CAPLIN

Caplin Platform 6.2

How To Manage And Interpret Log Files

October 2014

Contents

1 Preface 1

1.1 What this document contains 1
About Caplin document formats 1
1.2 Who should read this document 1
1.3 Related documents 1
1.4 Typographical conventions 3
1.5 Acknowledgments 4

2 The purpose of log files 5

2.1 Logging levels 5
2.2 Logging cycle periods 5
2.3 Text and binary log files 5
2.4 Configuration substitution characters 6

3 Default log file settings 7

3.1 Liberator 7
3.2 Transformer 8
3.3 DataSource applications 8
C DataSource applications 8
Java DataSource applications 9
DataSource.NET applications 9
3.4 StreamLink 9
StreamLink JS / StreamLink.NET 10
3.5 KeyMaster 10

4 Development environment: enabling maximum logging 11

4.1 Liberator 11
Changing the logging level of event and auth module logs 11
Enabling server-side RTTP logging 12
4.2 Transformer 12
4.3 DataSource applications 12
C DataSource Applications 12
Java DataSource applications 13
DataSource.NET applications 14
4.4 StreamLink 15
StreamLink JS 15
StreamLink Java 16

StreamLink.NET 17
StreamLink iOS 18
4.5 KeyMaster 18

5 Production environment: recommendations 19

5.1 Logging level 19
5.2 Logging cycle period (packet logs) 19

6 The investigation process 21

6.1 Identifying the problem 21
6.2 Identifying the log files you need to review 21
6.3 Identifying the data you need to look at in log files 23

7 Tracing an object request using Liberator log files 24

8 Example investigations 26

8.1 Example 1: Liberator – Integration Adapter 26
Starting the components 26
Logging in 28
Object requests 30
Stopping the Integration Adapter 35
8.2 Example 2: Liberator – Transformer – Integration Adapter 37
Object requests 37
8.3 Example 3: A 'Data Unavailable' message is displayed 41

9 If you need to contact Caplin Support 45

10 Appendix A: Packet logs and logcat 46

10.1 Displaying fields and flags by name 46
10.2 Redirecting logcat output to a text file 47
10.3 Inspecting real time packet logs 47
10.4 Splitting packet logs 48

11 Glossary of terms and acronyms 49

1 Preface

1.1 What this document contains

This document explains how you can use the log files produced by **Caplin Platform** components to compare expected system behavior with actual system behavior when troubleshooting a system based on the Caplin Platform.

About Caplin document formats

This document is supplied in Portable document format (*.PDF* file), which you can read on-line using a suitable PDF reader such as Adobe Reader®. The document is formatted as a printable manual; you can print it from the PDF reader.

1.2 Who should read this document

This document is intended for developers, testers and system administrators.

1.3 Related documents

- **Caplin Liberator Administration Guide**

  Contains a list of Caplin Liberator's log and debug messages.

  For a description of the Caplin Liberator server and its place within the Caplin Platform, and reference information about the most important configuration items, see the Liberator pages of the Caplin Developers' web site.

- **DataSource C API Documentation**

  The reference documentation for the C DataSource API.

- **Caplin Integration Suite for Java API Documentation: DataSource API section**

  The reference documentation for the Java DataSource API.

- **DataSource.NET API Dcoumentation**

  The reference documentation for the DataSource.NET API.

- **Caplin Platform: Server-side RTTP Logging**

  Explains how to set up server-side (Liberator) logging of RTTP messages between Liberator and a client communicating over RTTP, and how to interpret the resulting log files.

- **StreamLink JS API Documentation**

  The reference documentation for the StreamLink JS API.
  Includes examples of how to use the API.

- **StreamLink Java API Documentation**

  The reference documentation for the StreamLink Java API.
  Includes examples of how to use the API.

- **StreamLink.NET API Documentation**

The reference documentation for the StreamLink.NET API. Includes examples of how to use the API.

- **StreamLink iOS API Documentation**

  The reference documentation for the StreamLink iOS API. Includes examples of how to use the API.

- **StreamLink Android API Documentation**

  The reference documentation for the StreamLink Android API. Includes examples of how to use the API.

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 Acknowledgments

2 The purpose of log files

Log files record error notifications, and the events that occur when a **Caplin Platform system** is up and running (such as when components start and stop). Log files also record conversations between **DataSource peers** and the parameters that are sent between them, such as the username or the session ID of a new user logging into the system, trade information, filtering options, and **DataSource** information. **Logs** recorded at a specific time show what was going on at that time in the system (for example, if a user was disconnected, if a user initiated a new trade, a filtering query, and so on).

As each Caplin Platform system has a unique setup that can change with time, Caplin cannot replicate every incident. This is why Caplin Support will ask for log files when you report a new incident. In most cases the log files help Caplin Support to identify the root cause of the incident.

2.1 Logging levels

The messages that can appear in a log file are each assigned a **log level** by the application developer, and the application only records a log message if the following conditions exist:

- The application branches to the code that records the log message, such as when the application encounters an error or some other condition that the developer wants to record.
- The application is configured to record log messages at the level assigned to the log message.

Log levels have a hierarchical order. This means that if an application is configured to record log messages at a particular level, then messages at that and higher log levels will be recorded in the log file.

For example, **Liberator** can be configured to record event log messages at any one of following logging levels (in hierarchical order): CRITICAL (only records critical errors, which produces the smallest log file sizes), ERROR, NOTIFY, WARN, INFO, and DEBUG (records all errors and events, which can produce very large log file sizes). If the configured log level of Liberator is NOTIFY, Liberator will record events at the NOTIFY, CRITICAL, and ERROR levels.

2.2 Logging cycle periods

Log files can become quite large if they are not regularly cycled throughout the day. For this reason, log files can be configured to cycle at regular intervals. When a log file cycles, the current log file is closed and a new log file is opened.

2.3 Text and binary log files

All log files created by Caplin Platform components are pure text files that can be viewed in a text editor, except packet logs, which are binary files that can only be viewed using Caplin's **logcat** utility.

For further information about using the **logcat** utility to view packet logs, see Appendix B: Packet logs and logcat 46.

2.4 Configuration substitution characters

The configuration options for **C DataSource** applications, **Java DataSource** applications and **DataSource.NET** applications can have values that include substitution characters. The following substitution characters are used in the logging configuration examples shown in this document.

Substitution characters	Description	Example configuration item	Description of example
%r	The root directory of the installed application.	**log-dir %r/var**	The configured location of log files (in the *var* subdirectory of the installed application).
%u	The day of the week as an integer (range 1 to 7, Monday being 1).	**log-cycle-suffix %u**	The suffix appended to the name of log files when the log files are cycled.

Liberator and **Transformer** are C DataSource applications.

3 Default log file settings

Each **Caplin Platform** component has its own default log files, and its own default configuration settings for these files.

3.1 Liberator

Liberator logging is enabled by default. Liberator produces a large number of log files that can be categorized according to the available log level settings:

- ***Multi log-level log files:*** This category of log file can be configured to one of several log levels. The log level setting can be changed in the log file configuration, or dynamically using UDP or monitoring commands. The event log (*event-rttpd.log*) and auth module logs (*javaauth.log*, *xmlauth.log*, and *tokenauth.log*) are in this category.
- ***Single log-level log files:*** This category of log file has only one log level: `DEBUG`. The following log files are in this category:

  session-rttpd.log, *object-rttpd.log*, *request-rttpd.log*, HTTP logs (*http-access-rttpd.log* and *http-error-rttpd.log*), and the Liberator packet log (*packet-rttpd.log*).

The default settings of Liberator log files are summarized in the following tables:

The Event Log

Log File Name	Default Directory	Default Logging Level	Default Logging Cycle
event-rttpd.log	*%r/var*	INFO	Logs cycle every 24 hours at 4am, log file suffix %u

Auth Module Logs

Default Directory	Default Logging Level	Default Logging Cycle
%r/var	INFO	Logs cycle every 24 hours at 4am, log file suffix %u

All Other Logs

Default Directory	Fixed Logging Level	Default Logging Cycle
%r/var	DEBUG	Logs cycle every 24 hours at 4am, log file suffix %u

3.2 Transformer

Transformer logging is enabled by default. Transformer only produces two categories of log file: the Transformer event log and the packet log.

The default settings of Transformer log files are summarized in the following tables:

The Event Log

Log File Name	Default Directory	Default Logging Level	Default Logging Cycle
transformer.log	*%r/var*	INFO	Logs cycle every 24 hours at 4am, log file suffix %u

The Packet Log

Log File Name	Default Directory	Fixed Logging Level	Default Logging Cycle
packet-transformer.log	*%r/var*	DEBUG	Logs cycle every 24 hours at 4am, log file suffix %u

Note: More log files are produced if Transformer uses any Transformer Modules.

3.3 DataSource applications

The way that log files are configured, and whether they are enabled by default, depends on the software library used by the **DataSource application**.

C DataSource applications

C DataSource logging is enabled by default. The default settings of C DataSource log files are summarized in the following tables:

The event log

Log File Name	Default Directory	Default Logging Level	Default Logging Cycle
event-<appname>.log	*%r/var*	INFO	Logs cycle every 24 hours at 4am, suffix %u

The packet log

Log File Name	Default Directory	Fixed Logging Level	Default Logging Cycle
packet-<appname>.log	*%r/var*	DEBUG	Logs cycle every 24 hours at 4am, suffix %u

The default name of a log file depends on the name of the C DataSource application. For example, if the name of the C DataSource application is *datasrc*, the default name of the event log is *event-datasrc.log*, and the default name of the packet log is *packet-datasrc.log*.

For more about configuring logging for DataSource applications, see Logging in DataSource applications

Java DataSource applications

Java DataSource applications created using the Caplin Integration Suite (CIS)

Java DataSource applications created as Platform blades using using the Caplin Integration Suite (CIS) have logging enabled by default. The logging configuration is defined as for a C-based DataSource, using a DataSource configuration file, and is then automatically converted to XML when you deploy the blade. Please refer to the previous section, C DataSource applications 8, for more information.

Java DataSource applications created without the CIS

In Java DataSource applications that are created without the CIS, logging is not enabled by default. For further information about configuring logging for such applications, see the section Java DataSource applications 13 in Development Environment: Enabling maximum logging 11.

DataSource.NET applications

DataSource.NET applications do not have logging is not enabled by default.

For further information about configuring logging for a DataSource.NET application, see C DataSource / DataSource.NET 12 in Development Environment: Enabling maximum logging 11.

3.4 StreamLink

The **RTTP** messages that are sent between the **client application** and Liberator can be found in the **StreamLink** log files.

The way that the log files are configured, and whether they are enabled by default, depends on the software library used by the client application.

StreamLink logs are not enabled by default, and the way that you enable logging depends on the software library used by the client application. For further information, see StreamLink JS 15, StreamLink Java 16 and StreamLink iOS 18 in Development Environment: Enabling maximum logging 11.

StreamLink JS / StreamLink.NET

StreamLink logs are not enabled by default, and the way that you enable logging depends on the software library used by the client application. For further information, see StreamLink JS 15 and StreamLink.NET 17 in Development Environment: Enabling maximum logging 11.

3.5 KeyMaster

When KeyMaster is used for authentication, logging is enabled by default and records information about user logins.

Log File Name	Default Directory	Default Logging Level	Default Logging Cycle
servlet.log	Tomcat server: the top level installation directory of the server. JBoss server: the directory where the server was started. BEA WebLogic server: the domain directory where the server was started.	ALL	Logs cycle at 12am every 24 hours

4 Development environment: enabling maximum logging

This section describes how to enable maximum logging by Caplin Platform components for development purposes and debugging. In each case, log file names and directories are the defaults, as described in Default log file settings 7.

Note: Maximum logging should only be used for troubleshooting, as the performance of the Caplin Platform applications is reduced when maximum logging is enabled. For a production system, the recommended logging levels are described in Production environment: recommendations 19.

4.1 Liberator

The event and auth module logs are the only Liberator logs that have a logging level you can change. Server-side RTTP logging is disabled by default, but can be enabled for a particular user.

Changing the logging level of event and auth module logs

The Liberator event and auth module logs are enabled for maximum logging by setting the **log-level** configuration option to **Debug** in the Liberator and auth module configuration files, as shown in the following table:

Event log settings

Log File Name	Configuration Option	Configuration File	Configuration File Directory
event-rttpd.log	**log-level DEBUG**	*rttpd.conf*	*%r/etc*

Auth module log settings

Log File Name	Configuration Option	Configuration File	Configuration File Directory
javaauth.log	**log-level DEBUG**	*javaauth.conf*	*%r/etc*
xmlauth.log	**log-level DEBUG**	*xmlauth.conf*	*%r/etc*
tokenauth.log	**log-level DEBUG**	*tokenauth.conf*	*%r/etc*

The auth module that Liberator uses to authenticate users is defined by the **auth-module** configuration option in the Liberator configuration file *rttpd.conf* (see the **Caplin Liberator Administration Guide** for further information).

Enabling server-side RTTP logging

Server-side RTTP logging is disabled by default but can be enabled on a per-user basis, either in the Liberator configuration file or dynamically using the Caplin Management Console.

Tip: For further information about how to set up server-side RTTP logging, refer to the document **RTTP: Server-side RTTP Logging**.

Note: All server-side RTTP log files are created in the directory *%r/var/rttp* by default, where *%r* is the current working directory. Please make sure the *rttp* directory exists before you enable RTTP logging.

4.2 Transformer

The Transformer event log is enabled for maximum logging by setting the **log-level** configuration option to **Debug** in the Transformer configuration file *rttpd.conf*, as shown in the following table:

Log File	Configuration Option	Configuration File	Configuration File Directory
transformer.log	**log-level DEBUG**	*transformer.conf*	*%r/etc*

Note: If Transformer uses any Transformer Modules, the logging level of each module can be changed using the **log-level** configuration option in the corresponding module configuration file.

4.3 DataSource applications

The way that DataSource applications are enabled for maximum logging depends on the software library used by the DataSource application.

C DataSource Applications

C DataSource applications use the same plain text configuration options as Liberator and Transformer. The DataSource event log is enabled for maximum logging by setting the **log-level** configuration option to **Debug** in the DataSource configuration file d*atasource.conf*, as shown in the following table:

Log File Name	Configuration Option	Configuration File Name	Configuration File Directory
event-datasrc.log	**log-level DEBUG**	*datasource.conf*	*%r/etc*

Because logs are saved to the *%r/var* directory by default, there is no need to specify the log file directory. If DataSource.NET is launched from an IIS web server, it is recommended that you set the name and directory of the log file using the following configuration options, so that log files are written to a known location.

Configuration Option	Description	Example
event-log	Defines the name of the log file.	**event-log MyLogFileName.log**
log-dir	Defines the directory of the log file.	**log-dir MyLogFilePath**

Java DataSource applications

Setting up logging through configuration

Java-based DataSource applications are configured in the same way as C-based DataSource applications.

For Java DataSource applications that have been built using the **Caplin Integration Suite** (CIS) you can configure maximum logging in the same way as described in C DataSource Applications 12.

For Java DataSource applications that haven't been built using the Caplin Integration Suite, to configure maximum logging, follow these steps:

1. Create a folder called *etc* within your application.
2. Create a file called *datasource.conf*, where you configure the peer that the DataSource application is connecting too in the same way as for C-based DataSource applications.
3. Set the log-level configuration option to DEBUG (this is the same as FINE):

Log File Name	Configuration Option	Configuration File Name	Configuration File Directory
event-datasrc.log	**log-level DEBUG**	*datasource.conf*	*%r/etc*

4. Create a folder called *var* within your application.

Once you have started the application, you will see the logs created within the *var* folder.

For debugging purposes, we recommend using one of the following log levels:

DEBUG (or **FINE**): used for tracing messages

FINER: used for fairly detailed tracing messages

FINEST: used for the most finely tracing messages

Setting up logging in the application code

There is another way to enable logging from within the application code, by passing the logger in to the factory that creates an instance of the DataSource object. This is the recommended approach as it allows you to have full control of how event logging is implemented.

Here is an example of how to do this:

```
import java.io.File;
import java.io.IOException;
import java.util.logging.Logger;

import org.xml.sax.SAXException;

import com.caplin.datasource.DataSourceFactory;
import com.caplin.datasrc.DataSource;

public class DataSourceConstruction
{
   public static void main(String[] args) throws IOException, SAXException
   {
      com.caplin.datasource.DataSource newDS =
                           DataSourceFactory.createDataSource(
      args, Logger.getAnonymousLogger());
   }
}
```

This example outputs the logs in the console XML configuration schema with a logging level of INFO. To change the logging level (say to FINEST), just set it in the logger:

This example outputs to the console all log messages with a logging level of INFO. To change the logging level just set it in the logger:

```
logger.setLevel(Level.FINEST);
```

For more about setting up logging from within the application code, see the DataSource API section of the **Caplin Integration Suite for Java API Documentation**.

DataSource.NET applications

Setting up logging through configuration

DataSource.NET applications are configured in the same way as C-based DataSource applications.You enable the DataSource event log is enabled for maximum logging by setting the `log-level` configuration item to DEBUG in the DataSource configuration file *datasource.conf* as follows:

Log File Name	Configuration Option	Configuration File Name	Configuration File Directory
event-datasrc.log	**log-level DEBUG**	*datasource.conf*	*%r* (within the project)

To specify where the logs will be saved, just specify the `log-dir` configuration item and point it to the desired path.

Setting up logging in the application code

Another way of logging in DataSource.Net applications is by implementing the `ILogger` interface to receive log messages from the Caplin DataSource API. You can use the `ConsoleLogger` class that implements the `ILogger` interface to output all log messages to the console.

Example

```
public class Demosource
{
   private IDataSource dataSource;
   private ILogger logger;

   public Demosource(string[] args)
   {
      logger = new ConsoleLogger();
      dataSource = new DataSource("demosource.conf", args, logger);

      new MyDataProvider(dataSource);
   }
}
```

Tip: If you choose to implement your own ILogger make sure you wrap the log files so that the individual files don't get too big.

4.4 StreamLink

The way that StreamLink applications are enabled for maximum logging depends on the software library used by the client application.

StreamLink JS

Logging for **StreamLink JS** is disabled by default, but can be enabled by simply appending the query string `?debug=<required-logging-level>` to the application URL.

For example, the URL of the application will look something like this:

http://myapp.novobank.com:50180

To enable logging and set the logging level set to `FINER`, append the query string *?debug=finer* at the end of the URL:

http://myapp.novobank.com:50180?debug=finer

The StreamLink JS log opens in a separate Debug browser window (the StreamLink console).

You may find the following log levels useful when debugging connection problems:

- **FINE** is used for tracing messages. It includes the size of the response and update message queues for messages received from Liberator.
- **FINER** is used for fairly detailed tracing messages. It includes the RTTP messages sent in each direction between the client application and Liberator.
- **FINEST** is used for the most detailed tracing messages. It includes the HTTP headers of the HTTP communication with Liberator.

Note: Depending on the volume of data that is sent, low-level logging can severely impact the performance of the client. For that reason StreamLink logging should be only used for debugging purposes.

For more about this, see the **StreamLink JS API Documentation**.

StreamLink Java

The `Logger` interface allows StreamLink Java log messages to be written to a destination of your choice. To obtain an instance of the interface just call `streamLinkInstance.getLogger()` on your `StreamLink` instance.

You can use `caplin.streamlink.Logger` to write StreamLink's log messages to some other destination, such as a window that also contains log messages originating from your application. To do this, implement a `LogListener` that receives the StreamLink messages and logs them to the required destination, as this example shows:

```
// Set up log listener.

LogListener loglistener = new LogListener() {

@Override
   public void onLog(LogInfo logInfo) {
      System.out.println(logInfo);
   }
};
```

Once you have implemented the `LogListener`, you can obtain an instance of `Logger` and call `addListener()` to attach the `LogListener` to the instance. You can also choose the logging level in this step:

```
streamLink.getLogger().addListener(loglistener, LogLevel.FINEST);
```

This example outputs the log messages to the console. For an example off how to display the log messages in a window, take a look at the StreamLink swing demo application that comes with the StreamLink Java kit.

Log levels

The logging level determines how much information is logged, and you may find the following levels useful when debugging problems:

- **FINE** is used for tracing messages. It includes the size of the response and update message queues for messages received from Liberator.
- **FINER** is used for fairly detailed tracing messages. It includes the RTTP messages sent in each direction between the client application and Liberator.
- **FINEST** is used for the used for the most detailed tracing messages. It includes the HTTP headers of the HTTP communication with Liberator

For more about this, see the **StreamLink Java API documentation**.

StreamLink.NET

The `ILogger` interface allows StreamLink.NET log messages to be written to a destination of your choice. To obtain an instance of the interface just call `streamLinkInstance.getLogger()` on your StreamLink instance.

You can use `caplin.streamlink.Logger` to write StreamLink's log messages to some other destination, such as a window that also contains log messages originating from your application. To do this:

1. Implement the `ILogListerner` interface.
2. Create an instance of the `ILogListerner` implementation.
3. Attach this instance to an instance of `ILogger`.

The following code example shows a simple anonymous implementation of this interface:

```
using Caplin.StreamLink;
using System;
namespace com.caplin.streamlink.examplesnippets.logging
{
   public class LogListenerSnippet
   {
      public LogListenerSnippet(IStreamLink streamLink)
      {
         streamLink.Logger.AddListener(new ExampleLogListener(),
                                       LogLevel.FINEST);
      }
      class ExampleLogListener : ILogListener
      {
         public void OnLog(ILogInfo logInfo)
         {
            Console.WriteLine(logInfo.Message);
         }
      }
   }
}
```

Log levels

The logging level determines how much information is logged, and you may find the following levels useful when debugging problems:

- **FINE** is used for tracing messages. It includes the size of the response and update message queues for messages received from Liberator.
- **FINER** is used for fairly detailed tracing messages. It includes the RTTP messages sent in each direction between the client application and Liberator.
- **FINEST** is used for used for the most detailed tracing messages. It includes the HTTP headers of the HTTP communication with Liberator

For more about this, see the **StreamLink .NET API documentation**.

StreamLink iOS

StreamLink iOS does not automatically create log files on the device. The implementation of logging on iOS devices depends on the requirements of your application, which should take into account ease of support and the expectations of your application's end users.

To obtain logs from StreamLink iOS, you can use the `SLConsoleLogger` with the logging level set to the most detailed level (**FINEST**). Then run the application under the XCode debugger, and capture the logging information from the console window.

You may find the following log levels useful for development purposes:

- **SL_LOG_DEBUG** is used for tracing messages.
- **SL_LOG_FINER** is used for fairly detailed tracing messages.
- **SL_LOG_FINEST** is used for used for the most detailed tracing messages.

Note: We strongly recommend that you only use `SLConsoleLogger` during development and not in a product deployment.

For more about this, see the **StreamLink iOS API documentation**.

4.5 KeyMaster

KeyMaster is enabled for maximum logging by default, or by setting the **key.generator.level** configuration option to **ALL** in the XML configuration file *web.xml*, as shown in the following table:

Log File	Configuration Option	Configuration File	Configuration File Directory
server.log	**key.generator.level=ALL**	*web.xml*	*%r/WEB-INF*

Tip: You may need to enable maximum logging in the configuration file if logging is already enabled, but the logging level is not set to **ALL**.

Note: The default logging level is **ALL**, which means that all events are logged. When KeyMaster is not being debugged, the recommended logging level is **WARNING**.

5 Production environment: recommendations

The Caplin Platform can generate very large log files in a production environment. For this reason the following logging levels and cycle periods are recommended.

Note: All log file configuration settings must be verified before the Caplin Platform system is put into production. In particular, sufficient disk space must be available for the configured logging level and logging cycle period.

5.1 Logging level

In a production environment, the recommended logging level for all Caplin Platform products except KeyMaster is `INFO`. The recommended logging level for KeyMaster is `WARNING`. All log files should be retained for 7 days.

Tip: The recommended logging levels ensure significant problems with Caplin Platform components are recorded.

5.2 Logging cycle period (packet logs)

Liberator and Transformer packet logs can become quite large if they are not regularly cycled throughout the day. For this reason it is recommended that packet logs are cycled every 15 to 30 minutes.

Example configuration

```
add-log
name  packet_log
  period  15
  suffix  .%u%H%M
end-log
```

In the example configuration shown above, packets logs are configured to cycle every 15 minutes, creating a new log file of the form *packet-rttpd.%u%H%M*.

When specifying the suffix of the packet log, the following substitution characters can be used in the configuration:

Substitution characters	Description
%u	The day of the week as an integer (range 1 to 7, Monday being 1).
%H	The hour as a decimal number (range 00 to 23).
%M	The minutes past the hour.

Because the log file suffix in this example contains %u, packet logs will be overwritten every 7 days.

Tip: When reporting a problem to Caplin's Issue Management System (Jira), small file sizes also reduce the time it takes to transfer each log file. See If you need to contact Caplin Support 45 for contact details.

Tip: Appendix B 46 has tips on how to split large log files.

6 The investigation process

When you are investigating an issue, the following structured approach is recommended.

1. Identify the problem.

2. Identify the log files you need to review.

3. Identify the data you need to look at in these log files.

A structured approach will help you to narrow your search area. There will be fewer log messages to read and the log files will be easier to interpret.

6.1 Identifying the problem

When identifying the problem, try to gather as much relevant information as possible. Typical questions you can try to answer are:

- Did the incident occur at a particular time?
- Did an operation at the client application fail?
- Is the problem occurring consistently or did it occur only once?
- Do you know which system components are affected?
- Have any system changes been made recently?
- Is the problem affecting a new user?

The actual questions you ask depends on the problem you are trying to solve. The answer to these questions will help you identify the messages you expect to see in log files, allowing you to compare the expected system behavior with the actual system behavior.

6.2 Identifying the log files you need to review

Once you have identified the problem, try to identify the Caplin Platform components that may be contributing to the problem. Typical questions you can ask are:

- Is the problem caused by Liberator or Transformer?
- Is it a DataSource issue or an issue with the **client application** (such as **Caplin Trader**)?

If you can answer these questions, then you can limit the component log files you need to look at.

When you have identified the Caplin Platform components that may be contributing to the problem, there are eight categories of log file that you can review:

Log Category	Description
Auth module logs (Liberator only)	Auth module logs record information about the authorization and authentication of Liberator users.
Event logs	Event logs record ongoing component activity (such as the component starting or stopping). The more verbose the configured logging level, the more information you can obtain from these logs.
HTTP logs (Liberator only)	HTTP access logs record HTTP requests. HTTP error logs record HTTP requests that cause 'object not found' errors.
Object logs (Liberator only)	Object logs record request and discard commands for objects, and whether or not those commands were successful. Object logs are only created by Liberator.
Packet logs	Packet logs record the messages that are exchanged between DataSource peers, such as updates, requests, discards, peer status information, and so on. These logs tell you if a request sent from one peer was received by another.
Request logs (Liberator only)	Request logs record the RTTP messages sent by client applications to Liberator. Request logs are only created by Liberator.
Session logs (Liberator only)	Session logs record information about Liberator sessions and events, including the session ID allocated to a user. Session logs are only created by liberator.
Client logs	Client logs record the RTTP messages that are sent between the client application and Liberator. Client logs can either be viewed at the client side (see StreamLink 15) or at the Liberator server side (see Enabling server-side RTTP logging) 12.

The messages that can appear in Liberator logs are described in the document **Caplin Liberator Administration Guide**.

6.3 Identifying the data you need to look at in log files

When you have identified the problem and the components that may be contributing to the problem, try to identify the data that you need to look at in the log files. In a production environment, a simple packet log can contain thousands of lines, and you don't want to look at every line!

For example, if you know that a user was disconnected from the system at a particular time, you could look in the session logs for messages that might indicate the reason for the disconnection.

Tools like the Linux **grep** utility can be used to filter the content of log files, reducing the amount of information you need to review. The following example filters the packet log *packet-rttp.log* for lines that contain the text `PEERINFO`.

Example packet log filter

```
...bin/logcat packet-rttp.log | grep PEERINFO
```

Because the Caplin **logcat** utility must be used to view the content of packet log files, the output of **logcat** is piped to **grep**. The filtered output of this command would only show lines that contain the string `PEERINFO`:

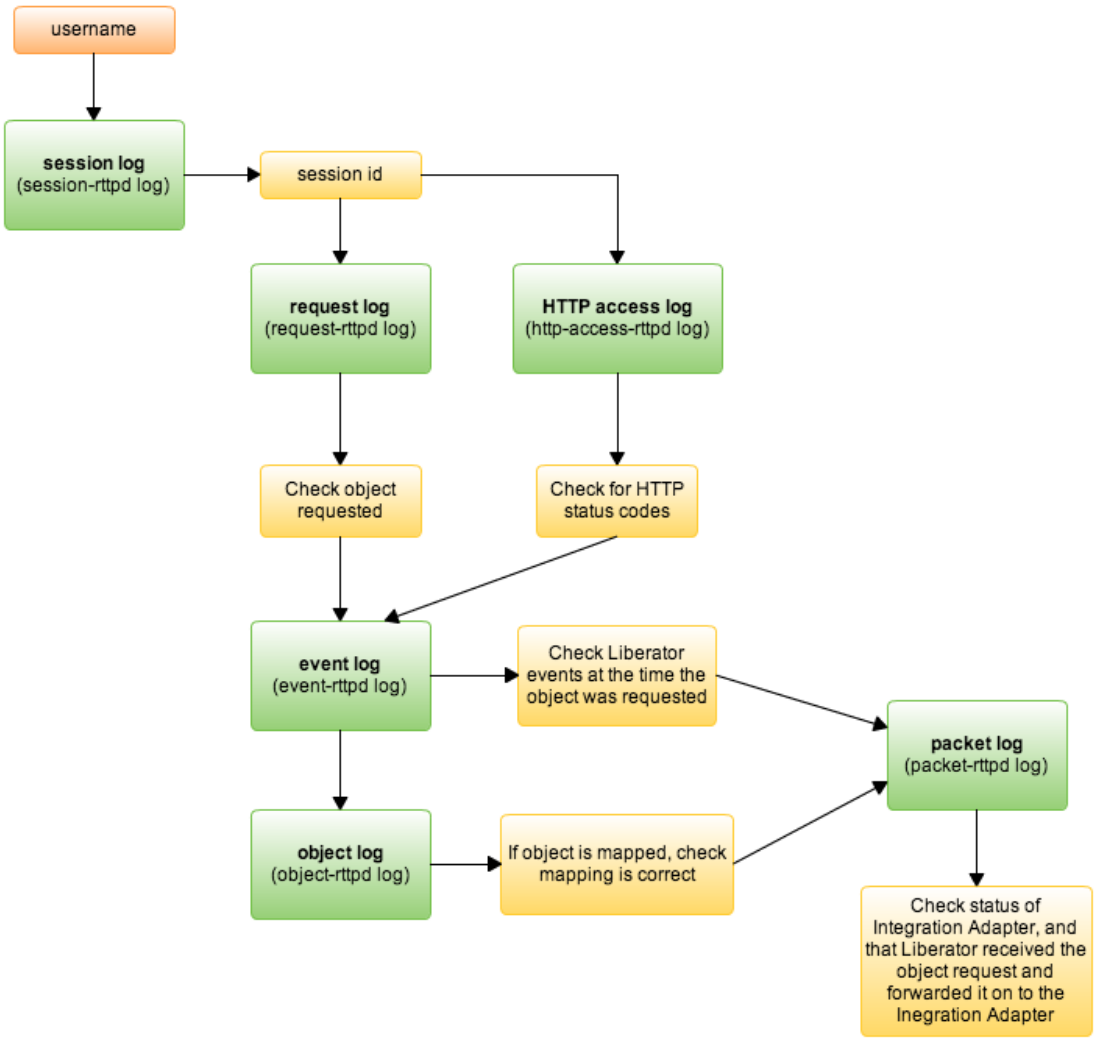
Example filter output

```
2014/08/11-09:05:29.425 +0100: 192.168.50.51 < PEERINFO 1795 DataProviderA1 0  ACTIVE
2014/08/11-09:05:29.425 +0100: 192.168.50.51 > PEERINFO 101 liberator1 0  BROADCAST
```

For further information about viewing packet logs using the **logcat** utility, see Appendix B: Packet logs and logcat 46.

7 Tracing an object request using Liberator log files

The following diagram shows some of the log files produced by Liberator, and how they can help you to assess the state of the system when an **end-user** logs in and the client application requests an object:



Tracing an object request using Liberator log files

Other DataSource applications, such as Transformer, also produce event and packet logs that can help you to assess the state of the system.

session-rttpd.log

If you know the **username** of the end-user, you can verify that they successfully logged in by inspecting the recorded LOGIN messages. This file also records the **session id** of the logged in user, which you will need when verifying object requests in other log files.

http-access-rttpd.log

This file logs URL requests and HTTP status codes.

request-rttpd.log

If you don't know which objects were requested, you can look in the request log for object requests associated with the client **session id**.

> **Note:** Although it is possible to check the request in the packet logs if you know which object was requested, sometimes it is better to verify the request using the request logs, as you can also verify that the request came from the correct client by looking at the **session id**.

event-rttpd.log

When you have identified the requested object, you can look at the event log to assess the state of Liberator at the time the object was requested.

object.rttpd.log

You can also look at the object log to see if the requested object is mapped, and if it is, that the object mapping is correct. For example, if the object `/FX/EUR` is mapped to `/EUR/FX`, Liberator returns the object `/EUR/FX` in response to a client request for `/FX/EUR`.

packet-rttpd.log

Liberator packet logs record object request messages. In this way you can verify that Liberator received the object request and forwarded it on to the correct DataSource. You can also look for other messages that could indicate there was a problem providing the requested data.

8 Example investigations

The following examples show how log files can be used to asses the state of a Caplin Platform system when an end-user logs in and the client application requests an object.

8.1 Example 1: Liberator – Integration Adapter

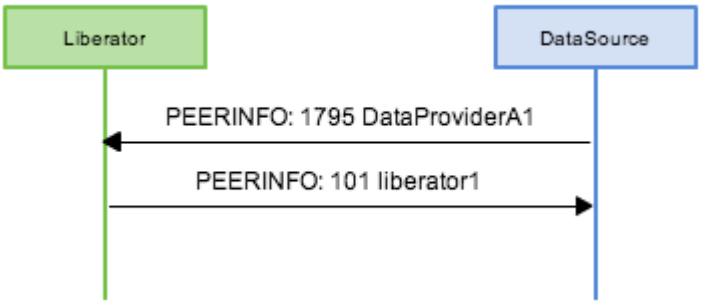
In this example investigation, the Caplin Platform system consists of a Liberator and an Integration Adapter.

Liberator and Integration Adapter

Log files record the state of the system when components start and stop, and when the client application requests an object. You can use this information to determine whether or not the system is behaving as expected.

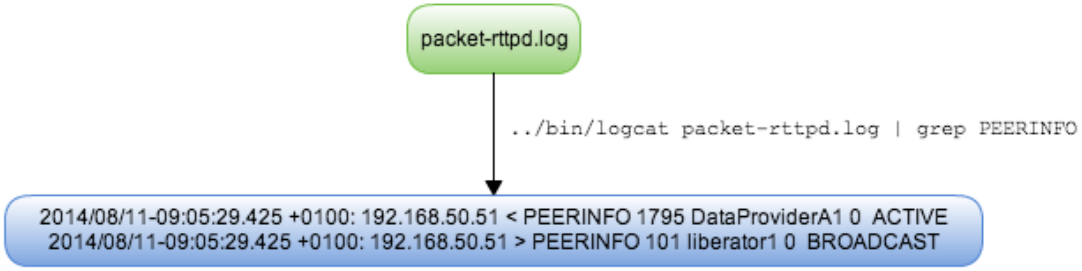
Starting the components

When an Integration Adapter (or other DataSource application), starts and connects to Liberator, the Liberator packet log *packet-rttpd.log* records the `PEERINFO` messages that are exchanged between Liberator and the Integration Adapter.



Liberator and Integration Adapter message exchange

The Liberator packet log is a binary file and you must use the **logcat** utility to view the content of the log file. To filter the packet log for `PEERINFO` messages, pipe the output of **logcat** to **grep**.



Using logcat and grep to filter the packet log

`The PEERINFO` messages record the date and time the connection request from the broadcast DataSource (`DataProviderA1`) was accepted by Liberator (`liberator1`):

Filtered PEERINFO messages

Date-Time +Zone	IP Origin/ Destination	In (<)/ Out (>)	Message Type	Peer ID	Peer Label	Field Not used	Peer Type
2014/08/11-09:05:29.425 +0100	192.168.50.51	<	PEERINFO	1795	DataProviderA1	0	ACTIVE
2014/08/11-09:05:29.425 +0100	192.168.50.51	>	PEERINFO	101	Liberator1	0	BROADCAST

The **StreamLink JS** and server-side RTTP logs also record information that shows the active DataSource connecting to Liberator.

Typical StreamLink JS log

```
2014/08/11-12:13:18.197 +0100 - FINER  : < 7_ 1795 DataProviderA1 DataProviderA1+IS+UP
2014/08/11-12:13:18.198 +0100 - FINER  : < 83
      DataProviderAPricingSvc1 DataProviderAPricingSvc1+IS+OK
```

In this case the StreamLink JS logging level is set to *FINER*. The amount of information that is displayed in the log is determined by the log level that you set (see StreamLink JS 15 for further information about enabling the StreamLink JS log).

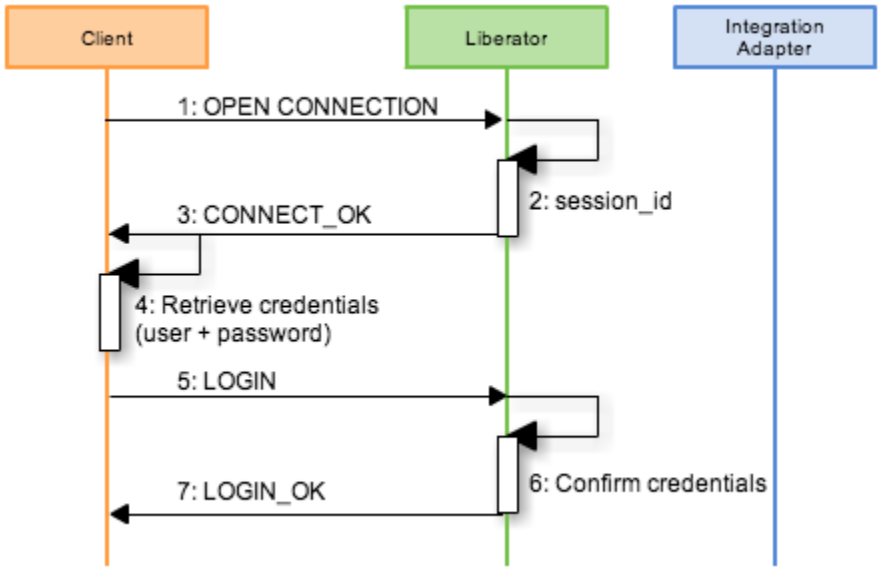
Typical server-side RTTP log

```
7_ 1795 DataProviderA1 DataProviderA1+IS+UP
83 DataProviderAPricingSvc1 DataProviderAPricingSvc1+IS+OK
```

The numbers at the beginning of each line are RTTP codes that identify the type of message. In this case `7_` identifies a 'DataSource up' message, and `83` identifies a 'Data Service OK' message. To enable server-side RTTP logging, see Enabling server-side RTTP logging 12.

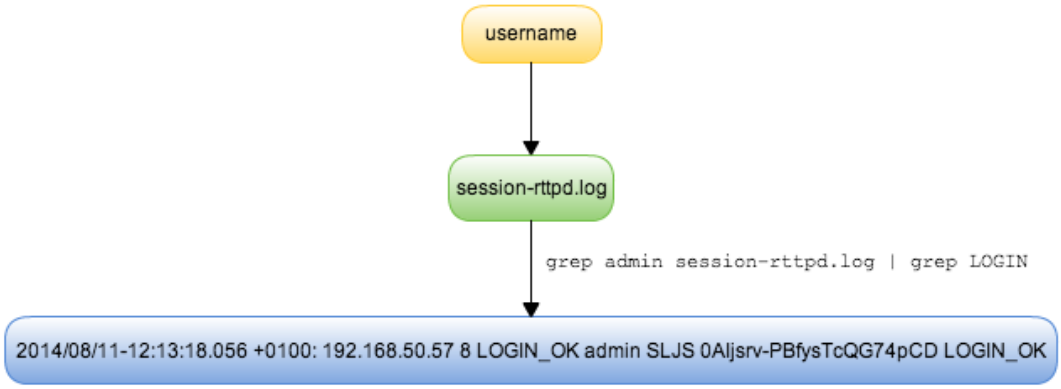
Logging in

When an end-user logs in from a client application such as Caplin Trader, Liberator allocates a session ID to the user session. You may need this information later because it is the session ID and not the username that is recorded in other log files, such as the event and object logs, when an object is requested.



Client application and Liberator message exchange

The session ID is assigned to the client as soon as it establishes a connection with the Liberator (before it has retrieved the credentials and logged in. The session ID allocated to a user session is recorded in `LOGIN` messages in the Liberator session log *session-rttpd.log*. The session log is a text file, and you can use **grep** to filter the `LOGIN` messages.



Using grep to filter the session log

The `LOGIN` message shows that username `admin` is allocated the session ID `0AIjsrv-PBfysTcQG74pCD`, and that the login was successful (`LOGIN_OK`):

Filtered LOGIN message

Date-Time +Zone	IP Address	Connection-Type	Msg-Type	Username	App-ID	Session-ID	Reason
2014/08/11-12:13:18.056 +0100	192.168.50.57	8	LOGIN_OK	admin	SLJS	0AIjsrv-PBfysTcQG74pCD	LOGIN_OK

Session logs also record the date and time that the client application disconnects, and if the session times out.

The SL4B and server-side RTTP logs also record information when a user logs in.

Typical StreamLink JS log

```
2014/08/11-12:13:17.879 +0100 - INFO   : Trying next connection:
                                        ws://supportlinux3:18082

2014/08/11-12:13:17.879 +0100 - INFO   : Using Connection Type: WebSocketConnection

2014/08/11-12:13:17.880 +0100 - INFO   : Connection state changed to: CONNECTING
                                         ws://supportlinux3:18082

2014/08/11-12:13:18.087 +0100 - FINER  : < 01 0AIjsrv-PBfysTcQG74pCD host=unknown
                                         version=2.1 server=unknown
                                         time=1407755597 timezone=0000

2014/08/11-12:13:18.092 +0100 - INFO   : Connection state changed to: CONNECTED
                                         ws://supportlinux3:18082

2014/08/11-12:13:18.093 +0100 - INFO   : Connection state changed to:
                                         RETRIEVINGCREDENTIALS ws://supportlinux3:18082

2014/08/11-12:13:18.094 +0100 - FINE   : Received credentials:
                                         Credentials [username=admin, password=admin]

2014/08/11-12:13:18.094 +0100 - INFO   : Connection state changed to:
                                         CREDENTIALSRETRIEVED ws://supportlinux3:18082

2014/08/11-12:13:18.097 +0100 - FINER  : > 0AIjsrv-PBfysTcQG74pCD+LOGIN+0+SLJS/+RTTP/
2.1+admin+admin+HttpRequestLineLength%3D%2CHttpBodyLength%3D%2CMergedCommands
%3D%2CHeartbeatInterval%3D10000

2014/08/11-12:13:18.186 +0100 - FINER  : < 1b LOGIN+OK HttpRequestLineLength=960,
                                           HttpBodyLength=65536…

2014/08/11-12:13:18.188 +0100 - INFO   : Connection state changed to: LOGGEDIN
                                         ws://supportlinux3:18082
```

Some of the log lines contain RTTP codes:

- `01`, which identifies a ‘Connection greeting’ message. This is the message where the Liberator sends the session ID to the client.
- `1b`, which identifies a ‘Login ok’ message.

Typical Liberator session log

```
2014/08/11-12:13:17.940 +0100: 192.168.50.57 8 OPEN 0AIjsrv-PBfysTcQG74pCD

2014/08/11-12:13:18.056 +0100: 192.168.50.57 8 LOGIN_OK admin SLJS
                                               0AIjsrv-PBfysTcQG74pCD LOGIN_OK
```

Typical server-side RTTP log

```
01 0AIjsrv-PBfysTcQG74pCD host=unknown version=2.1 server=unknown time=1407755597
      timezone=0000

0AIjsrv-PBfysTcQG74pCD LOGIN 0 SLJS/ RTTP/2.1 admin admin HttpRequestLineLength=,
      HttpBodyLength=,MergedCommands=,HeartbeatInterval=10000

1b LOGIN+OK HttpRequestLineLength=960,HttpBodyLength=65536
```

Object requests

When a client application requests an object (such as an instrument) from Liberator, the messages that are exchanged between Liberator and the Integration Adapter depend on whether the Adapter that supplies the object is a broadcast DataSource or an active DataSource application.

Broadcast DataSource application message exchange

A broadcast DataSource application does not wait for an object request, but sends the objects it has to all connected DataSource applications as soon as a connection is established. When a broadcast DataSource application receives an update to an object, the update is also sent to all connected DataSource applications.

The objects that Liberator receives are recorded as `DATAUPDATE` messages in the Liberator packet log *packet-rttpd.log*. To filter the packet log for `DATAUPDATE` messages, you can pipe the output of **logcat** to **grep**.

Filtered DATAUPDATE message

```
../bin/logcat -l packet-rttpd.log | grep DATAUPDATE
```

> **Tip:** We recommend using the `-l` option of the `logcat` command so that all the message flags are displayed.

`DATAUPDATE` messages record the date and time the object was received, and the subject of the received object.

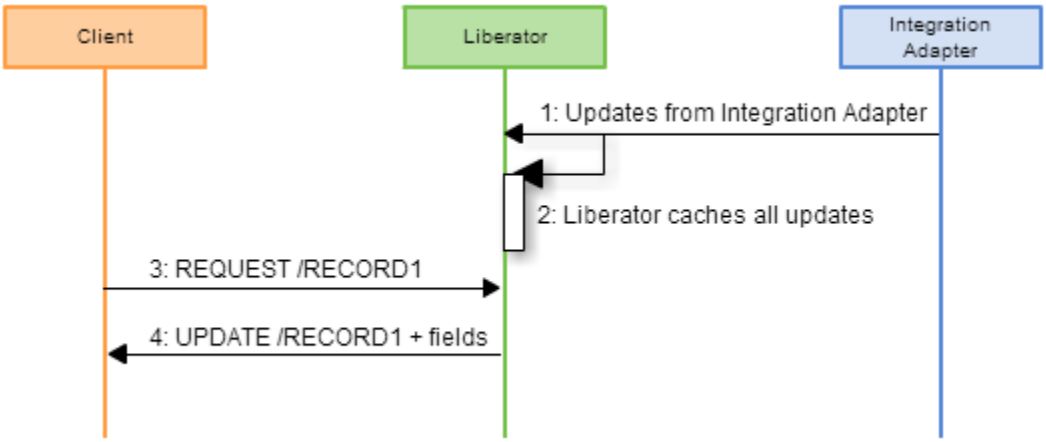
Typical DATAUPDATE message

Date-Time +Zone	IP Origin	In (<)	Msg-Type	Peer-Label	SeqNum	Flag
2014/08/11-14:59:14.372 +0100	192.168.50.51	<	DATAUPDATE2	DataProviderA1	0	F_IMAGE| F_CREATEPARENT | F_CREATEOBJECT (4144

Subject	Type	Num of Fields	Field number =Value	Field number =Value
/RECORD1	222	2	-10022=Subscription-1	-10001=Mon Aug 11 14:59:23 BST 2014

In this case the subject of the received object is `/RECORD1`, and the object has two fields

When a client application requests an object supplied by a broadcast DataSource application, Liberator can send the object to the client without first requesting the object from the DataSource application. This is because Liberator caches all objects supplied by the broadcast DataSource application.



Message exchange between client application, Liberator, and a broadcast DataSource application (Integration Adapter)

In the example shown above, the client application requests the object /RECORD1, which Liberator already has in its cache.

Object request and update messages, to and from the client application, are recorded in the StreamLink JS and server-side RTTP logs

Typical StreamLink Java log

```
2014/08/11-15:23:23.065 +0100 - INFO   : StreamLink subscribe /RECORD1 called.

2014/08/11-15:23:23.066 +0100 - FINER  : > 0Km7En22D1-8W65nbpgq7N+REQUEST+/RECORD1

2014/08/11-15:23:23.181 +0100 - FINER  : < 3U0001 /RECORD1 1=Subscription-1
                                           7=Mon+Aug+11+14:59:23+BST+2014 B=211
                                           C=DataProviderAPricingSvc1
```

In this example the text `0Km7En22D1-8W65nbpgq7N` is the session ID, and `3U0001` is a code that identifies the message type to Caplin Support:

In `3U-0001` consists of:

- `3U` - the RTTP code
- `0001` - the object number

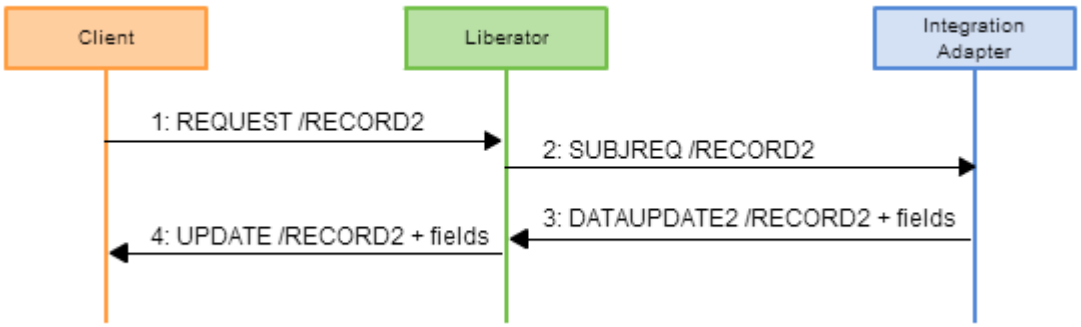
Typical server-side RTTP log

```
0Km7En22D1-8W65nbpgq7N REQUEST /RECORD1

data: 3U0001 /RECORD1 1=Subscription-1 7=Mon+Aug+11+14:59:23+BST+2014 B=211
      C=DataProviderAPricingSvc1
```

Active DataSource application message exchange

An active DataSource application accepts requests for an object, and only sends an object or update to an object to DataSource peers that request the object.



Message exchange between client application, Liberator, and an active DataSource (Integration Adapter)

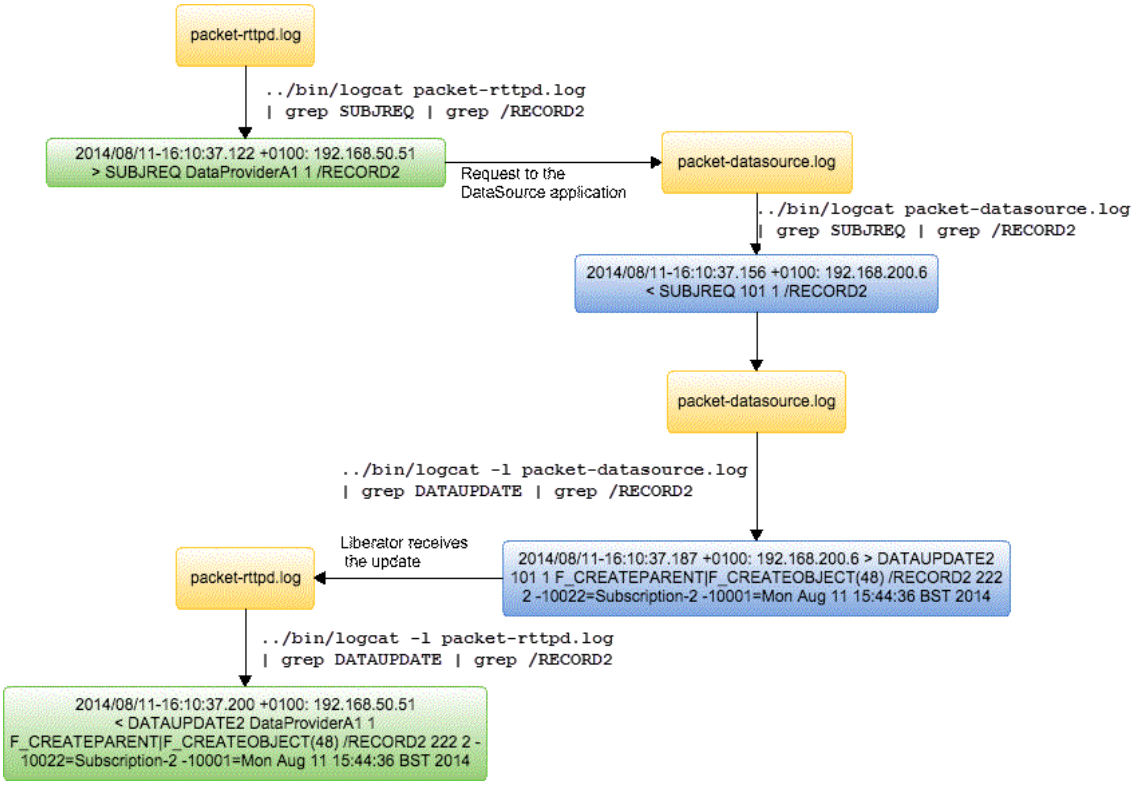
In the example shown above, the client application requests the object `/RECORD2`. Because the DataSource application that supplies this object is an active DataSource, Liberator must request the object from the DataSource application before it can send it on to the client.

Several log files record the messages associated with this object request:

- **Liberator side:** *event-rttpd.log*, *request-rttpd.log*, *object-rttpd.log*, *packet-rttpd.log*

- **DataSource side:** *datasrc.log*, *packet-datasrc.log*

Packet logs are the best logs to review if you want to trace the object request, as they contain all the messages you need to look at (except the session ID):



Using logcat and grep to trace an object request in the packet logs

The filtered output of the log files above confirm that the requested object was received by Liberator:

1. Liberator requests the object `/RECORD2` from the active DataSource (`SUBJREQ` message in *packet-rttpd.log*).
2. The DataSource receives the request and sends the requested object to Liberator (`SUBJREQ` and `DATAUPDATE` messages in *packet-datasrc.log*).
3. Liberator receives the requested object `/RECORD2` from the active DataSource (`DATAUPDATE` message in *packet-rttpd.log*).

Event, object, and request logs

If the event log is configured to record `DEBUG` messages, the request for the object is recorded in a `REQUEST` message:

Typical filtered output from the event log

```
2014/08/11-16:10:37.200 +0100: DEBUG: 0hGoeyrnCY4knBnFPJjSaB REQUEST /RECORD2
                                      (/RECORD2) [0x7f2b3d2f0bf0]
```

Object and request logs also record object requests in `REQUEST` messages:

Typical filtered output from the object log

```
2014/08/11-16:10:37.200 +0100: 0hGoeyrnCY4knBnFPJjSaB REQUEST /RECORD2 (/RECORD2)
```

Typical filtered output from the request log

```
2014/08/11-16:10:37.122 +0100: 192.168.50.57 admin 0hGoeyrnCY4knBnFPJjSaB
                                          "0hGoeyrnCY4knBnFPJjSaB REQUEST /RECORD2"
```

Note that the event, object, and request logs record the session ID of the client (`0hGoeyrnCY4knBnFPJjSaB`). The request log also records the username (`admin`). Packet logs do not record the session ID.

StreamLink Java and server-side RTTP logs

The StreamLink Java and RTTP server-side logs also record information about object requests from the client application.

Typical StreamLink Java log

```
2014/08/11-15:44:25.785 +0100 - INFO   : StreamLink subscribe /RECORD2 called.
2014/08/11-16:10:37.023 +0100 - FINER  : > 0hGoeyrnCY4knBnFPJjSaB+REQUEST+/RECORD2
2014/08/11-16:10:37.155 +0100 - FINER  : < 380002 /RECORD2
2014/08/11-16:10:37.189 +0100 - FINER  : < 6c020002 B=211 C=DataProviderAPricingSvc1
2014/08/11-16:10:37.191 +0100 - FINER  : < 6c040002 1=Subscription-2
                                                   7=Mon+Aug+11+15:44:36+BST+2014
```

In this example the text `0hGoeyrnCY4knBnFPJjSaB` is the session ID, and `380002` and `6c020002/6c040002` are codes that identify the message type to Caplin Support.

`38-0002` consists of:

- `38` – RTTP code that corresponds to a 'Blank Response' message, and means that the object is being requested from an active DataSource application
- `0002` – the object id for `/RECORD2`

`6c-02-0002` consists of:

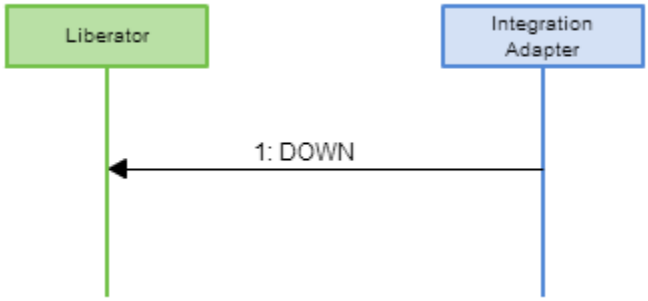
- `6c` – RTTP code that corresponds to a 'Record type 1 update'
- `02` – sequence number
- `0002` – the object id for `/RECORD2`

Typical server-side RTTP log

```
0hGoeyrnCY4knBnFPJjSaB REQUEST /RECORD2
data: 6c020002 B=211 C=DataProviderAPricingSvc1
data: 6c040002 1=Subscription-2 7=Mon+Aug+11+15:44:36+BST+2014
```

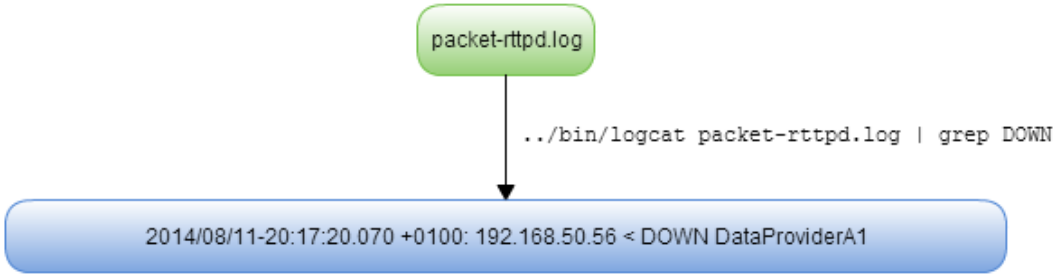
Stopping the Integration Adapter

If you stop an Integration Adapter (DataSource application), it sends a `DOWN` message to Liberator that is recorded in the Liberator packet log (*packet-rttpd.log*).



Integration Adapter and Liberator message exchange

The Liberator packet log is a binary file and you must use the **logcat** utility to view the content of the log file. To filter the packet log for `DOWN` messages, pipe the output of **logcat** to **grep**.

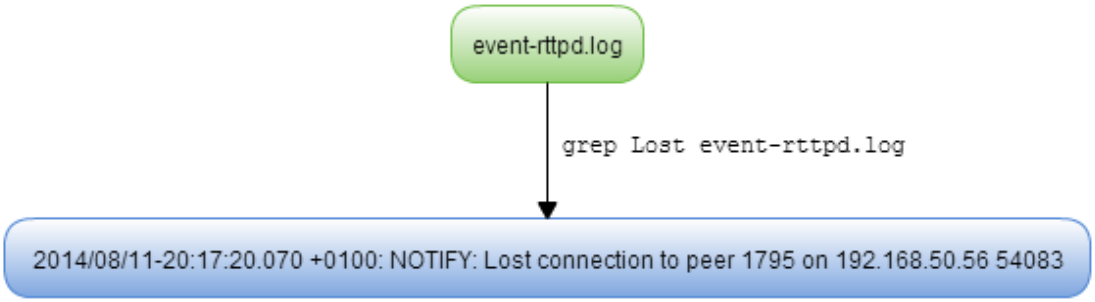


`The DOWN` message records the peer ID of the DataSource, and the date and time that the DataSource stopped:

Filtered DOWN message

Date-Time+Zone	IP Address	In	Msg-Type	Peer-Label
`2014/08/11-20:17:20.070 +0100`	`192.168.50.56`	`<`	`DOWN`	`DataProviderA1`

A `NOTIFY` message is recorded in the Liberator event log when you stop an Integration Adapter. The event log is a text file, and you can use **grep** to filter `NOTIFY` messages.



The StreamLink JS and server-side RTTP logs also record information when a DataSource stops.

Typical StreamLink JS log

```
2014/08/11-20:17:20.970 +0100 - FINER  : < 84 DataProviderAPricingSvc1
                                             DataProviderAPricingSvc1+IS+DOWN
2014/08/11-20:17:20.970 +0100 - FINER  : < 80 1795
                                             DataProviderA1 DataProviderA1+IS+DOWN
```

Typical server-side RTTP log

```
data: 84 DataProviderAPricingSvc1 DataProviderAPricingSvc1+IS+DOWN
data: 80 1795 DataProviderA1 DataProviderA1+IS+DOWN
```

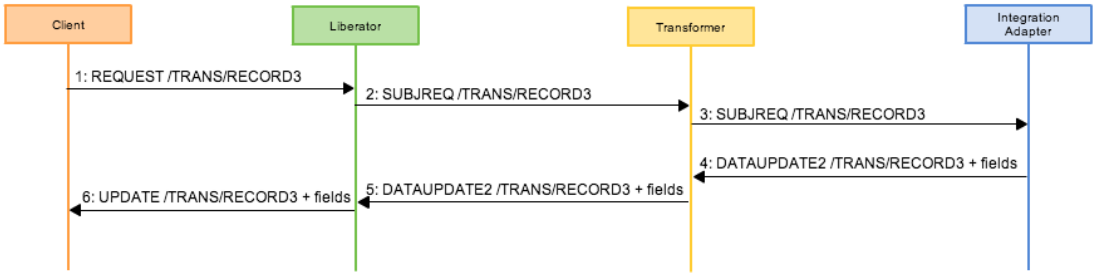
8.2 Example 2: Liberator – Transformer – Integration Adapter

In this example investigation, a Transformer is inserted between Liberator and the Integration Adapter in the Caplin Platform system.

The messages that are recorded when the end-user logs in, and when the Integration Adapter starts and stops, are similar to the messages described in the previous example (Example 1: Liberator – Integration Adapter 26) and will not be described again, but the object request messages are different.

Object requests

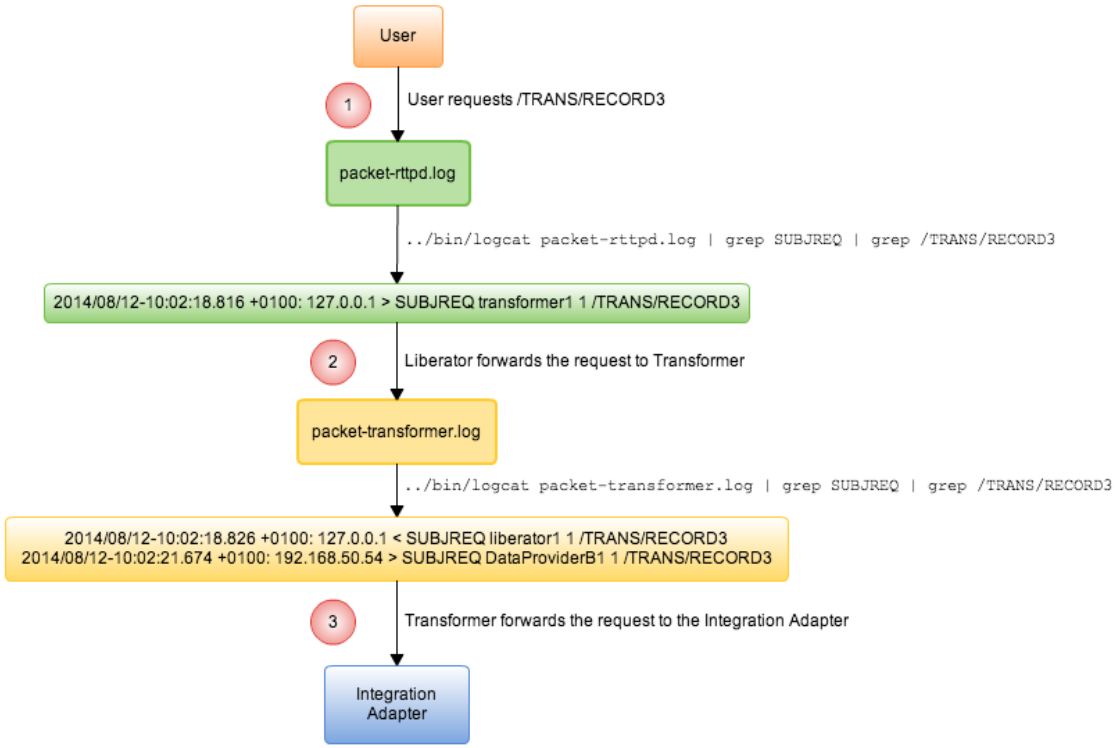
In this Caplin Platform system, Transformer receives object requests from Liberator and forwards them on to the Integration Adapter (DataSource application), and also receives objects and object updates from the Integration Adapter and forwards them on to Liberator.



Message exchange sequence

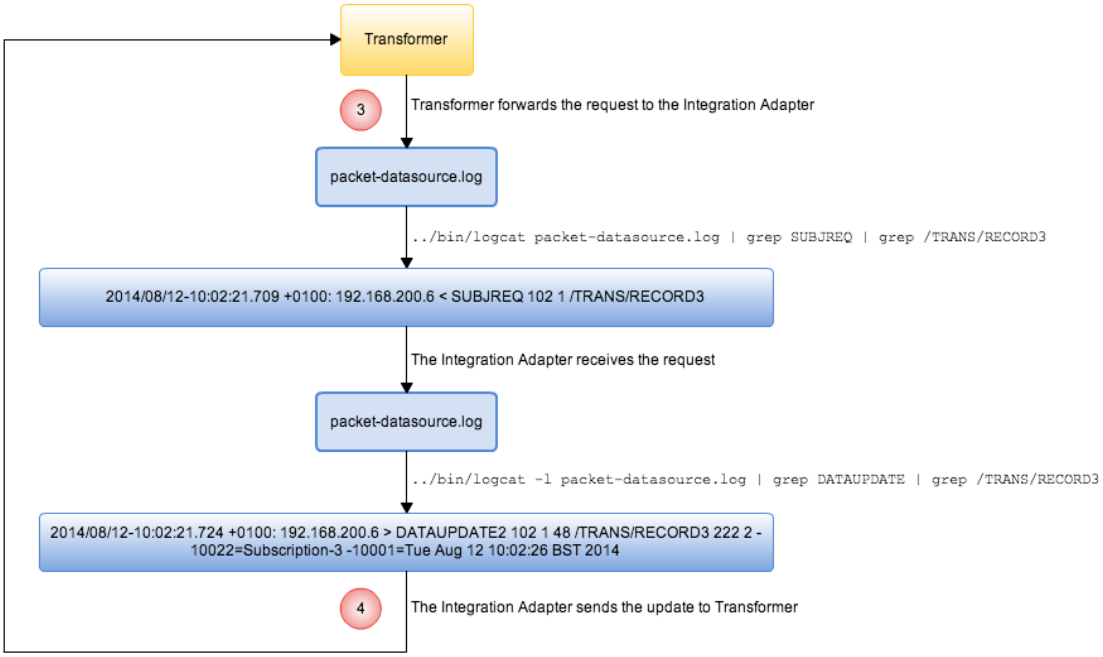
In this case the client requests the object `/D/2`, and the Integration Adapter that supplies this object is an active DataSource.

The packet logs contain all the messages you need to look at in order to trace the object request. The following diagram shows the logs you can look at to verify steps 1 to 3 of the object request: the request from the client application to the Integration Adapter. The circled numbers in this diagram correspond to the process steps in the message exchange sequence 37 diagram.



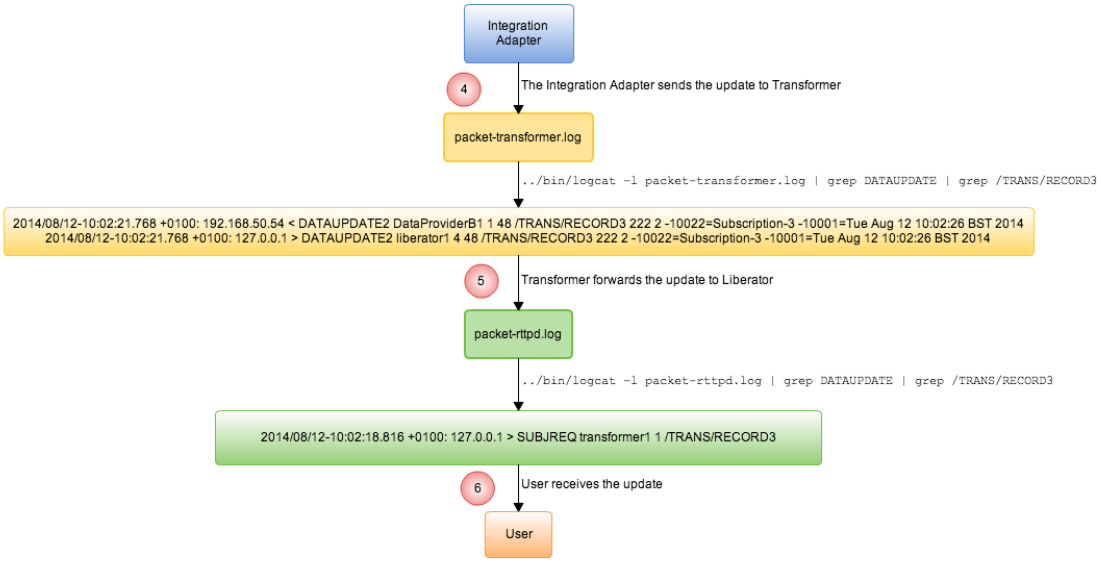
Verifying steps 1 to 3 of the object request

The next diagram shows the logs you can look at to verify steps 3 and 4 of the object request: the request by Transformer and the response from the Integration Adapter.



Verifying steps 3 and 4 of the object request

The final diagram in this example shows the logs you can look at to verify steps 4 and 5 of the object request: the receipt of the requested object at Transformer and Liberator.



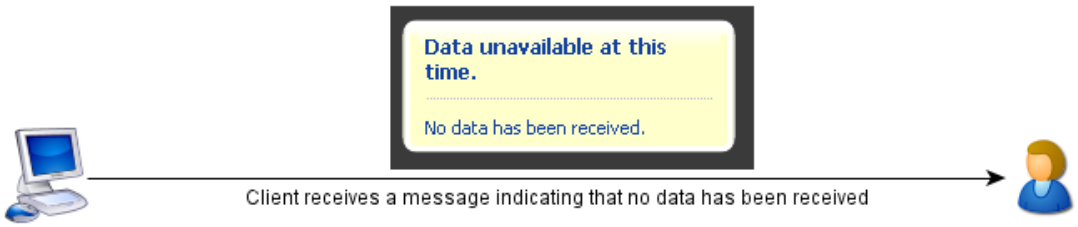
Verifying steps 4 and 5 of the object request

To verify step 6, the receipt of the requested object at the client, you need to review the StreamLink JS or server-side RTTP logs (as shown in Object requests 30 of Example 1).

8.3 Example 3: A 'Data Unavailable' message is displayed

This example investigation simulates the following situation:

- An end-user logs in to a Caplin Trader client application.
- The client application requests the container object `/E/CONTAINER`.
- A Data unavailable error message is displayed on the screen of the client application.



The displayed error message

Following the structured approach described in The investigation process 21, you can gather information that will help you to narrow the search area.

- **Identifying the problem**

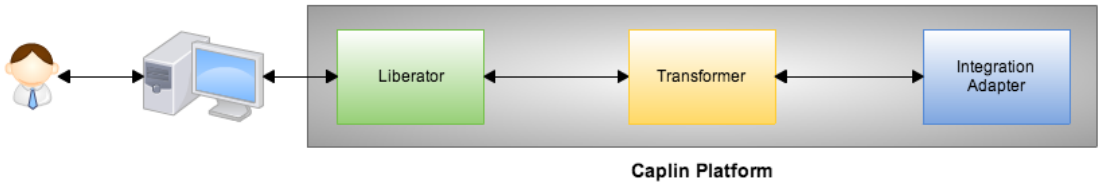
  The end-user `user1@caplin.com` successfully logged in.

  An error message is displayed when the container object `/E/CONTAINER` is requested.

  The container has five records of the form `/E/RECORDx`, where `x` is an integer (0-4).

- **Identifying the log files you need to review**

  In this case the end-user logged in to Caplin Trader, and the Caplin Platform system consists of a Liberator, Transformer, and an Integration Adapter that is an active DataSource.



  For this system, the log files to review are the Liberator logs, the Transformer logs, and the Integration Adapter logs.

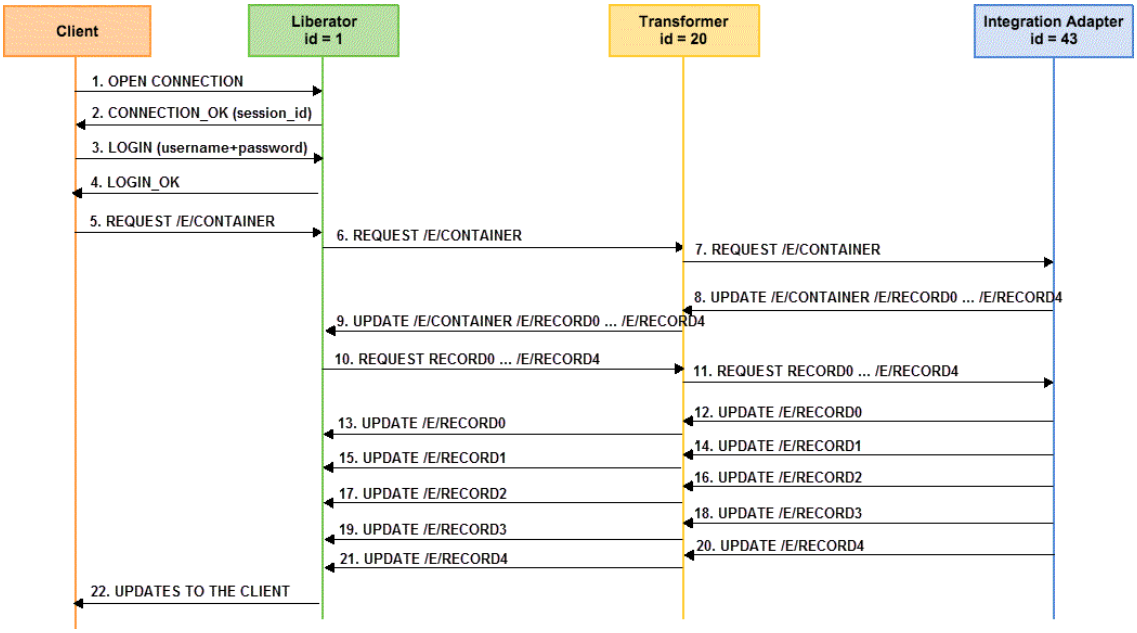
- **Identifying the data you need to look at in the log files**

  The username of the logged in end-user is `user1@caplin.com`.

  The requested container object is `/E/CONTAINER`.

  The container `/E/CONTAINER` has five records: `/E/RECORD0` ... `/E/RECORD4`.

The following diagram shows the expected sequence of messages when a container like this is requested. When you review the log files, you can compare the expected messages with the actual messages that were logged.

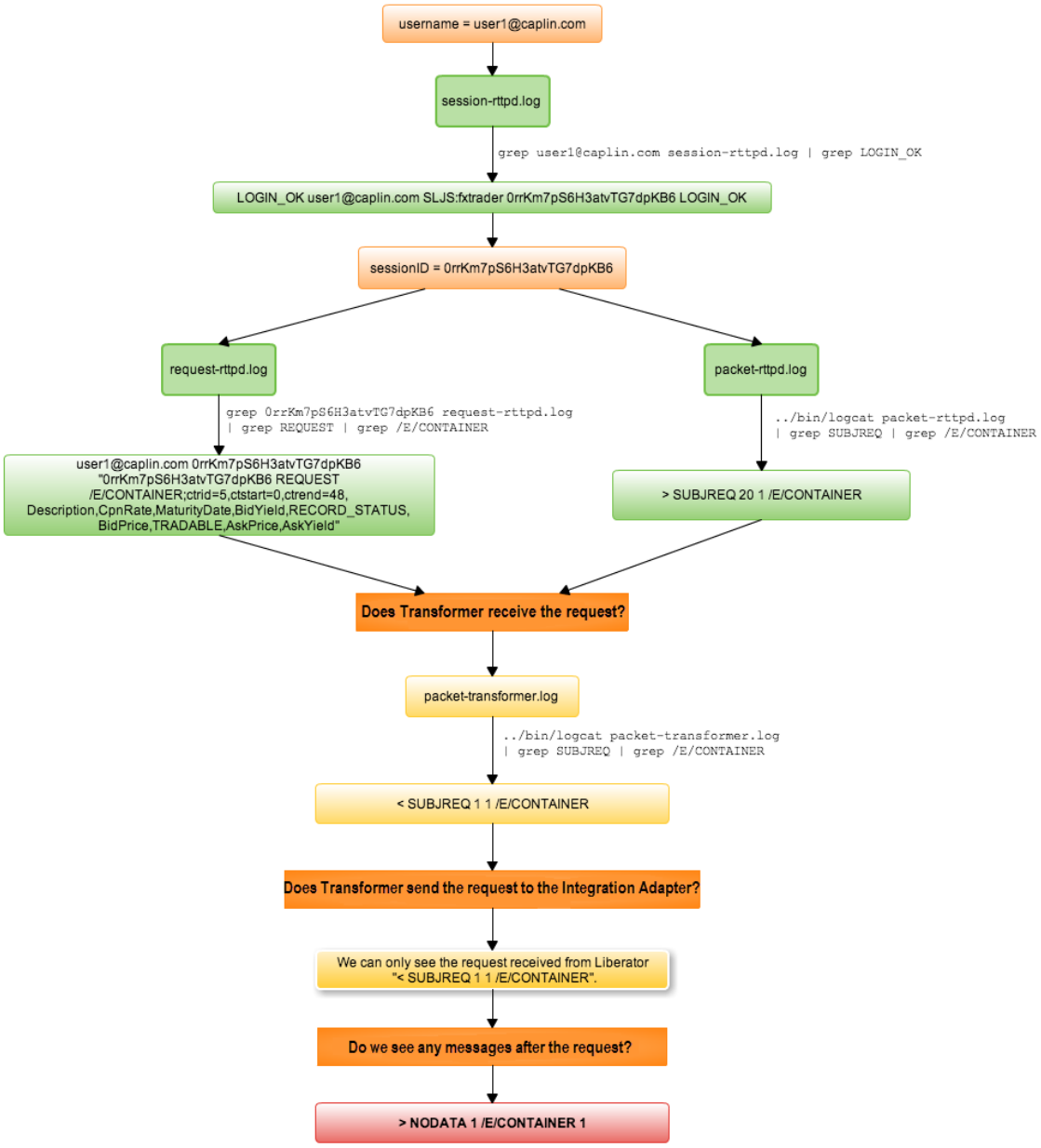


Expected sequence of messages

Several log files record messages associated with this container object request:

- **Liberator side:** *event-rttpd.log*, *session-rttpd.log*, *request-rttpd.log*, *packet-rttpd.log*
- **Transformer side:** *transformer.log*, *packet-transformer.log*
- **DataSource (Integration Adapter) side:** *datasrc.log*, *packet-datasrc.log*

A review of the log files reveals the actual messages recorded (timestamps have been removed for clarity):



The *packet-transformer.log* shows that Transformer received the request for the container `/E/CONTAINER`, as recorded in the `SUBJREQ` message, but returned a `NODATA` message to Liberator in response to this request. A `NODATA` message means that the requested object does not exist (see the **Caplin Liberator Administration Guide**). The reason for this could be:

- The Integration Adapter is down. Look at the Transformer packet log for `PEERINFO` messages that show the DataSource started. If the DataSource is down, try starting it.
- Incorrect configuration. Review the Transformer and Integration Adapter configuration files for errors in the configuration, and correct as necessary.

If the cause of the error is still not found, you can review activity in the component event logs. Remember that the event log only records detailed information if the logging level is set for maximum logging (see Development environment: enabling maximum logging 11).

In a production environment you may find issues that are more complicated than the situation described in this simple example, but the investigation process is the same. The more you know about your system, the easier it is to diagnose any problems that arise.

9 If you need to contact Caplin Support

If you have a problem with a Caplin Platform system that you cannot resolve, and you have a support contract with Caplin Systems, you can contact Caplin Support for assistance.

Before you do, gather as much information about the problem as possible. In particular:

- How does the problem occur?
- Can you replicate it?
- What was the time of the incident?
- Have any system changes been made recently?
- Take copies of all components logs.
- If any errors are displayed, take screenshots.

You can contact Caplin Support in the following ways. Please supply the log files that you captured, together with any other information that will help Caplin Support to quickly diagnose and correct the problem.

Log the problem in Caplin's Issue Management System (Jira):	https://jira.caplin.com
Email Caplin Support:	support@caplin.com
Telephone Caplin Support:	+44 (0) 20 7826 9601

10 Appendix A: Packet logs and logcat

Most Caplin Platform component logs are simple text files that can be viewed using a suitable text display utility or text editor, such as the Linux commands **cat**, **more**, and **vim**.

Packet logs have a binary format and must be viewed using Caplin's **logcat** utility. **logcat** is used in the same way as the standard Linux **cat** command, and is located in the *bin* directory of the Liberator and Transformer installation.

This appendix provides four tips that may help you when you review or transfer packet logs. If you need more information about the **logcat** utility, please refer to the **Liberator Administration Guide**, which contains a detailed description of **logcat** options (Section **Viewing log files: the Logcat utility**). You can also run `logcat --help` in the command line to view the available **logcat** options.

10.1 Displaying fields and flags by name

When you use **logcat** to display messages that contain fields or flags, **logcat** normally displays the fields and flags by number. For example, the following command pipes the output of **logcat** to **grep**, which filters the packet log for `DATAUPDATE` messages.

Filter packet log for DATAUPDATE messages

```
../bin/logcat packet-rttpd.log | grep DATAUPDATE
```

In this example the `DATAUPDATE` message has one flag (`4144`) and two fields (field numbers `4` and `5`).

Example DATAUPDATE message showing fields and flags by number

Date-Time +Zone	IP Origin	In (<)	Msg-Type	Peer-ID	SeqNum	Flag
2014/08/21-12:47:20.712 +0000	127.0.0.1	<	DATAUPDATE2	10	1	4144

Subject	Type	Num of Fields	Field number =Value	Field number =Value
/F/0	222	2	4=64.91	5=65.14

To display fields and flags by name rather than by number, add the following arguments to the **logcat** command.

-F Display fields by name.

-f The file that maps field names to numbers. The default value is *fields.conf* in the current directory.

-l Display flags by name.

The following example uses these arguments to filter the same packet log for `DATAUPDATE` messages.

Filter packet log with arguments

```
../bin/logcat -F -f ../../CommonConfig/fields-caplintrader.conf -l packet-rttpd.log
| grep DATAUPDATE
```

Fields and flags are now displayed by name and not by number.

Example DATAUPDATE message showing fields and flags by name

Date-Time +Zone	IP Origin	In (<)	Msg-Type	Peer-ID	SeqNum	Flag
2014/08/21-12:47:20.712 +0000	127.0.0.1	<	DATAUPDATE2	10	1	F_IMAGE| F_CREATEPARENT| F_CREATEOBJECT (4144)

Subject	Type	Num of Fields	Field name =Value	Field name =Value
/F/0	222	2	BestBid=64.91	BestAsk=65.14

In this example the flag indicates that the packet is an image and not an update. For a description of each flag that a DataSource can send, see the DataSource API reference documentation.

10.2 Redirecting logcat output to a text file

The output of Caplin's **logcat** utility is normally displayed on the screen, but you can redirect the output to a text file that can be viewed later using a text editor. The following example runs **logcat** on the packet log *packet-rttpd.log*, and redirects the output to the text file *packet-rttpd.txt*.

Example command

```
logcat packet-rttpd.log > packet-rttpd.txt
```

Message displayed on the screen

```
Logcat: Log Type 'packet' Version 4 created by 'rttpd' in timezone 'Europe/London
```

10.3 Inspecting real time packet logs

The **tail** command can be used with **logcat** to display the last part of a packet log on the screen. The following example displays the last part of the packet log *packet-rttpd.log* in real time (as the log is being created).

Example command

```
tail -f packet-rttpd.log | ../bin/logcat
```

The **tail** argument `-f` specifies that data is appended to the output as the log file grows in size (as more data is logged).

Typical information displayed on the screen

```
2014/08/21-09:21:17.375 +0100: 127.0.0.1 < DATAUPDATE2 1 632 48
/Examples/Records/Stocks/JAVAD 222 4 10441=21.0321 10005=09:21:17 13010=1302769277374
10436=20.9921
2014/08/21-09:21:17.474 +0100: 127.0.0.1 < DATAUPDATE2 1 633
8240 /Examples/Broadcast/Stocks/GOOG 222 6 10005=09:21:17
10056=-0.3753 10011=0.5721 13010=1302769277474 10032=5219000 10006=723.9571
2014/08/21-09:21:17.494 +0100: 127.0.0.1 < DATAUPDATE2 1 634
8240 /Examples/Broadcast/FX/ZAR 222 4 10441=6.7488
10005=09:21:17 13010=1302769277494 10436=6.7388
```

10.4 Splitting packet logs

The Linux **split** command can be used to split a packet log into multiple files of a smaller size. This can be useful if you want to review a large packet log, or if you need to transfer a packet log to Caplin's Issue Management System (Jira), which limits the size of file you can transfer. The following example splits the text version of the packet log *packet-rttp.txt* (see Redirecting the logcat output to a text file 47), but you can also split the binary version of a packet log.

Example command

```
split -b 10m packet-rttp.txt packet_
```

This example splits the packet log *packet-rttp.txt* into separate files, each 10Mb in size and named *packet_aa.txt*, *packet_bb.txt* ... (and so on).

Note: The **split** command can produce a lot of files if you are not careful with the size parameter.

11 Glossary of terms and acronyms

This section contains a glossary of terms, abbreviations, and acronyms relating to the management and interpretation of Caplin Platform component log files.

Term	Definition
Active DataSource	A **DataSource application** that accepts subscriptions requests for an object, and only sends an object or update to the object to the **DataSource peers** that requested it.
API	Application Programming Interface
App	An application that runs in a web browser or on a mobile device.
Broadcast DataSource	A **DataSource application** that does not wait for its **DataSource peers** to request an object, but sends the objects it has to all peers that connect to it. When the broadcast DataSource receives an update to an object, it sends the update to all connected peers.
Caplin Director	A complete user administration system for the **Caplin Platform**.
Caplin Integration Suite (CIS)	A set of APIs and tools for creating adapters that integrate the **Caplin Platform** with external systems. Also see **Integration Adapter**.
Caplin Integration Suite for Java (CIS for Java)	The Java edition of the **Caplin Integration Suite**.
Caplin KeyMaster	A component that integrates the **Caplin Platform** with any web-based authentication system. It generates a secure encrypted token that enables **Caplin Liberator** to identify authenticated users, and is generally used in conjunction with a single sign-on system.
Caplin Management Console (CMC)	A Java application that communicates with the **Caplin Platform** and **Integration Adapters** via **JMX**, and provides a GUI for monitoring and controlling these components.
Caplin Liberator	A financial internet hub that delivers data and messages in real time to and from subscribers over any network.
Caplin Platform	An integrated suite of software that supports the services and distribution capabilities needed for web trading. It consists of **Caplin Liberator**, **Caplin Transformer**, **Caplin KeyMaster**, **Caplin Director**, and **Caplin Management Console**.
Caplin Platform System	A single-dealer platform that is built using the **Caplin Platform**.
Caplin Trader	A complete development suite for creating HTML5 trading **apps**. It includes BladeRunner, Verifier, the Caplin HTML5 Libraries, and a selection of blades, Bladesets and Motifs as required.
Caplin Trader application	A **client application** that has been built using **Caplin Trader**.
Caplin Transformer	An event-driven, real-time data transformation engine optimised for web trading services. These services are implemented in **Transformer Modules**.
cat	A Linux utility that can be used to display text from a file.
C DataSource	An implementation of the **DataSource API**, and underlying code library, for writing **DataSource applications** in the C programming language.
Client	In this document, client is short for **client application**.

Term	Definition
Client application	In the context of the **Caplin Platform**, a client application is any application that uses the **StreamLink API** to communicate with **Caplin Liberator**.
Container	A data type supported by the **Caplin Platform** that represents a list of items.
DataSource	DataSource is the messaging infrastructure used by the **Caplin Platform** and **Integration Adapters**. In some older documents DataSource is also used as a synonym (but non-preferred term) for **DataSource application**.
DataSource API	An API that allows server applications (including **Integration Adapters**) to communicate with the **Caplin Platform**.
DataSource application	An application that uses the DataSource API. **Caplin Liberator**, **Caplin Transformer**, and **Integration Adapters** are all DataSource applications.
DataSource.NET	The implementation of the **DataSource API** and underlying code library for writing **DataSource applications** in the .NET programming language.
DataSource peer	A **DataSource application** that another DataSource application is configured to communicate with.
end-user	A person who logs in to, and interacts with, a **client application**.
grep	A Linux utility that can be used to display lines in a text file that match a pattern.
Integration Adapter	A server application that allows an external system to communicate with the **Caplin Platform**. An Integration Adapter is a **DataSource application** and is created using the **Caplin Integration Suite**.
Java DataSource	The implementation (in the **Caplin Integration Suite**) of the **DataSource API**, and underlying code library, for writing **DataSource applications** in the Java programming language.
Liberator	In this document, Liberator is short for **Caplin Liberator**.
log	In this document, log is short for **log file**. To log something means to record it in a log file.
logcat	A Caplin utility that is used to display the content of a packet log.
log file	Log files record error notifications, and the events that occur when a **Caplin Platform system** is up and running (such as components starting and stopping). Log files also record conversations between **DataSource peers**, and between the **client application** and **Liberator**.
log level	The log level determines the severity of errors and events that are recorded in a **log file**. For some log files there is only one log level that cannot be changed, while for other log files the log level can be configured to be one of several permitted log levels. The log level is sometimes referred to as the logging level.
more	A Linux utility that can be used to view a text file, one screen page at a time.

Term	Definition
RTTP	Real Time Text Protocol Caplin's protocol for streaming real-time financial data from **Caplin Liberator** servers to **client applications**, and for transmitting trade messages and other messages between clients and Liberator in both directions.
StreamLink	The library underlying the **StreamLink API**.
StreamLink API	An API that allows a **client application** to communicate with a **Caplin Liberator**. There are StreamLink APIs for various technologies; for example, Java, JavaScript, .NET and Silverlight applications, and Objective-C running on iOS.
StreamLink Java	The **StreamLink API** for Java.
StreamLink JS	The **StreamLink API** for JavaScript. In Caplin Platform 6 and Caplin Trader 3 it replaces **StreamLink for Browsers**.
StreamLink.NET	The **StreamLink API** for .NET.
SL4B	Abbreviation for **S**tream**L**ink 4 (**for**) **B**rowsers.
Transformer	In this document, Transformer is short for **Caplin Transformer**.
Transformer Module	A software module in **Caplin Transformer** that implements a service. For example, the Refiner module provides a **Container** filtering and sorting service.
user	In this document, user is short for **end-user**.
vim	A Linux utility that can be used to edit a text file.

CAPLIN
THE WEB TRADING TECHNOLOGY COMPANY

Contact Us

Caplin Systems Ltd

Cutlers Court

115 Houndsditch

London EC3A 7BR

Telephone: +44 20 7826 9600

Caplin Systems, Inc.

7 World Trade Center

46th Floor

New York, NY 10007

Telephone: +1 (212) 266 0198

Caplin Systems (Singapore) Pte Ltd.

Level 39, MBFC Tower 2

10 Marina Boulevard

Singapore 018983

Telephone: +65 6818 6290

www.caplin.com