CAPLIN

Caplin Platform 6.0

How To Create C And Lua Blades

December 2012

Contents

1 Preface

1.1 What this document contains

This document describes how to create new 'C' and **Lua**-based **Caplin Platform blades**. It also explains how to deploy a finished blade to the Caplin Platform Deployment Framework.

If you wish to create a Java™-based blade, see the document
Caplin Integration Suite: How To Create A Platform Java Blade.

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 who want to develop C and Lua **blades** for a web trading platform based on the **Caplin Platform Deployment Framework**.

Before reading this document, you should be familiar with the concepts and terms that are introduced in the following documents.

- **Caplin Platform Overview** (all sections)
- **Caplin Platform: Deployment Framework Overview** (all sections)
- **Caplin Liberator Administration Guide** (Overview section)

1.3 Related documents

- **Caplin Platform: Overview**

  A business and technical overview of the Caplin Platform
- **Caplin Platform: Deployment Framework Overview**

  Gives an overview of the Caplin Platform Deployment Framework, and explains the concept of Caplin Platform blades.
- **Caplin Platform: How To Use The Deployment Framework**

  Explains how to install and use the Caplin Platform Deployment Framework. It also describes how Caplin Platform blades are deployed to the Framework to create a working Caplin Platform system.

- **How To Create A Platform Java blade**

  Explains in detail how to create new Java-based Caplin Platform Blades using the Caplin Integration Suite Toolkit.

- **Caplin Liberator Administration Guide**

  Describes the Caplin Liberator server. Includes configuration reference information and a list of Liberator log and debug messages.

- **DataSource For C Configuration Syntax Reference**

  Describes the syntax of the language that is used to configure **DataSource applications** that have been built using Caplin's C DataSource API. Such applications include Caplin Liberator, Caplin Transformer, and Integration Adapters that use this API.

- **Caplin DataSource for C API Documentation**

  The reference documentation for the C DataSource API.

- **Best Practices For Deploying the Caplin Platform**

  Provides recommendations for deploying Caplin Platform in a typical live environment, and discusses failover scenarios for achieving high service availability.

1.4 Typographical conventions

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

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

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

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

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

1.5 Feedback

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

Please email your feedback to documentation@caplin.com.

1.6 Acknowledgments

Adobe Reader is a registered trademark of Adobe Systems Incorporated in the United States and/or other countries.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Java is a trademark or registered trademark of Oracle® Corporation in the U.S. and other countries.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Lua is free software from the Pontifical Catholic University of Rio de Janeiro.
Lua 5.0 license Copyright © 1994-2007 Lua.org, PUC-Rio.

2 Getting started

The concept of Caplin Platform blades and the framework used to deploy them are both described in the **Caplin Platform Deployment Framework Overview**. In summary, a Caplin Platform blade is a re-usable software module containing the code and resources needed to implement a working feature of a trading system that uses the **Caplin Platform**.

Each blade is a self contained set of files. There are three types of blade:

- **Config blade**

  This type of blade consists solely of configuration for the Platform's **core components** (**Liberator** and/or **Transformer**).

- **Adapter blade**

  This type of blade connects to, and supplies data to, a Liberator or Transformer. It consists of an **Integration Adapter** (an executable binary file), **DataSource** configuration, and core component configuration. Integration Adapters can be written in Java or C.

- **Service blade**

  This type of blade contains a module written in C, Java, or Lua that is to be loaded into one of the core components; for example, a permissioning auth (authorization) module that is loaded into Liberator.

This document explains how to develop:

- **Adapter blades** written in C using the **C DataSource API**

  (Called **C-based Adapter blades** in the rest of this document)

- **Service blades** for Caplin Transformer, written in Lua

  (Called **Lua module blades** in the rest of this document)

In the rest of this document Caplin Platform blades are just called blades.

Tip: If you wish to develop Adapter blades in Java, see the document **Caplin Integration Suite: How To Create A Platform Java Blade**.

2.1 Blade naming conventions

When developing a new blade, you should follow some naming conventions that will make it easier to maintain the blade in future:

- The blade name should reflect the feature that the blade implements.

  For example, **TradeFXBlade**.

- The name of the root directory where the blade files are located must be the blade name.

  For example, the blade called **TradeFXBlade**, must be located in the directory ***TradeFXBlade***

  For another example of this, see “Creating the Adapter blade’s directory structure” in section 3.2 “Creating the C-based Adapter blade”.

- For C-based Adapter blades, in DataSource configuration that refers to the Adapter, the **label** and **remote-name** options of the **add-peer** configuration item should contain the name of the blade.

  For an example of this, see “Liberator configuration” and “Transformer configuration” in section 3.2 “Creating the C-based Adapter blade”.

- If the blade provides a **data service** to Liberator, the **service-name** option in the **add-data-service** configuration item of the Liberator configuration should contain the name of the blade.

  For an example of this, see “Liberator configuration” in section 3.2 “Creating the C-based Adapter blade”.

3 Creating and developing a C-based Adapter blade

This section explains how to create and develop an Adapter blade where the Integration Adapter part is written in C.

> **Tip:** If you wish to develop an Adapter blade in Java, see the document **Caplin Integration Suite: How To Create A Platform Java Blade**.

3.1 C-based Adapter blade structure

A C-based Adapter blade contains:

- The blade's field definitions in *<blade_name>/blade_config/fields.conf*
- Liberator related configuration in *<blade_name>/Liberator/etc/rttpd.conf*
- Transformer related configuration in *<blade_name>/Transformer/etc/transformer.conf*
- Configuration, binary files, and startup scripts for the Integration Adapter itself are in *<blade_name>/DataSource/*

3.2 Creating the C-based Adapter blade

The following sections describe how to create a C-based Adapter blade. The best way to describe the development is to describe an example Integration Adapter that is implemented in C and is called **NewAdapterBlade**. This Adapter connects to the Liberator and Transformer and provides `/ADAPTEREXAMPLELIB` and `/ADAPTEREXAMPLETRANS` data containing specific fields.

Deploying the Caplin Platform Deployment Framework

Since the blade is part of the Caplin Platform Deployment Framework it uses configuration from that kit to define Liberator and Transformer peer connections. This configuration must be available during development. A convenient way to achieve this is to deploy a Caplin Platform Deployment Framework on the machine where you wish to develop the Adapter blade

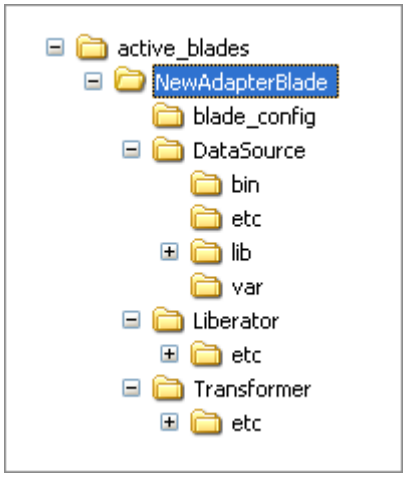
- To deploy the Framework, follow the instructions in the document **Caplin Platform: How To Use The Deployment Framework**.

- You will need a Caplin Transformer and/or Caplin Liberator to help test your Adapter blade whilst you develop it. If you do not have Transformer/Liberator already deployed on a machine for this purpose, you may wish to deploy one or both of these components on your development machine, to the Framework you have just deployed. To do this, follow the instructions in the above document.

Creating the Adapter blade's directory structure

- Create a directory with the name of the blade (for example, *NewAdapterBlade*) and its subdirectories.

  The directory tree must be in the *active_blades* directory of the Caplin Platform Deployment Framework on your development machine. It must have the following structure:



Note: Initially the *var* directory contains no files, but it must exist and it must become part of the blade kit. This directory is used by the Integration Adapter at run time.

Creating the *fields* file

- Create a ***fields.conf*** file listing all the fields provided by the Integration Adapter and put it in the *blade_config* directory of the blade.

Example of *fields.conf* entries for a C-based Adapter blade:

```
add-field ExampleAdapterField1  -99000 0
add-field ExampleAdapterField2  -99001 0
```

Writing the core component configuration

Your new Adapter blade must include configuration for the core components that it interacts with. This configuration should typically consist of:

- Data services for the data provided by the blade.
- An **add-peer** configuration entry if the Integration Adapter connects to the core component.
- Failover configuration if the Integration Adapter can be located in a failover leg.

Which core components require configuration depends on how your Integration Adapter is to interact with the Caplin Platform. The most typical situations are:

- The Integration Adapter connects directly to a Liberator and does not send data to or require any services from a Transformer.

  In this case, you only need to define Liberator configuration for the blade – for connections to the Adapter, and usually for one or more data services.
- The Integration Adapter connects to a Transformer and sends data to it; the Transformer forwards the (usually modified) data on to a Liberator.

  In this case, you need to define both Transformer and Liberator configuration for the blade. The Transformer configuration must define connections to the Adapter, and usually one or more data services. The Liberator configuration typically defines one or more data services (that are served by the Transformer), but no Adapter connections.

Background information

- For more information about data services, see the **Caplin Platform Overview** and the **Caplin Liberator Administration Guide**.
- For more information about failover legs, see the document **Caplin Platform: How To Use The Deployment Framework**.

Liberator configuration

- Create a Liberator configuration file called ***rttpd.conf*** in *NewAdapterBlade/Liberator/etc/*
- The contents of this file are typically:

Integration Adapter configuration for Liberator (*rttpd.conf*)

```
#
# Peer connection to primary instance
# of the example Integration Adapter,
# with heartbeats configured.
#
add-peer
        remote-id                ${THIS_LEG}9999
        remote-type              active
        remote-name              NewAdapterBlade${THIS_LEG}
        label                    NewAdapterBlade${THIS_LEG}
        heartbeat-time           15
        heartbeat-slack-time     5
end-peer

#
# Peer connections to the secondary instance
# of the example Integration Adapter.
# Only configured if failover is enabled –
# see global_config/environment.conf
# in the Caplin Platform Deployment Framework.
#
if "${FAILOVER}" == "ENABLED"
        add-peer
                remote-id                ${OTHER_LEG}9999
                remote-type              active
                remote-name              NewAdapterBlade${OTHER_LEG}
                label                    NewAdapterBlade${OTHER_LEG}
                heartbeat-time           15
                heartbeat-slack-time     5
        end-peer
endif
```

```
#
# Data service for the example Integration Adapter.
# The Integration Adapter provides data in
# the namespace /ADAPTEREXAMPLELIB to the Liberator.
# If failover is enabled the secondary example Integration Adapter
# will provide the data when a failover occurs.
#
add-data-service
        service-name            NewAdapterBlade${THIS_LEG}
        include-pattern         ^/ADAPTEREXAMPLELIB

        add-source-group
                required
                add-priority
                        label   NewAdapterBlade${THIS_LEG}
                end-priority

                if "${FAILOVER}" == "ENABLED"
                        add-priority
                                label   NewAdapterBlade${OTHER_LEG}
                        end-priority
                endif

        end-source-group
end-data-service
```

This is fairly standard configuration. The blade-specific points to note are:

- The blade name is used in **label** configuration options, **service-name** options, and **remote-name** options (see section 2.1 “”).
- The macros `THIS_LEG` and `OTHER_LEG` are used to differentiate the configuration of primary and secondary failover legs.
- The `FAILOVER` macro conditionally enables configuration of a failover peer connection to Liberator and configuration of failover data services.
- The Integration Adapter’s connection to Liberator uses heartbeats with the recommended heartbeat configuration (configuration options **heartbeat-time** and **heartbeat-slack-time**).
- All the macros in the configuration above are defined in the global configuration of the Deployment Framework (*global_config/environment.conf*).

Transformer configuration

- Create a Transformer configuration file called ***transformer.conf*** in *NewAdapterBlade/Transformer/etc/*
- The contents of this file are typically:

Integration Adapter configuration for Transformer (*transformer.conf*)

```
#
# Peer connection to primary instance
# of the example Integration Adapter
# with heartbeats configured.
#
add-peer
        remote-id                ${THIS_LEG}9999
        local-type               active
        remote-name              NewAdapterBlade${THIS_LEG}
        label                    NewAdapterBlade${THIS_LEG}
        heartbeat-time           15
        heartbeat-slack-time     5
end-peer

#
# Peer connections to the secondary instance
# of the example Integration Adapter.
# Only configured if failover is enabled –
# see global_config/environment.conf.
# in the Caplin Platform Deployment Framework.
#
if "${FAILOVER}" == "ENABLED"
        add-peer
                remote-id                ${OTHER_LEG}9999
                local-type               active
                remote-name              NewAdapterBlade${OTHER_LEG}
                label                    NewAdapterBlade${OTHER_LEG}
                heartbeat-time           15
                heartbeat-slack-time     5
        end-peer
endif
```

```
#
# Data service for the example Integration Adapter.
# The Integration Adapter provides data in
# the namespace /ADAPTEREXAMPLETRANS to the Transformer.
# If failover is enabled the secondary example Integration Adapter
# will provide the data when a failover occurs.
#
add-data-service
        service-name            NewAdapterBlade${THIS_LEG}
        include-pattern         ^/ADAPTEREXAMPLETRANS

        add-source-group
                required
                add-priority
                        label   NewAdapterBlade${THIS_LEG}
                end-priority

                if "${FAILOVER}" == "ENABLED"
                        add-priority
                                label   NewAdapterBlade${OTHER_LEG}
                        end-priority
                endif

        end-source-group
end-data-service
```

This is fairly standard configuration. The same blade specific points apply as for the Liberator configuration.

Integrating the initial blade configuration with the Deployment Framework

When you have created the initial core component configuration for your Adapter blade, you can use this configuration to allow a Transformer and/or Liberator in your development environment to accept peer connections from the Integration Adapter you are developing.

- Copy the *NewAdapterBlade* directory and its subdirectories to each server machine where the Transformer and/or Liberator run. Put the directory tree in the *active_blades* directory of the server machine's Caplin Platform Deployment Framework.
- Stop each of the Deployment Frameworks and restart them.

  (See "Starting and stopping the Deployment Framework" in **Caplin Platform: How To Use The Deployment Framework**.)

  At this stage the Liberator and Transformer are running and will accept peer connections from the new Integration Adapter in your development environment if it attempts to connect into them.

Tip: At this stage you do not need to have populated any of the blade's *DataSource* directory tree.

Writing the DataSource configuration for the Integration Adapter

You must define some DataSource configuration for the Integration Adapter part of your new Adapter blade. This configuration only needs to be defined on the machine where the blade is being developed; it does not need to be copied to the other Deployment Frameworks where the Transformer and/or Liberator run.

- Create a configuration file called ***<adapter_binary_name>.conf*** in *NewAdapterBlade/DataSource/etc*, where *<adapter_binary_name>* is the name of the binary executable file for the Integration Adapter.
- The contents of this file are typically:

Integration Adapter's DataSource configuration (*<adapter_binary_name>.conf*)

```
# Include base configuration
include-file "${ENV:CONFIG_BASE}/bootstrap.conf"


include-file "${ENV:FIELDS_DIR}fields.conf"

#
# Event logging configuration
#
log-dir %r/var
log-level INFO

#
# DataSource identifier.
# Each DataSource application must have a unique identifier.
#
datasrc-id            ${THIS_LEG}9999

#
# Peer connection to primary Liberator with heartbeats configured.
#
add-peer
        local-type              active|contrib
        remote-name             liberator${THIS_LEG}
        addr                    ${LIBERATOR_HOST_${THIS_LEG}}
        port                    ${LIBERATOR_DATASRCPORT_${THIS_LEG}}
        heartbeat-time          15
        heartbeat-slack-time    5
end-peer
```

```
#
# Peer connections to secondary Liberator and Transformer. Only
configured if failover is enabled - see environment.conf.
#
if "${FAILOVER}" == "ENABLED"
        add-peer
                local-type              active|contrib
                remote-name             liberator${OTHER_LEG}
                addr                    ${LIBERATOR_HOST_${OTHER_LEG}}
                port             ${LIBERATOR_DATASRCPORT_${OTHER_LEG}}
                heartbeat-time          15
                heartbeat-slack-time    5
        end-peer

        add-peer
                local-type              active|contrib
                remote-name             transformer${OTHER_LEG}
                addr                    ${TRANSFORMER_HOST_${OTHER_LEG}}
                port             ${TRANSFORMER_DATASRCPORT_${OTHER_LEG}}
                heartbeat-time          15
                heartbeat-slack-time    5
        end-peer
endif

#
# Peer connection to primary Transformer with heartbeats configured.
#
add-peer
        local-type              active|contrib
        remote-name             transformer${THIS_LEG}
        addr                    ${TRANSFORMER_HOST_${THIS_LEG}}
        port                    ${TRANSFORMER_DATASRCPORT_${THIS_LEG}}
        heartbeat-time          15
        heartbeat-slack-time    5
end-peer
```

The blade-specific points to note here are:

- In this example, the Integration Adapter can connect to both a Liberator and a Transformer. If your Adapter only connects to one of these core components, omit the configuration for the other component.
- The DataSource Adapter must be given an ID number that is unique across all instances of DataSource applications (other Adapters, Liberators, and Transformers) deployed on all server machines in the Platform. The ID is defined by the configuration item **datasrc-id**.

Note: It is recommended that **datasrc_id**s for new blades start from `10000` for the primary instance of an Integration Adapter, and from `20000` for the secondary (failover) instance. This avoids any clash with the IDs allocated to core components. When adding a new Integration Adapter blade, review the IDs used for existing blades and select a new one for the new Integration Adapter.

The example configuration file shown here qualifies the ID by the failover leg (`THIS_LEG`) to which the Adapter instance belongs. This ensures that the primary ID is of the form `1nnnn`, and the secondary ID is of the form `2nnnn`.

- The Liberator and Transformer's hostnames, as used in the **addr** option of **add-peer**, and their DataSource ports, as used in the **port** option of **add-peer**, are defined by global configuration macros (`LIBERATOR_HOST`, `LIBERATOR_DATASRCPORT`, `TRANSFORMER_HOST`, and `TRANSFORMER_DATASRCPORT`)
- The configuration defines connections to a failover Liberator and a failover Transformer.
- The `THIS_LEG` and `OTHER_LEG` macros are used in the **remote_name**, **addr**, and **port** options of **add-peer**, to distinguish these option settings according to the failover leg to which they refer.
- The Integration Adapter's connections to Liberator and Transformer use heartbeats with the recommended heartbeat configuration (configuration options **heartbeat-time** and **heartbeat-slack-time**).

3.3 Developing the new C-based Integration Adapter

- Develop your C-based Integration Adapter following the usual pattern as described in the **Caplin DataSource for C API Documentation**.
- When the development is complete, copy the new files into the relevant blade directories:

File type:	Copy to directory in blade:
Binary files	*NewAdapterBlade/DataSource/**bin**/*
Library files	*NewAdapterBlade/DataSource/**lib**/*
etc/datasrc files in the DataSource for C SDK	*NewAdapterBlade/DataSource/**etc**/<**adapter_binary_name**>/* where *<adapter_binary_name>* is the name of the binary executable file for the Integration Adapter.
Any other configuration files required; for example, *java.conf*, *jmx.conf*	*NewAdapterBlade/DataSource/**etc**/*

3.4 Starting the new C-based Adapter blade

- To enable the newly written C-based Adapter blade to be started in your development environment, add the following line to the *global_config/**hosts.conf*** file of the Deployment Framework on the host machine where you are developing the blade:

Add this line to *global_config/hosts.conf*

```
define NewAdapterBlade${THIS_LEG}_HOST     <hostname>
```

where `<hostname>` is the hostname of the development host machine on which the Integration Adapter runs.

> **Tip:** `<hostname>` can be set to `localhost`, but it is good practice to always set it to the actual hostname of the machine.

- Make sure the Transformer and/or Liberator are running on their host machines (so as to accept connection requests from the new Integration Adapter).

- To start the blade's Integration Adapter, run the following command from *NewAdapterBlade/DataSource/*:

  `./etc/<binary name>` **`start`**

- When the Integration Adapter has started, examine its log files in *NewAdapterBlade/DataSource/var/* to see if it has connected to the Transformer and/or Liberator as appropriate.

> **Tip:** An Integration Adapter can create several types of log file. Generally there will be an event log and a packet log. There may be other DataSource specific logs depending on your Adapter implementation.

- If the Integration Adapter should return data to Liberator, use the Liberator's Liberator Explorer or Object Browser to request data for the relevant subjects (for example, data for the subjects `/LIBERATORBLADE` and `/TRANSFORMERBLADE`).

- To stop the blade's Integration Adapter, run the following command *from NewAdapterBlade/DataSource/*:

  `./etc/<binary name>` **`stop`**

4 Packaging the new blade

Before a new Adapter blade or Lua module blade can be deployed to the production Deployment Framework, you must package it into a kit.

- The name of the blade kit must have the format:

  `<blade_name>-<build_number>`

  `<build_number>` is optional, but the `<blade_name>` must be followed by a '–' (hyphen character).

Creating the blade kit on Windows

- To create the new blade kit on Windows®, use the following command:

  ```
  zip –qyr <blade_name>-<build_number>.zip <blade_name>
  ```

  For example:

  ```
  zip –qyr NewAdapterBlade-000017.zip NewAdapterBlade
  ```

Creating the blade kit on Linux

- To create the new blade kit on Linux®, use the following commands:

  ```
  tar cf <blade_name> -<build_number>.tar <blade_name>
  gzip <blade_name> -<build_number>.tar
  ```

  For example:

  ```
  tar cf NewAdapterBlade -000017.tar NewAdapterBlade
  gzip NewAdapterBlade -000017.tar
  ```

5 Deploying the finished blade

When you have finished developing an Adapter blade or Lua module blade, you can deploy it to the production Deployment Framework.

5.1 Removing old versions of the blade

- Before deploying the new version of a blade, remove all `<BladeName>` directories (for example *NewAdapterBlade*) from the `active_blades` directory of all the production Deployment Frameworks.

5.2 Setting up an Adapter blade's host machine

If your blade is an Adapter blade, you must decide what production server machine the Integration Adapter is to run on, and, if failover is enabled, the production server machine on which its failover instance is to run.

- Add the following lines to the Deployment Framework's *global_config/**hosts.conf*** file on all the production servers to which the Framework is deployed:

```
define <blade_name>${THIS_LEG}_HOST   <hostname_of_primary_machine>

if "${FAILOVER}" == "ENABLED"
   define <blade_name>${OTHER_LEG}_HOST
                                          <hostname_of_secondary_machine>
endif
```

where:

`<blade_name>` is the name of the blade; for example, `NewAdapterBlade`.

Note: Each blade that is deployed to a Deployment Framework must have a unique name.

`<hostname_of_primary_machine>` is the hostname of the server machine on which the primary instance of the Integration Adapter is to be located. If failover is not configured, this is the only machine on which the Integration Adapter runs. If failover is configured, it is the machine where the Integration Adapter in the primary failover leg runs.

`<hostname_of_secondary_machine>` is the hostname of the server machine where the Integration Adapter in the secondary failover leg runs. If you do not require the Integration Adapter to support failover, you do not need to include the lines `"${FAILOVER}"` … `endif` in *hosts.conf*.

Adding these lines ensures that the new Integration Adapter only starts on the host it is configured to start on.

5.3 Deploying the blade

To deploy a new blade to the Deployment Framework:

- Copy the new blade kit to the *kits* directory of the Caplin Platform Development Framework on all the server machines that host the Framework.
- Run the `deploy.sh` script on each server machine:

  ```
  ./kits/deploy.sh
  ```

  This script stops any Caplin Platform components that are running on the server, unpacks (unzips) the blade kit, and activates the new blade.

Note: The *deploy.sh* script also deploys the Caplin Platform components, but does *not* start these components.

- Start the Caplin Platform Deployment Framework on all the servers.
  To do this, from the root directory of the installed Deployment Framework run the script:

  ```
  start-all.sh
  ```

  For more information about starting and stopping the Deployment Framework, see the document **Caplin Platform: How To Use The Deployment Framework**.

5.4 Troubleshooting the blade

Troubleshooting Adapter blades

If your new Integration Adapter will not start properly, or misbehaves when running, here is a list of commonly occurring faults to check for:

- Have you forgotten to create a *var* directory in the blade?
- Is the blade's host machine defined in the *hosts* file?
  See section 5.2 "Setting up an Adapter blade's host machine".
- Is there erroneously more than one instance of the Integration Adapter running?
  - For Integration Adapters that connect to the Liberator, you can easily check this by looking at the Liberator's status page, where you will find that the DataSources section shows the Adapter as a flashing entry.
  - For Integration Adapters that connect only to the Transformer, look in the *servers/Transformer/var/transformer*.* files on the server under which the Transformer is running, and check for multiple peer connection and disconnection messages.
- Are there any errors shown in the log files in the blade's *var* directory?
- On the server where the Liberator is running, check the *servers/Liberator/var/event-rttpd*.* files for errors. Look out especially for field-related errors (reused field names or numbers).
- If you are using a Transformer, on the server where the Transformer is running, check the *servers/Transformer/var/transformer*.* files for errors. Look out especially for field-related errors (reused field names or numbers).

Troubleshooting Lua Module blades

- See "Deploying and testing the Lua module blade" in section 6.3 "Developing the new Lua module".

6 Creating and developing a Lua module blade

This section explains how to create and develop a Service blade for Caplin Transformer, where the blade's logic is written in Lua as a **Transformer module**.

6.1 Lua module blade structure

A Lua module blade contains:

- The blade's field definitions in *<blade_name>/blade_config/fields.conf*
- Liberator related configuration in *<blade_name>/Liberator/etc/rttpd.conf*
- Transformer related configuration in *<blade_name>/Transformer/etc/transformer.conf*
- The configuration for the Lua pipeline in *<blade_name>/Transformer/etc/pipeline.conf*
- The Lua source for the pipeline in *<blade_name>/Transformer/etc/<BladeName>.lua*

6.2 Creating the Lua module blade

The following sections describe how to create a Lua module blade. The best way to describe the development is to describe an example blade called **NewLuaBlade**. This blade provides `/LUALIBERATOR` data which can then be requested by a Liberator.

Deploying the Caplin Platform Deployment Framework

Since the blade is part of the Caplin Platform Deployment Framework it uses configuration from that kit. This configuration must be available during development. A convenient way to achieve this is to deploy a Caplin Platform Deployment Framework on the machine where you wish to develop the Lua module blade.

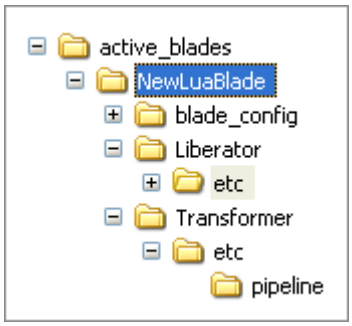
- To deploy the Framework, follow the instructions in the document **Caplin Platform: How To Use The Deployment Framework**.

- You will need a Caplin Transformer and Caplin Liberator to help test your blade whilst you develop it. The Transformer should be deployed on the machine where you are developing the blade.
  If you do not have Liberator already deployed on a machine for testing purposes, you could deploy an instance on your development machine, to the Framework you have just deployed. To do this, follow the instructions in the above document.

Creating the Lua module blade's directory structure

- Create a directory with the name of the blade (for example, *NewLuaBlade*) and its subdirectories.

  The directory tree must be in the *active_blades* directory of the Caplin Platform Deployment Framework on your development machine. It must have the following structure:



Creating the *fields* file

- Create a *fields.conf* file listing all the fields provided by the Lua module and put it in the *blade_config* directory of the blade.

Example of *fields.conf* entries for a Lua module blade

```
add-field    NewDSField1     1     0
add-field    NewDSField2     2     0
add-field    NewDSField3     3     0
```

Writing the core component configuration

Your new Lua module blade must include configuration for the core components that it interacts with. This configuration should typically consist of:

- Data service definitions for the Liberator that allow the Liberator to request the data provided by the blade.
- Transformer pipeline component configuration for the Lua module.

Background information

- For more information about data services, see the **Caplin Platform Overview** and the **Caplin Liberator Administration Guide**.

Liberator configuration

- Create a Liberator configuration file called ***rttpd.conf*** in *NewLuaBlade/Liberator/etc/*

The contents of this file are typically:

Lua module blade configuration for Liberator (*rttpd.conf*)

```
#
# Liberator configuration for data from the NewLuaBlade Lua module blade.
#
#

add-data-service
        service-name            NewLuaBlade${THIS_LEG}
        include-pattern         ^/LUALIBERATOR /

         add-source-group
                required        true
                add-priority
                        label   transformer${THIS_LEG}
                end-priority
                if "${FAILOVER}" == "ENABLED"
                        add-priority
                                label   transformer${OTHER_LEG}
                        end-priority
                endif
        end-source-group
end-data-service
```

This is fairly standard configuration. The blade specific points to note are:

- The blade name is used in **service-name** configuration options (see section 2.1 “Blade naming conventions”).
- The macros `THIS_LEG` and `OTHER_LEG` are used to differentiate the configuration of primary and secondary failover legs.

Transformer configuration

- Create a Transformer pipeline configuration file called ***pipeline.conf*** in *NewLuaBlade/Transformer/etc*
- The contents of this file are typically:

Lua module blade's Transformer pipeline configuration (*pipeline.conf*)

```
#
# Location of the Lua module blade's pipeline components
#
pipeline-paths  ${ccd}/pipeline/

add-pipeline
    id               NewLuaBlade${THIS_LEG}
    pipeline-file    NewLuaBlade.lua

    provider-regex  ^/LIBERATORLUA/.*

    request-func    request
end-pipeline
```

The blade specific points to note are:

- The blade name is used as the pipeline **id**, and in the Lua module name that is defined by the **pipeline-file** configuration option.

- The `ccd` macro defines the current directory path of the file in which it is referenced. So in this example, because the `ccd` reference is in the file *NewLuaBlade/Transformer/etc/pipeline.conf*, the macro defines the path *NewLuaBlade/Transformer/etc*

- The line
  `pipeline-paths ${ccd}/pipeline/`
  instructs the Transformer to look for the blade's pipeline files in the directory *NewLuaBlade/Transformer/etc/pipeline/*

6.3 Developing the new Lua module

- Write your Lua module to conform to the requirements of a Transformer pipeline module.

- When the development is complete, copy the Lua files that implement the pipeline into the into the directory *<BladeName> NewLuaBlade/Transformer/etc/pipeline/*

  For example, copy the files to *NewLuaBlade/Transformer/etc/pipeline/*

Deploying and testing the Lua module blade

- Copy the new Lua blade's directory structure and contents to all the *active_blades* directories of all the Caplin Platform Deployment Frameworks in your test environment.

  For example, copy *NewLuaBlade* and its subdirectories into *active_blades*

- Stop each of the Deployment Frameworks and restart them.

  (See "Starting and stopping the Deployment Framework" in **Caplin Platform: How To Use The Deployment Framework**.)

  At this stage the Liberator and Transformer are running and the Liberator should be able to request the data provided by the Transformer pipeline module that the new blade implements.

- If there is any issue with your new Lua module blade, check for errors in the *servers/Transformer/var/pipeline.** files on the server where the Transformer is running.

6.4 Deploying the new Lua module blade to the production system

When you have tested the new Lua blade to your satisfaction, you can deploy it to the production system:

- First, package the blade into a deployable kit:
  follow the steps in section 4 "Packaging the new blade".
- Then deploy the blade kit:
  follow the steps in section 5 "Deploying the finished blade".

7 Glossary of terms and acronyms

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

Term	Definition
Adapter blade	A **blade** for the **Caplin Platform** that consists of an **Integration Adapter** and associated configuration. Also see, **C-based Adapter blade**.
Blade	A re-usable software module containing the code and resources needed to implement a business feature. In this document the term blade is short for **Caplin Platform blade**.
C-based Adapter blade	An **Adapter blade** written in C using the **C DataSource API**.
Caplin Integration Suite (CIS)	A set of APIs and tools for creating adapters that integrate the **Caplin Platform** with external systems.
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 blade	A **blade** designed for use with the **Caplin Platform**. A Caplin Platform blade can be an **Adapter blade**, **Config blade**, or **Service blade**. Also see, **C-based Adapter blade** and **Lua module blade**.
Caplin Platform Deployment Framework	A configuration and deployment environment for the **Caplin Platform** that supports **Caplin Platform Blades**.
Caplin Transformer	An event-driven, real-time data transformation engine optimized for web trading services.
C DataSource API	An implementation of the **DataSource API** that is written in the C language.
Config blade	A **blade** for the **Caplin Platform** that enables a feature through configuration.
Core component	A Caplin Platform component that is supplied with the **Caplin Platform Deployment Framework**, but is not a blade. The core components are **Caplin Liberator** and **Caplin Transformer**.

Term	Definition
Data service	A set of rules used by the **Caplin Platform** to determine how a requested data item should be sourced based on its subject. A data service can incorporate priority, failover and load balancing. Data services are defined in **Caplin Liberator** configuration.
DataSource	DataSource is the messaging infrastructure used by the **Caplin Platform** and **Integration Adapters**.
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.
Deployment Framework	In this document, this term is short for the **Caplin Platform Deployment Framework**.
Framework	In this document, this term is short for the **Caplin Platform Deployment Framework**.
Integration Adapter	A server application that allows an external system to communicate with the **Caplin Platform**. An Integration Adapter is a **DataSource application**.
Liberator	The short form of **Caplin Liberator**.
Lua	A scripting language devised by the Pontifical Catholic University of Rio de Janeiro. See http://www.lua.org/.
Lua module blade	A **Service blade** for **Caplin Transformer** that is written in Lua.
Service blade	A blade for the **Caplin Platform** that includes a **Transformer module** or a **Liberator Auth module**. Also see, **Lua module blade**.
Transformer	The short form of **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.

CAPLIN

THE WEB TRADING TECHNOLOGY COMPANY

Contact Us

Caplin Systems Ltd

Cutlers Court

115 Houndsditch

London EC3A 7BR

Telephone: +44 20 7826 9600

www.caplin.com

Caplin Platform 6.0: How To Create C And Lua Blades, December 2012, Release 2