part of the name.

Note: *Remember that this is a regular expression and should start with a “^” if the pattern should only match from the beginning of the object name.*

Specifying a single DataSource peer

The DataSource peers to be mapped are specified by adding them as labels (see “Configuring data services” on page 62).

For example:

```
add-data-service
      service-name           MyService
      include-pattern        ^/NA/
      add-source-group
            required         true
            add-priority
                  label      sic2
            end-priority
      end-source-group
end-data-service
```

would request any object starting with the characters /NA/ from the DataSource peer with ID sic2.

Specifying alternative DataSource peers

By sending your requests to a sequence of DataSource peers, you can ensure that no individual peer is overloaded. This is particularly useful when a number of peers hold similar data.

Enter alternative DataSource peers within the same priority group (see “Configuring data services” on page 62).

For example:

```
add-data-service
      service-name            MyService
      include-pattern         ^/NA/
      add-source-group
            required          true
            add-priority
                  label       src1
                  label       src2
                  label       src3
            end-priority
      end-source-group
end-data-service
```

This means each request that matches "^/NA/" will go to one of the DataSource peers src1, src2, or src3. The request is directed to the peer with the smallest number of existing subscriptions, thus spreading the the load evenly across the peers.

Specifying multiple datasource peers

To send the same request to more than one DataSource peer, enter more than one source group (see “Configuring data services” on page 62).

For example:

```
add-data-service
      service-name            MyService
      include-pattern         ^/NA/
      add-source-group
            required          true
            add-priority
                  label       src1
            end-priority
      end-source-group
      add-source-group
            required          true
            add-priority
                  label       src2
            end-priority
      end-source-group
end-data-service
```

This will mean any request starting "/NA/" will be sent to DataSource peer src1 and peer src2 at the same time.

Note: *If both DataSource peers reply with data then the updates will be duplicated, so this configuration should not be used if both peers have the same data. This combination is more likely to be useful when you are not sure which peer has what data.*

Specifying priority or failover

You can configure a data service to send to an alternative DataSource peer if your first choice of peer is down, for example:

```
add-data-service
      service-name           MyService
      include-pattern        ^/NA/
      add-source-group
            required         true
            add-priority
                  label      src1
            end-priority
            add-priority
                  label      src2
            end-priority
      end-source-group
end-data-service
```

This will only request from peer src2 if peer src1 is down.

More complex mappings

More complex combinations of DataSource peers can be defined. For example:

```
add-data-service
      service-name           MyService
      include-pattern        ^/NA/
      add-source-group
            add-priority
                  label      src1
                  label      src2
            end-priority
      end-source-group
      add-source-group
            add-priority
                  label      src3
                  label      src4
            end-priority
      end-source-group
end-data-service
```

This results in the server sending requests to two DataSource peers simultaneously, one to whichever of src1 or src2 has the smallest number of existing subscriptions, and one to whichever of src3 or src4 has the smallest number of existing subscriptions.

Waiting for responses

Use the following parameters in the configuration file *transformer.conf* to set the timeout period to wait for responses from a peer following a request for data.

service-request-timeout	Time in seconds that the Transformer will wait for a Service to answer a request—after this time the Transformer will send a discard to all peers that have not responded.to request from another peer if the service defines a suitable alternative. A discard is sent to the DataSource peer to cancel the timed out request. This value can be overridden for an individual service by using the request-timeout option of the ***add-data-service*** entry (see page 64).
source-request-timeout	Time in seconds that the Transformer will wait for an individual DataSource to answer a request—after this time Transformer will attempt to request from another peer if the service defines a suitable alternative. A discard is sent to the datasource peer to cancel the timed out request. This value can be overridden for an individual source by using the ***request-timeout*** option of the ***add-peer*** entry (see page 54).

3.7 Making SSL connections with DataSources

SSL certificates can be configured at either or both client and server ends of the channel—Transformer is said to be operating in server mode when accepting connections from DataSources, and in client mode when connecting to DataSources.

There is no fallback to non-SSL operation should the SSL connection fail to be established.

- To configure SSL certificates specify the following configuration parameters in the file *transformer.conf*..

start-ssl	Configures the SSL connection when setting up Transformer to be both client and server ends of an SSL channel. This group is needed in the configuration file of both client and server applications. See page 57.
ssl-passwordfile	Identifies the file containing the SSL certificate passphrase. See page 61.

Server mode only configuration

- To configure Transformer for SSL when in server mode, use the ***datasrc-sslport*** option to select the network port to listen for SSL connections from DataSource peers (see page 53)***.***

- It is possible for Transformer to accept both SSL and non-SSL connections on different ports. Non-SSL connections should be configured using the ***datasrc-port*** option (see page 53).

Client mode only configuration

To configure Transformer for SSL when in client mode, use the ssl option in the add-peer entry for the DataSource peer that acts as server. For more information see ***add-peer*** on page 54.

3.8 Monitoring connections using heartbeats

Despite the many precautions taken in all Caplin products to ensure that data quality is maintained, it remains possible that a link in the chain may fail and that updates may be delayed, or may not arrive at all. In such circumstances, it is essential that the client is alerted instantly to the fact that the data may be stale.

The Health Check system uses heartbeat messages to guarantee this. There are two types of heartbeat:

- DataSource heartbeats are exchanged between a DataSource and Transformer to signal that the source of data being accessed (for example a trading system, exchange feed or spreadsheet) is still alive.

- Application level heartbeats allow client applications to check whether data may be stale.

DataSource heartbeats

Configure DataSource heartbeats for each DataSource connected to Transformer using the ***heartbeat-time*** and ***heartbeat-slack-time*** options within the ***add-peer*** configuration option. See page 56.

Application level heartbeats

You can define an application level heart beat for Transformer, so that client applications can check whether Transformer is running.

Application level heartbeats work by creating a special heartbeat object with a symbol name defined by the ***heartbeat-symbol*** configuration option. This object contains a timestamp field and the ***heartbeat-symbol-time*** configuration option defines how often Transformer updates this field. Every time Transformer updates the time stamp field the update is propagated (usually via a Liberator server) to applications that have subscribed to the heartbeat object. See “Application level heartbeat settings” on page 72.

3.9 Failover and recovery

Caplin Transformer uses its own internal local storage of data, for persistence during shutdowns. Two machines running in parallel are necessary to ensure failover in the event of software or hardware failure.

The failover mechanism has the following characteristics:

- in the event of a hardware failure, the incoming feed continues to be processed by the remaining machine;

- in the event of a software crash on one machine resulting in data not being processed correctly, the secondary machine takes over the task of supplying data to all applications;

- In all instances of a failure or recovery an event log is produced and e-mailed to support staff;

Caplin Transformer receives and sends data sent using Caplin's DataSource protocol. All DataSource-enabled applications can be configured to be used as alternative or failover sources, which means that when any connection in the chain is lost the side responsible for the connection can attempt to reconnect using a degrading retry algorithm (in other words, the side responsible for the connection will wait longer before retrying after each successive failure to reconnect).

3.10 Improving performance using DataSource threads

Transformer uses a thread for each connection it has to a DataSource. This helps to improve performance when the Transformer is connected to several DataSources, as there will be a separate thread for each DataSource.

If the Transformer has just one incoming DataSource feeding it updates at a high rate, you may be able to improve performance by configuring more than one connection to the DataSource. Each connection will use a separate thread, so the updates will be spread across multiple threads. For how to configure multiple connections to a single DataSource see “Multiple connections to a DataSource” on page 24.

4 Configuring Transformer

Caplin Transformer is configured by editing configuration files.

- Configure the core processing by editing the file *transformer.conf*, which can be found in the *etc/* directory. All configuration options in this chapter can be found in this file.
- Configure individual modules by editing the file *[module name].conf*, which can also be found in the *etc/* directory.

4.1 Configuring fields

The following options in *transformer.conf* configure the fields contained in objects.

The **add-field** entries in the *transformer.conf* configuration file define what fields are used within Caplin Transformer. They configure the field name and field number, as well as setting various flags which can customise the characteristics of the field. You can configure multiple field numbers to be translated to the same field name if necessary, but not vice versa.

add-field

Adds a field. You can have any number of entries.

syntax: ***add-field FieldName FieldNumber***

The options in this entry are:

Name	Type	Default	Description
FieldName	string	[no default]	The name of the field.
FieldNumber	integer	[no default]	The number of the field. Must be less than 64,000.
FieldFlags	integer	0	The flags passed by the field.
FieldFlagsData	-	-	Not used by Caplin Transformer.
FieldFormat	-	-	Not used by Caplin Transformer.

Note: *There is no need to add fields unless the application is using text-based field names.*

Figure 4-1 below shows the acceptable values of *FieldFlags*. *FieldFlags* can be represented by either a text string or an integer.

FieldFlags(text)	FieldFlags (integer)	Description	FieldFlagsData	FieldFormat
type2_index	1	Identifies field as Type 2 index	Not used	Not used
type2	2	Identifies field as Type 2 field	Not used	Not used
type3	4	Identifies field as Type 3 field	Not used	Not used

Table 4-1: Acceptable values of FieldFlags option

fields-file

Identifies a separate file containing all the required **add-field** configurations.

Type:string

Default value: *fields.conf*

4.2 Running Caplin Transformer in Daemon mode

The following option in *transformer.conf* determines whether Caplin Transformer runs in Daemon mode (i.e. as a background process).

daemon-enable

Allows Caplin Transformer to run in daemon mode.

Type:	boolean
Default value:	FALSE

4.3 Configuring log files

The following options in *transformer.conf* configure Caplin Transformer's log files. The log files keep a record of all data and to what destinations the data was sent.

log-dir

Default directory in which to store log files.

Type:	string
Default value:	%r/var

log-file

Name of log file

Type:	string
Default value:	transformer.log

log-cycle-period

Interval between cycling logs, in minutes.

Type:	integer
Default value:	1440 (i.e. daily)

log-cycle-offset

Specifies how many minutes to take off the current time when creating the suffix.

Type:	string

Default value:The same as **log-cycle-period**. For example, if cycling at 0400 hours, the time passed into **strftime** to create the suffix will be 0400 hours the previous day.

log-cycle-suffix

Suffix for cycled logs. This is passed through **strftime** (refer to your UNIX manual for further information on **strftime**). The default of %u results in a file being created for each day of the week.

Type:	string
Default value:	%u

add-log

As well as the global configure options for log file cycling, individual log files can be cycled. This option overrides the global defaults for a particular log file.

syntax:

```
add-log
        name [value]
        maxsize[value]
        time   [value]
        period[value]
        suffix [value]
        offset [value]
        level  [value]
        monitor-level[value]
end-log
```

The options in this entry are:

Name	Type	Default	Description
name	string		The name of the log to cycle. If no value is entered the global settings are used. Acceptable values are: event_log the event log file (see page 40) packet_log the packet log file (see page 52)
maxsize	integer	0	Maximum log file size in bytes. The log files will be cycled if they exceed the size specified here, therefore a value of 0 means log files will cycle every time they are checked.
time	integer	240 (i.e. 0400 hours)	Time at which logs will cycle, in minutes from midnight.
period	integer	1440 (i.e. daily)	Interval between cycling logs, in minutes.

Name	Type	Default	Description
suffix	string	%u	Suffix for cycled logs. This is passed through **strftime** (refer to your UNIX manual for further information on **strftime**). The default value of %u results in a file being created for each day of the week.
offset	integer	**log-cycle-period**	Specifies how many minutes to take off the current time when creating the suffix. For example, if cycling at 0400 hours, the time passed into **strftime** to create the suffix will be 0400 hours the previous day.]
level	string	INFO (this defaults to the global option ***log-level***).	Debug level for the log. ***Note:*** *This is only valid for the event log.*
monitor-level	string	NOTIFY (this defaults to the global option ***monitor-level***).	Debug level to send messages to monitoring. ***Note:*** *This is only valid for the event log.*

4.4 Configuring the core memory

The following options in *transformer.conf* determine how the Caplin Transformer's core memory and caching functions are configured.

memory-hash-size

Size of hash table.

Type:	integer
Default value:	16384

memory-file

Location of file where the cache is saved on shutdown.

Type:	string
Default value:	%r/var/memory

constant-field

Fieldname of any "constant" fields, which are kept safe when the memory is reset.

Type:	string
Default value:	[no default]

add-purge

Sets purging settings for a particular object. Purging involves deleting the object from Transformer's cache—this enables non-active objects to be refreshed regularly.

syntax ***add-purge [symbol pattern] [purge time] [purge days]***

The options in this entry are:

Name	Type	Default	Description
symbol	string	[no default]	Object name.
purge time	integer	-1 [no purge]	Number of minutes from midnight to start purging.
purge days	integer	[no default]	Number of days to keep this object in Transformer's cache when purging.

type2-hash-size	Size of the Type 2 data hashtable.
	Type: integer
	Default value: 65536

4.5 Configuring email

Certain events can trigger Caplin Transformer to send emails, such as when a source has been started or when a source is down. These emails are sent in batches

The following options in *transformer.conf* configure the emails Caplin Transformer sends.

mail-rcpt

Comma separated list of recipient email addresses.

Type:	string
Default value:	[no default]

mail-from

Sender of email messages.

Type:	string
Default value:	[no default]

mail-subject

Subject of the email.

Type:	string
Default value:	[no default]

mail-time

Frequency (in seconds) with which emails should be sent.

Type:	integer
Default value:	600

mail-program

Location of script or program that sends email.

Type:	string
Default value:	[no default]

4.6 Identifying the UDP port

This section of *transformer.conf* configures how Caplin Transformer listens for UDP messages. For information on how to send UDP messages from Transformer, see **Using UDP Commands** on page 76.

udp-port

Port to listen to for UDP messages.

Type:	integer
Default value:	10001

4.7 Identifying which modules are used

This section of *transformer.conf* identifies the business processing modules that Caplin Transformer uses. Each module consists of a library file (*[module name].so*) and a configuration file (*[module name].conf*.)

modules-dir

Directory where the module library files (*[module name].so*) files are kept.

Type:	string
Default value:	/%r/lib

modules-conf-dir

Directory where the module configuration files (*[module name].conf*) files are kept.

Type:	string
Default value:	/%r/etc

add-module

Loads a module into the system.

syntax:***add-module [module name] [extend api]***

The options in this entry are:

Name	Type	Default	Description
module name	string	[no default]	The name of the module to load. This can be with or without the filename extension. On Linux or Sun Solaris the file name extension is *.so* On Windows the file name extension is *.dll*
extend api	integer	0	(Optional) Determines whether this module extends the Transformer API. Acceptable values are: *0* does not extend API *1* extends API

In order to load a Java module, you must also include an **add-javaclass** entry after **add-module** (see page 49, and the example on page 51).

For example:

```
add-module          jtm
add-javaclass
   class-name       examples.rules.RulesTransformerModule
   class-id         jtm
   classpath        %r/lib/java/examples.jar
end-javaclass
```

4.8 Configuring the Java Virtual Machine

The Java Virtual Machine (JVM) is the software required to execute the Java modules created using Transformer SDK for Java. An example JVM configuration is included on page 51.

Note: *When using Linux, LD_LIBRARY_PATH must be set to /usr/java/jre/lib/i386:/usr/java/jre/lib/i386:/server where the java runtime environment is installed in /usr/java/jre.*

jvm-location

Location of the Java Virtual Machine file *libjvm.so*. Should contain the complete path and include the *.so* suffix.

Type:	string
Default value:	[no default]

jvm-global-classpath

Location of the global classpath. There must be a separate **jvm-global-classpath** entry for each classpath—see the example on page 51. You can also specify classpaths using the classpath parameter of as **add-javaclass** entry see page 49.

Type:	string
Default value:	%r/lib/java

add-javaclass

Identifies the Transformer Java module to be loaded. Must be used with an associated **add-module** entry (see page 47).

To load a second Java module, make a copy of the Transformer Java implementation file *jtm.so* (found in the *lib* folder), and add a second **add-javaclass** entry. The *class-id* parameter must be set to the name of the copied file: see the example on page 51.

syntax:

```
add-javaclass
    class-name
    class-id
    classpath
end-javaclass
```

The options in this entry are:

Name	Type	Default	Description
class-name	string	[no default]	The fully-qualified class name of the Java module to load.
class-id	string	0	Short identifier of the Java class.
classpath	string array		Adds a Java classpath, in addition to those specified using **jvm-global-classpath** entries (see page 49).

jvm-options

Adds a standard startup option for the JVM. Multiple configuration lines may be specified.

Type:	string
Default value:	no default

For example, to enable socket debugging on port 9955 the following configuration options could be added:

```
jvm-options    -Xdebug
jvm-options    -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=9955
```

Example Java module configuration

The example configuration below instantiates **RulesTransformerModule** and **SecondTransformerModule** modules. The code for this example is included in *examples.jar*.

```
add-module              jtm.so

# Configuration of Java class to be loaded
add-javaclass
    class-name          examples.rules.RulesTransformerModule
    class-id            jtm
    classpath           %r/lib/java/examples.jar
end-javaclass


# Configuration of the JVM
jvm-location            /usr/local/jdk/jre/lib/sparc/libjava.so
jvm-global-classpath    %r/lib/java/transformermodule.jar


#Load a second Transformer Java Module jtm2

add-module              jtm2.so

add-javaclass
    class-name          examples.ssecond.SecondTransformerModule
    class-id            jtm2
    classpath           %r/lib/java/examples2.jar
end-javaclass
```

4.9 Configuring DataSource peers

This section of *transformer.conf* configures how the Caplin Transformer connects to other sources of data, called DataSource peers. A DataSource peer is a remote application which uses DataSource messaging to exchange real time data with the Transformer.

datasrc-name

The name of this Caplin Transformer, and how DataSource peers will identify it.

This name can be overridden by putting a value in the *local-name* option of the ***add-peer*** entry (see ***add-peer*** on page 54).

Type:	string
Default value:	[name and host of this application, set automatically by the application]

datasrc-id

ID number of this Caplin Transformer.

This ID can be overridden by putting a value in the *local-id* option of the ***add-peer*** entry (see page 54), in which case it must match the *remote-id* given in the **add-peer** entry in the remote DataSource peer's configuration.

Type:	integer
Default value:	0

datasrc-reject-new-peers

Determines whether a DataSource peer trying to connect to Transformer when there is already one connected with the same ID, is forbidden to connect.

Type:	boolean
Default value:	FALSE

datasrc-pkt-log

Name of Caplin Transformer's packet log file. The location of the file must either be relative to ***log-dir*** (see page 40) or absolute.

Type:	string
Default value:	packet-%a.log

datasrc-interface	Network interface to listen for connections from DataSource peers. Type: integer Default value: all available interfaces
datasrc-port	Network port to listen for connections from DataSource peers. The default of 0 means that no connections can be made to Caplin Transformer. Type: integer Default value: 0
datasrc-sslport	Network port to listen for SSL connections from DataSource peers. The default of 0 means that no SSL connections can be made to Caplin Transformer. Type: integer Default value: 0
datasrc-default-obj-hash-size	Default number of entries in the active object hashtable. This size can be overridden by putting a value in the ***obj-hash-size*** option of the ***add-peer*** entry. Type: integer Default value: 16384
datasrc-rerequest-timeout	The time in seconds that the Transformer waits for a DataSource to respond when rerequesting an object that the DataSource was previously sending to Transformer. A rerequest happens when a DataSource goes down and comes back up. Default value: 30

add-peer

Adds a DataSource peer. You can have a maximum of 63 add-peer entries in your configuration file.

syntax

```
add-peer
        <option>            [value]
        ...
add-peer
```

The options in this entry are:

Name	Type	Default	Description
remote-id	integer	1	ID number of DataSource peer.
remote-name	string	scr-0	Name of DataSource peer. Gets overridden by the startup packet when the peer connection is made.
remote-flags	integer	0	DataSource peer flags. Gets overridden by the startup packet when the peer connection is made.
remote-type	integer	0	DataSource peer type. Gets overridden by the startup packet when the peer connection is made. Possible values are: "none" (0) "active" (1) "contrib" (2).
local-id	integer	***datasrc-id***	ID number of Caplin Transformer. Sent to the DataSource peer.
local-name	string	***datasrc-name***	Name of Caplin Transformer. Sent to the DataSource peer.

Name	Type	Default	Description
local-flags	integer	0	Flags determining restart and reconnection behaviour. The flags can be ORed together (for example "`sendfromseq|recvautoreplay`"). Possible values: "`none`" or 0 No special restart/reconnection behaviour; "`sendfromseq`" or 1 When reconnecting, missed packets should be requested based on sequence number; "`recvautoreplay`" or 4 When restarting, this peer should accept replay updates.
local-type	integer	0	Data source type sent to the connecting peer. Possible values are: "none" (0), "active" (1) contrib" (2).
addr	array of strings	localhost	Space-separated list of addresses to connect to if making the connection and not listening/accepting the connection. See Note below.
port	array of integers	[no default]	Space-separated list of ports to connect to if making the connection and not listening/ accepting the connection. See Note below.
queue-size	integer	50	Message queue size.
queue-delay	float	0.1	Message queue delay in seconds.

Name	Type	Default	Description
obj-hash-size	integer	***datasrc-default-obj-hash-size***	Number of entries in active object hashtable.
ssl	boolean	False	Determines whether this connection should be made using SSL. For more information on SSL connections, see “Making SSL connections with DataSources” on page 34.
request-timeout	float	-1 (no timeout)	Time in seconds that the Transformer will wait for this DataSource peer to answer a request. When set to a positive value it overrides the global request timeout option ***source-request-timeout***.
heartbeat-time	integer	[disabled]	Time in seconds between DataSource heartbeats. The two peers involved in a DataSource connection compare their ***heartbeat-time*** values and use the lowest.
heartbeat-slack-time	integer	2	When the Transformer does not receive an expected DataSource heartbeat it waits ***heartbeat-slack-time*** seconds before disconnecting from the peer and trying to reconnect to it. (This value is not compared by peers.)
label	string	peer[int]	There must be a label set for each ***label*** used in the ***add-priority*** section of the ***add-data-service*** option (see page 67).

Note: ***addr*** *and* ***port*** *should only be included if the connection is to be made to the peer as opposed to listening for a connection. If additional addr and port combinations are given they will be used as failover addresses if the first fails to connect (the peer must be configured to accept connections—this is done through the* ***datasrc-port*** *entry in the peer's configuration file).*

peerdown-time

This option controls Transformer's cache flushing behaviour when it is connected to a Liberator server.

If Transformer detects that its connection to the Liberator has gone down it waits ***peerdown-time*** seconds. After this time it discards any objects in its cache that have not been re-requested by the Liberator since the connection was lost. This option should be set to a value that allows a failed Liberator to restart and re-request the majority of the objects it requires before Transformer deletes them, so that Transformer does not have to fetch them again from its other DataSources. The default value is intended to meet this requirement.

Type:	integer
Default value:	60

start-ssl

Configures the SSL connection when setting up the Transformer to be both client and server ends of a Secure Sockets Layer channel.

Note: *Not all options listed in this group are appropriate for both server and client modes.*

syntax

```
start-ssl
    enable-server
    enable-client
    server-authmode [value]
    client-authmode [value]
    server-cert     [value]
    client-cert     [value]
    server-key      [value]
    client-key      [value]
    cipher          [value]
    ssl2
    ssl3
    CApath          [value]
    CAfile          [value]
    ssl-info
end-ssl
```

The options in this entry are:

Name	Type	Default	Description
enable-server	boolean	FALSE	Enables server-side SSL.
enable-client	boolean	FALSE	Enables client-side SSL.
server-authmode	integer	0	A logical OR of the flags described in Table 4-2 on page 61. Exactly one of the mode flags SSL_VERIFY_NONE and SSL_VERIFY_PEER must be set at any time.
client-authmode	integer	0	A logical OR of the flags described in Table 4-2 on page 61. Exactly one of the mode flags SSL_VERIFY_NONE and SSL_VERIFY_PEER must be set at any time.
server-cert	string	server.pem	Filename of the location of the server-side certificate.
server-key	string	server-cert	Filename of the location of the server-side private key.
client-cert	string	NULL	Filename of the location of the client-side certificate.
client-key	string	client-cert	Filename of the location of the client-side private key.

Name	Type	Default	Description
cipher	string	[strongest common cipher]	Sets the cipher to be used for the connection (usually] defined on the SSL client side). For more information on cipher types please refer to the OpenSSL documentation at http://www.openssl.org
ssl2	boolean	FALSE	Sets the SSL protocol level to Level 2.
ssl3	boolean	FALSE	Sets the SSL protocol level to Level 3.
CApath	string	System CApath	Sets the directory name of the directory where the trusted certificates are held.
CAfile	string	NULL	Sets the filename of the file where all trusted certificates are held.
ssl-info	boolean	FALSE	Enables SSL connection negotiation debugging.

client-authmode and server-authmode flags

Table 4-2 below describes the flags to be used for the *client-authmode* and *server-authmode* options within the ***start-ssl*** group.

Name	Value	server-authmode description	client-authmode description
SSL_VERIFY_NONE	0	Transformer will not send a client certificate request to the client, so the client will not send a certificate.	If not using an anonymous cipher (disabled by default), the Transformer will send a certificate which will be checked. The handshake will be continued regardless of the verification result.
SSL_VERIFY_PEER	1	Transformer sends a client certificate request to the client. The certificate returned (if any) is checked. If the verification process fails, the TLS/SSL handshake is immediately terminated, with an alert message containing the reason for the verification failure. The behaviour can be controlled by the additional SSL_VERIFY_FAIL and SSL_VERIFY_CLIENT_ONCE flags.	The server certificate is verified. If the verification process fails, the TLS/SSL handshake is immediately terminated with an alert message containing the reason for the verification failure. If no server certificate is sent, because an anonymous cipher is used, SSL_VERIFY_PEER is ignored.

Name	Value	server-authmode description	client-authmode description
SSL_VERIFY_FAIL	2	If the client did not return a certificate, the TLS/SSL handshake is immediately terminated with a "handshake failure" alert. This flag must be used together with SSL_VERIFY_PEER.	Ignored
SSL_VERIFY_ CLIENT_ONCE	4	Only request a client certificate on the initial TLS/SSL handshake. Do not ask for a client certificate again in case of a renegotiation. This flag must be used together with SSL_VERIFY_PEER.	Ignored

Table 4-2: client-authmode and server-authmode flags

ssl-passwordfile Identifies the file containing the SSL certificate passphrase.

Type:	string
Default value:	.transformer.ssl.pass

4.10 Configuring data services

This section of *transformer.conf* configures Transformer’s data services.

What is a data service?

See “Data services” on page 28 for an overview of what data services are and how they are used.

The service name

This is an identifier which can be used in status messages.

Note: *When picking a service for an object, the first defined match takes priority. As such you should ensure that each object is associated with one and only one service.*

The subject patterns

These are regular expression strings to accept or deny for this service. A service will allow multiple patterns including patterns to deny. Exclude patterns can help to define the namespace used.

Note: *When checking pattern matches within a service definition, the first match takes priority whether it is an include or an exclude.*

The source groups

The main part of the service definition is the source groups. This is one or more sets of sources, plus certain attributes which define the behaviour of the group. In most cases only a single group is defined. When multiple groups are defined for a service it means that a request will attempt to get the object from a source from each group. Multiple groups allow an object to have different sets of fields coming from different sources, for example.

Priorities

Priorities are defined within each source group and are taken in the order in which they are defined. Multiple labels can be defined within each priority. Within a priority, the peer selected is the one with the smallest number of existing subscriptions.

Timeouts

There are several timeouts associated with data services:

- ***service-request-timeout*** (see page 63)
- ***request-timeout*** option of ***add-data-service*** (see page 64)
- ***retry-time*** option of ***add-source-group*** in ***add-data-service*** (see page 66)
- ***source-request-timeout*** (see page 63)
- ***request-timeout*** option of ***add-peer*** (see page 56)
- ***cleanup-stale-timeout*** (see page 63)

Use the following options to configure data services.

service-request-timeout

Global request timeout in seconds for all data services.

Type:	float
Default value:	10

The ***service-request-timeout*** applies to a request for an object via a service. Should no response be received within this time from the peers providing a service, the object will be assumed to be not available.

service-request-timeout is a global setting applied to all data services. You can override this timeout for individual DataServices by setting the ***request-timeout*** option of ***add-data-service*** – see page 64.

source-request-timeout

Global request timeout in seconds for all DataSources.

Type:	float
Default value:	-1 (no timeout set)

The ***source-request-timeout*** applies to individual DataSources within data services. It is the time that the Transformer will wait for a DataSource peer to answer a request.

source-request-timeout is a global timeout that applies to all DataSource peers. You can override this timeout for individual peers by setting the ***request-timeout*** option of ***add-peer*** – see page 56.

cleanup-stale-timeout

When all the DataSource peers in a service have been down for ***cleanup-stale-timeout*** seconds, stale objects will be deleted from the Transformer cache .

Type:	float
Default value:	-1 (no timeout set)

add-data-service Starts the definition of a data service.

```
add-data-service
    service-name            [value]
    request-timeout         [value]
    exclude-pattern         [value]
    include-pattern         [value]
    add-source-group
        required            [boolean]
        retry-time          [value]
        add-priority
            label           [value]
            ...
        end-priority
        ...
    end-source-group
    ...
end-data-service
```

service-name Name of the service group.

Type:	string
Default value:	none

request-timeout This option defines the timeout in seconds for all requests for this service. Should no response be received within this time from the peers providing the service, the object is assumed to be unavailable. The ***request-timeout*** is the master timeout, it overrides the ***retry-time*** option and the timeouts on individual peer requests (see ***retry-time*** on page 66).

Type:	float
Default value:	-1

The default value of -1 means that requests will never time out.

exclude-pattern	Patterns to exclude, defined as regular expressions. For examples of the specification format see “include-pattern” on page 65 .

Type:	string array
Default value:	none

include-pattern	Patterns to include, defined as regular expressions.

Type:	string array
Default value:	none

Example 1:

```
include-pattern    ^/A/ ^/B/ ^/C/
```

This example specifies three patterns to include: ‘^/A’, ‘^/B’ and ‘^/C’.

Example 2:

```
include-pattern    ^/A/
include-pattern    ^/B/
include-pattern    ^/C/
```

This example specifies the same three patterns as in example 1, but as separate ***include-pattern*** statements.

add-source-group

Add a source group.

syntax:

```
add-data-service
...
    add-source-group
        required          [boolean]
        retry-time        [value]
        add-priority
            label         [value]
            label         [value]
            ...
        end-priority
        ...
    end-source-group
...
end-data-service
```

See also the examples in “Data services” on page 28.

required

When set to true, a status stale message or status information message is generated if a DataSource peer within the source group goes down.

Type:	boolean
Default value:	false

retry-time

After finding that none of the peers (defined by ***label*** options) in the source group have responded to a request, the Transformer waits ***retry-time*** seconds before reissuing the request to the group.

Type:	float
Default value:	30

The Transformer will issue the request to each of the peers in the source group in turn. Each request is timed out according to the setting of ***request-timeout*** in the ***add-peer*** option (see page 64). If none of the peers in the group replies, the Transformer waits ***retry-time*** and then again tries each connection in turn. It will repeat this sequence until the master timeout for the service, defined by the ***request-timeout option*** of ***add-data-service***, expires (see page 64). If the ***add-data-service request-timeout*** out is set to -1, or is not defined, the Transformer will continue to reissue the request indefinitely.

add-priority	Start a priority group. Specifies one or more DataSource peers to be assigned to the data service, in a group of the same priority. For examples of how to use this configuration option see "Data services" on page 28.

Syntax:

```
syntax:     add-data-service
            ...
                add-source-group
                    ...
                    add-priority
                        label      [value]
                        label      [value]
                        ...
                    end-priority
                ...
                end-source-group
            ...
            end-data-service
```

label	Peer label identifying a peer that provides the data service. The label is inserted in an ***add-priority*** option.

Type:	string array
Default value:	none

The label is defined using a ***label*** option within the ***add-peer*** configuration option – see "add-peer" on page 54.

Example 1:

```
            add-priority
                 label       src1 src2
            end-priority
```

This example specifies two peer labels, `src1` and `src2`.

Example 2:

```
            add-priority
                  label       src1
                  label       src2
            end-priority
```

This example specifies the same two peer labels as in example 1, but as separate ***label*** statements.

Below is an example section of *transformer.conf* illustrating how data services are configured. See also the examples in “Data services” on page 28.

```
add-data-service

      service-name            FX

      exclude-pattern         ^/I/X/
      include-pattern         ^/I/
      include-pattern         ^/B/

      add-source-group
            required          true
            retry-time        45
            add-priority
                  label       sourceA
            end-priority
            add-priority
                  label       sourceB
                  label       sourceC
            end-priority
      end-source-group

      add-source-group
            required          false
            add-priority
                  label       source1
                  label       source2
            end-priority
      end-source-group

end-data-service
```

Default behaviour

If no data service is defined in *transformer.conf* then the application will act as if the following data service configuration was defined:

```
add-data-service
      service-name default
      include-pattern        ^/
      add-source-group
            required         false
            add-priority
                  label      source1
            end-priority
      end-source-group
      add-source-group
            required         false
            add-priority
                  label      source2
            end-priority
      end-source-group
      .
      .
      .
      add-source-group
            required         false
            add-priority
                  label      sourceN
            end-priority
      end-source-group
end-data-service
```

This means that a request will be sent to all active DataSources at once.

Conversion of pre-version 4.0 source mapping

Pre-version 4.0 source mapping should be converted to version 4.0 data services, as shown in the following examples.

Note that all peers should have a label defined in the ***add-peer*** configuration section. In the examples the label is 'src' appended with the the remote-id.

add-source-mapping /A/* 1 should be converted to:

```
include-pattern        ^/A/
add-source-group
      required         true
      add-priority
            label      src1
      end-priority
end-source-group
```

add-source-mapping /A/* 1,2 should be converted to:

```
include-pattern        ^/A/
add-source-group
      required         true
      add-priority
            label      src1
            label      src2
      end-priority
end-source-group
```

add-source-mapping /A/* 1 2 should be converted to:

```
include-pattern        ^/A/
add-source-group
      required         true
      add-priority
            label      src1
      end-priority
end-source-group
add-source-group
      required         true
      add-priority
            label      src2
      end-priority
end-source-group
```

add-source-mapping /A/* 1,2 3,4 should be converted to:

```
include-pattern         ^/A/
add-source-group
      required          true
      add-priority
            label       src1
            label       src2
      end-priority
end-source-group
add-source-group
      required          true
      add-priority
            label       src3
            label       src4
      end-priority
end-source-group
```

4.11 Application level heartbeat settings

The following options in *transformer.conf* configure Caplin Transformer's application level heartbeat signals. See also “Monitoring connections using heartbeats” on page 34.

heartbeat-symbol

The symbol name of the application level heartbeat object. The name can include the placeholders `%h` (host name) and `%n` (application name) , which are filled in at run time..

Type:	string
Default value:	NULL (disabled)

heartbeat-symbol-time

The time in seconds between updates of the time stamp in the application level heartbeat object.

Type:	float
Default value:	30

4.12 Configuring news

news-max-headlines	Maximum number of headlines to be cached for a particular news stream. Type: integer Default value: [unlimited]
news-purge-time	This represents the number of minutes from midnight that the purge of news headlines (i.e. deletion from Transformer's cache) should take place. Type: integer Default value: -1 (no purge, in which case **news-max-headlines** will limit the number of headlines stored.)
news-purge-days	Number of days-worth of headlines to keep when purging. This only takes effect if **news-purge-time** is greater than or equal to 0. Type: integer Default value: 0
record-max-cache	The maximum number of update history levels to keep within each type 3 record. (Updates to Type 3 records are retained as separate entries in the record, rather than overwriting the existing data.) Type: integer Default value: 10

add-newsconfig

Sets purging settings for a particular news stream. Purging involves deleting the news stream object from Transformer's cache.

syntax:

```
add-newsconfig
        symbol          [value]
        purge-days      [value]
        max-headlines   [value]
end-newsconfig
```

The options in this entry are:

Name	Type	Default	Description
symbol	string	[no default]	News stream symbol
purge-days	double	[no default]	Number of days of headlines to keep in Transformer's cache when purging.
max-headlines	integer	[no default]	Maximum number of headlines to be cached for this news stream.

4.13 Logging

log-level

Determines the level of errors and events that are reported to the *transformer.log* log file when Caplin Transformer is operating. Acceptable values are shown in Table 4-3 below.

If the UDP message interface is enabled then the logging level can be changed whilst Transformer is running. For details on how to achieve this using UDP, see **debug** on page 77.

Note: *A list of all error and event messages ,and their associated logging level, can be found in **Appendix A** on page 79.*

Type:	string
Default value:	INFO

Value	Description
DEBUG	Reports all errors and events.
INFO	Reports events and information regarding normal operation and all errors included in the WARN, NOTIFY, ERROR and CRIT debug levels.
WARN	Reports minor errors and all errors included in the NOTIFY, ERROR and CRIT debug levels.
NOTIFY	Report errors regarding data corruptions and all errors included in the ERROR and CRIT debug levels.
ERROR	Reports serious errors regarding network connections and all errors included in the CRIT debug level.
CRIT	Reports critical errors that prevent Transformer running.

Table 4-3: Error and event log levels

5 Using UDP commands

Caplin Transformer includes a UDP command interface that enables you to send UDP messages regarding debugging and the writing of lists of watched objects to files.

5.1 Configuring the UDP interface

udp-port	Port to listen on for UDP messages. If not specified then UDP signals are disabled. Default value: [no default]
udp-interface	Network interface to listen on for UDP messages. If not specified then Caplin Transformer listens for UDP signals on all interfaces Default value: [no default]

5.2 UDP commands

The following UDP commands can be sent over Caplin Transformer's UDP interface. For a list of available commands see Table 5-1 on page 76.

udpsend

A utility program that enables Transformer to send a UDP message.

Syntax:

```
udpsend
    -s [host]
    -p [port]
message
```

Parameters:

Name	Type	Default	Description
-s	string	localhost	Host name or IP address of machine to send message to.
-p	integer	10001	Port that host machine listens on for UDP messages.
message	string	[no default]	Message to send. The messages that can be sent are listed in Table 5-1 below.

Command	Parameter	Description
debug	Level of error and event reporting messages. See Table 4-3 on page 75	Change the level of error and event reporting dynamically. Overrides the level set using the configuration option **log-level** (page 70).
memory_write	[none]	Dump the core memory into the memory file.
peer-reconnect	peers addr-num	Reconnect with specified peers. (See “Reconnecting peers using the UDP interface” on page 27)
symbol_delete	Symbol	Delete all stored updates for the symbol.
symbol_publish	Symbol	Publish last update received for the symbol.

Table 5-1: Transformer Core UDP messages

Example:

```
udpsend -s 127.0.0.1 -p 10001 symbol_publish /LO/VOD
```

This command publishes all cached data from the last update for Vodafone on the London Stock Exchange to a transformer running on the same machine, listening on port 10001.

6 Appendix A: Log levels and messages

6.1 CRITICAL messages

```
CRITICAL: Couldn't set up UDP port (%d) to listen on
CRITICAL: No memory for job allocation
CRITICAL: calloc returned NULL for new mail message
```

Table 6-1: CRITICAL debug level messages

6.2 ERROR messages

```
ERROR: Error finding mod_init symbol in API module - this may be
OK...
ERROR: Module load error: %s
ERROR: Module exit error: %s
ERROR: No `From:' address specified
ERROR: No `To:' address specified
ERROR: Could not open tmp file %s for writing
ERROR: Fork to spawn mail sender failed..
ERROR: exec failure %d %s
ERROR: Unknown constant fieldname <%s>
ERROR: Received an L2 packet with date older than last - discarding
%s (%s)
```

Table 6-2: ERROR debug level messages

6.3 NOTIFY messages

```
NOTIFY: Internal request issue
NOTIFY: No time delta symbol, will use system time instead
```

Table 6-3: NOTIFY debug level messages

6.4 INFO messages

```
INFO: Log Level is %s Active is %d
INFO: Registering interest in type %d
INFO: Registering wildcard for %s
INFO: Adding listener for %s
INFO: We conf'd %d modules confdir=<%s>
INFO: Doing exit function for %s
INFO: Command unload called with %s
INFO: Loading module <%s>
INFO: Unloading module %s
INFO: Module %s not loaded
INFO %d %d %d %d
INFO: Reading from file <%s>
INFO: Peer %d requesting %s %d
INFO: Broadcast source can't discard %s
INFO: Sending status %d for %s <%s>
INFO: Sending flags %d for %s to peer %d
INFO: Cleaning up for disconnect from peer %d
INFO: Registering module provider for symbols <%s>
INFO: Deregistering module provider for symbols <%s>
INFO: peer %d discarding %s
INFO: Listening on UDP port %d for messages
INFO: Added job on cmd <%s> @%p for "%s"
INFO: Successfully removed UDP job <%s>
INFO: UDPjob (%d)/cmd combo not found so couldn't delete
INFO: UDP Command is %s time is %lu
INFO: Sending <%s> %d bytes to 127.0.0.1:%d (UDP)
INFO: New DELTA time-delta is %ld
INFO: Opened tmp file %s (Subject: <%s> To: <%s>
INFO: Deleted mail file <%s>
INFO: About to exec(): %s
INFO: Accepted shell connection
INFO: Closed shell connection
INFO: Received flags %d from peer %d for %s
INFO: We got an update for %s (%d fields, type %d) feed %d (%d)
INFO: Request flag set so sending out %d fields for <%s>
INFO: Received status flags %d message <%s> for %s from peer %d
```

```
INFO: Trying queued request %s ret = %d
INFO: Successful for %s adding listener if required
INFO: Symbol %s already exists, setting flags
INFO: No functions registered for userdata type <%x>
INFO: Removing symbol <%s> type %d
INFO: Publishing packet for %s (%d fields)
INFO: Overwriting datatype functions for type %d
INFO: Adding handler functions for type %d
INFO: Deregistering type functions for type %d
```

Table 6-4: INFO debug level messages

6.5 DEBUG messages

```
DEBUG: Entering main loop
DEBUG: Module registering interest in symbol <%s>
DEBUG: Sending packet for %s to peer %d
DEBUG: Matching <%s> to provider <%s>
DEBUG: Read %d bytes on UDP port
DEBUG: Already received read message
DEBUG: Calling functions for command
DEBUG: Sending heartbeat message: %s
DEBUG: parent success
DEBUG: Updating packet for %s field %d with %s
DEBUG: Field %d not present in packet for %s, adding
DEBUG: Registering userdata functions for ID 0x%x (%d)
DEBUG: Finding userdata functions for ID 0x%x
DEBUG: Matched <%s> to wildcard <%s>
```

Table 6-5: DEBUG debug level messages

Single-dealer platforms for the capital markets

CAPLIN

Contact Us

Caplin Systems Ltd
Triton Court
14 Finsbury Square
London EC2A 1BR
UK
Telephone: +44 20 7826 9600
Fax: +44 20 7826 9610

www.caplin.com

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

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

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

This publication could include technical inaccuracies or typographical errors and is subject to change without notice. Changes are periodically added to the information herein; these changes will be incorporated in new editions of this publication. Caplin Systems Limited may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

Copyright © 1998-2010 Caplin Systems Ltd.
All rights reserved.