

CAPLIN

# Caplin Integration Suite 6.0

---

## How To Create A Platform Java Blade

December 2012



## Contents

<b>1</b>	<b>Preface .....</b>	<b>1</b>
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.....	2
1.5	Feedback.....	3
1.6	Acknowledgments .....	3
<b>2</b>	<b>About the Caplin Integration Suite Toolkit .....</b>	<b>4</b>
2.1	What is the Caplin Integration Suite Toolkit? .....	4
2.2	Supported blade types.....	4
2.3	Installing the plugin and the command line utility .....	5
<b>3</b>	<b>Creating a blade .....</b>	<b>6</b>
3.1	Wizard step 1:.....	8
3.2	Wizard step 2:.....	9
3.3	Wizard step 3:.....	11
3.4	Blade file structure .....	13
<b>4</b>	<b>Running your blade from Eclipse.....</b>	<b>15</b>
4.1	Changing the blade's run configuration.....	17
4.2	Creating a new run configuration.....	19
4.3	Debugging the blade .....	20
<b>5</b>	<b>Exporting your blade as a kit .....</b>	<b>21</b>
5.1	To export an Adapter blade: .....	21
5.2	To export a Config blade: .....	23
<b>6</b>	<b>Creating a blade from the command line.....</b>	<b>24</b>
6.1	Command for creating a new blade project.....	25
6.2	Command for building a blade.....	27
6.3	Command for running a blade .....	28
6.4	Command for exporting an existing blade project.....	30
<b>7</b>	<b>Glossary of terms and acronyms .....</b>	<b>32</b>

# 1 Preface

## 1.1 What this document contains

This document explains how to create new Java®-based Caplin Platform blades, using the Caplin Integration Suite Toolkit.

If you develop your blades using the **Eclipse™** IDE, read sections 2, 3, 4, and 5.

If you do not use Eclipse, you will need to use the Toolkit's command line utility instead; see sections 2.1, 2.2, and 6.

### 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. Before reading it, 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 DataSource Overview** (all sections)
- ◆ **Caplin Liberator Administration Guide** (Overview section)

## 1.3 Related documents

- ◆ **Caplin Platform Overview**  
A technical overview of the Caplin Platform.
- ◆ **Caplin Platform: Deployment Framework Overview**  
Explains the concept of the Caplin Platform Deployment Framework, including Caplin Platform blades and how they are deployed.
- ◆ **Caplin Liberator Administration Guide**  
Describes the Caplin Liberator server and its place within the Caplin Platform.  
It includes configuration reference information and a list of Liberator log and debug messages.
- ◆ **Caplin DataSource: Overview**  
A technical overview of Caplin DataSource.

#### ◆ DataSource For C Configuration Syntax Reference

Describes the syntax of the language that is used to configure DataSource applications (such as Integration Adapters) that use *.conf* configuration files.

## 1.4 Typographical conventions

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

Type	Uses
<b>aMethod</b>	Function or method name
<i>aParameter</i>	Parameter or variable name
<i>/AFolder/Afile.txt</i>	File names, folders, and directories
<div style="border: 1px solid black; padding: 2px;">Some code;</div>	Program output and script examples
The value=10 attribute is...	Script 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
<b>Glossary term</b>	Items that appear in the “Glossary of terms and acronyms”
<b>XYZ Product Overview</b>	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.

## 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](mailto:documentation@caplin.com).

## 1.6 Acknowledgments

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

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

*Eclipse* is a trademark of Eclipse Foundation, Inc.

## 2 About the Caplin Integration Suite Toolkit

### 2.1 What is the Caplin Integration Suite Toolkit?

The Caplin Integration Suite Toolkit provides the ability to create and export **Caplin Platform blades** that drop in to the **Caplin Platform Deployment Framework**.

You can create a blade from within the Eclipse™ IDE by first installing a plugin from the Toolkit (the Caplin Integration Suite for Eclipse plugin) and then using the plugin's Caplin Integration Adapter project wizard. The plugin also allows you to run and debug a blade project when you develop using Eclipse.

Alternatively, you can use a command line utility (via the `java -jar ...` command) to create blades, and to export blades, as explained in section 6.

**Note:** In the rest of this document the term “blade” means “**Caplin Platform blade**”.

For information about the Caplin Platform Deployment Framework see the document **Caplin Platform Deployment Framework Overview**.

### 2.2 Supported blade types

The Caplin Integration Suite Toolkit supports the following types of blade:

- ◆ **Adapter blades** written in Java.

An Adapter blade connects to, and supplies data to, a **Caplin Liberator** or **Caplin Transformer**. It consists of an **Integration Adapter** (an executable binary file), DataSource configuration, and core component configuration. The Integration Adapter must be written in Java, using one or more of the **Caplin Integration Suite's** Java APIs.

- ◆ **Config blades**

A Config blade enables a feature purely through configuration of existing Caplin Platform components.

The Caplin Platform Deployment Framework comes with several built-in blades of this type. For more information, see the **Caplin Platform Deployment Framework Overview**.

## 2.3 Installing the plugin and the command line utility

Before you can create new Caplin Platform blades using Eclipse, you must install the Caplin Integration Suite plugin.

- You can obtain the plugin from the Caplin website at <http://www.caplin.com/developer/eclipse>.

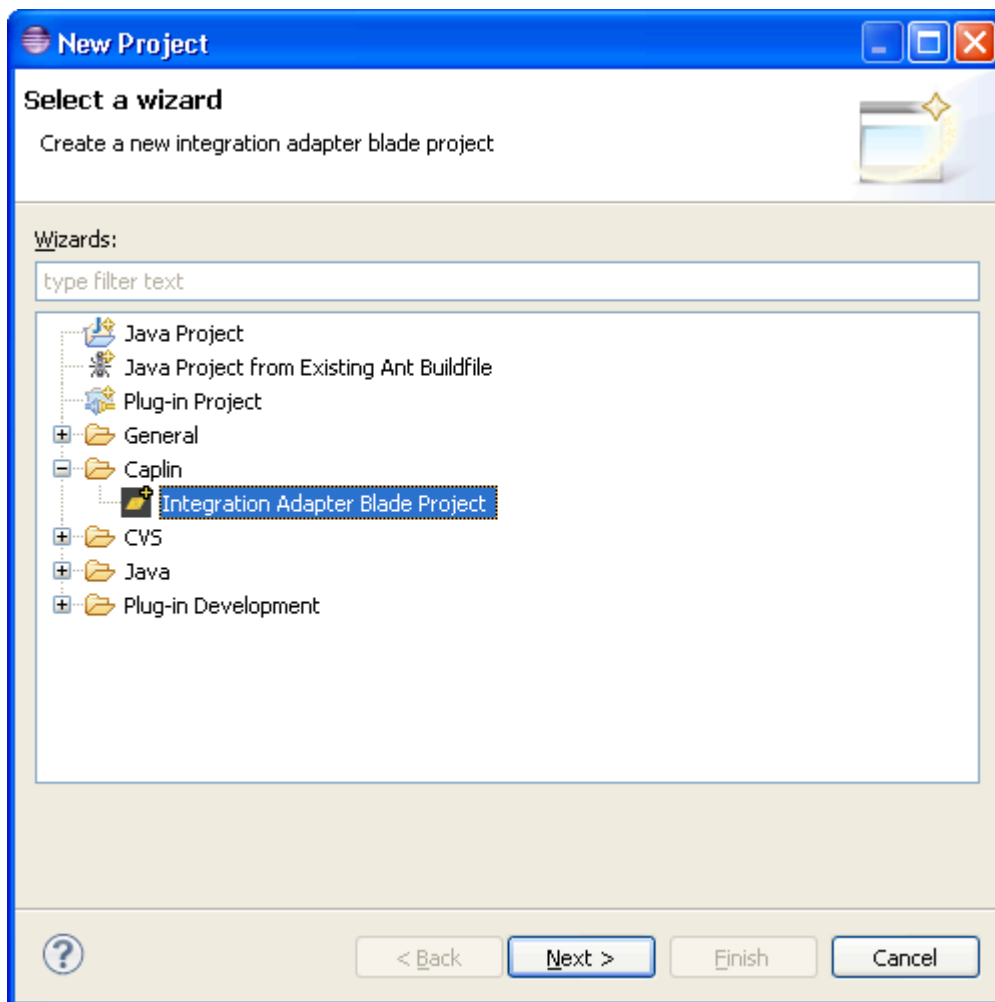
**Tip:** For information on how to install Eclipse plugins, refer to the documentation that came with your Eclipse distribution.

- The command line utility described in section 6 “Creating a blade from the command line” is in the tools directory of the unzipped Caplin Integration Suite.

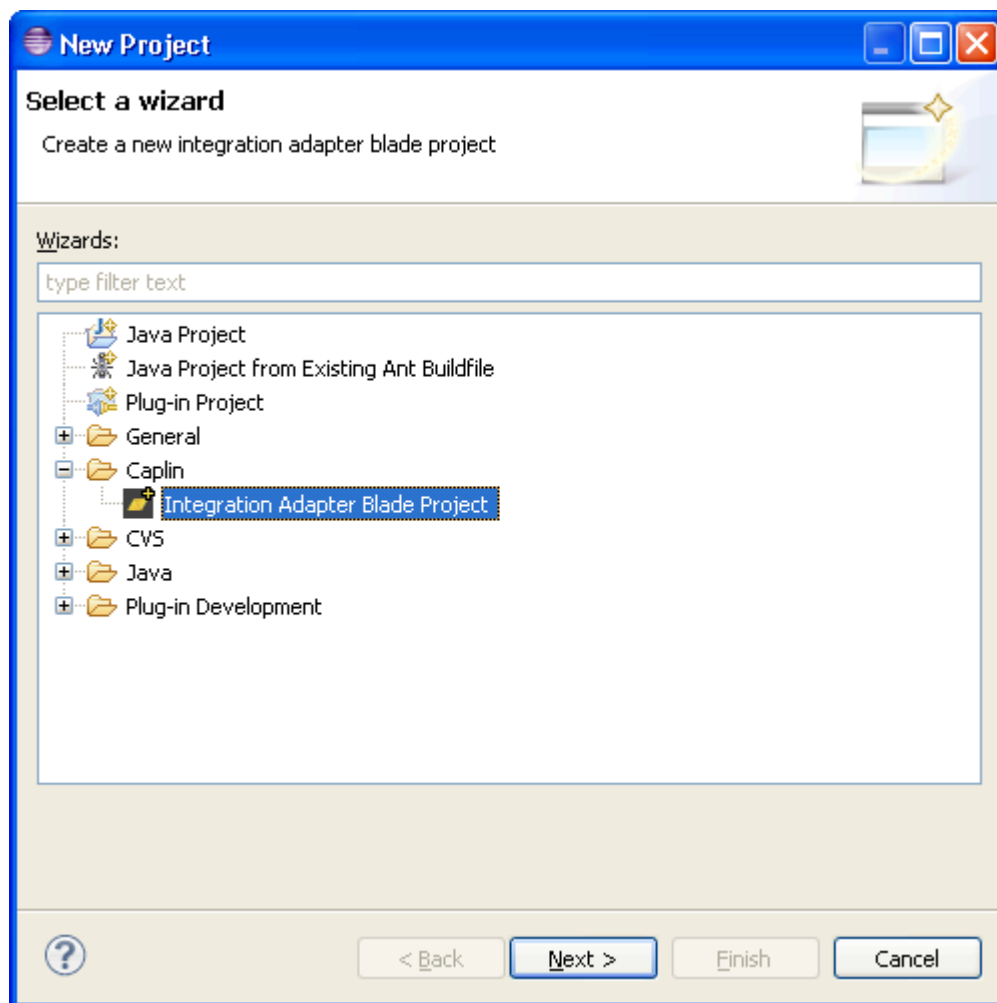
### 3 Creating a blade

You create a blade project from Eclipse using the Caplin Integration Adapter Blade project wizard.

- From Eclipse's New Project dialog (File > New > Project...), select the Caplin node, followed by the node called Integration Adapter Blade Project.







- Select Next.  
The Caplin Integration Adapter project wizard starts.

### 3.1 Wizard step 1:

**Create an Integration Adapter Blade Project**

Create an Integration Adapter Blade Project

Enter a project name.

Project name:

☒ Use default location

Location:

**Base package**

This package will be used to generate code.

Package:

**Working sets**

☐ Add project to working sets

Working sets:

- Enter a project name

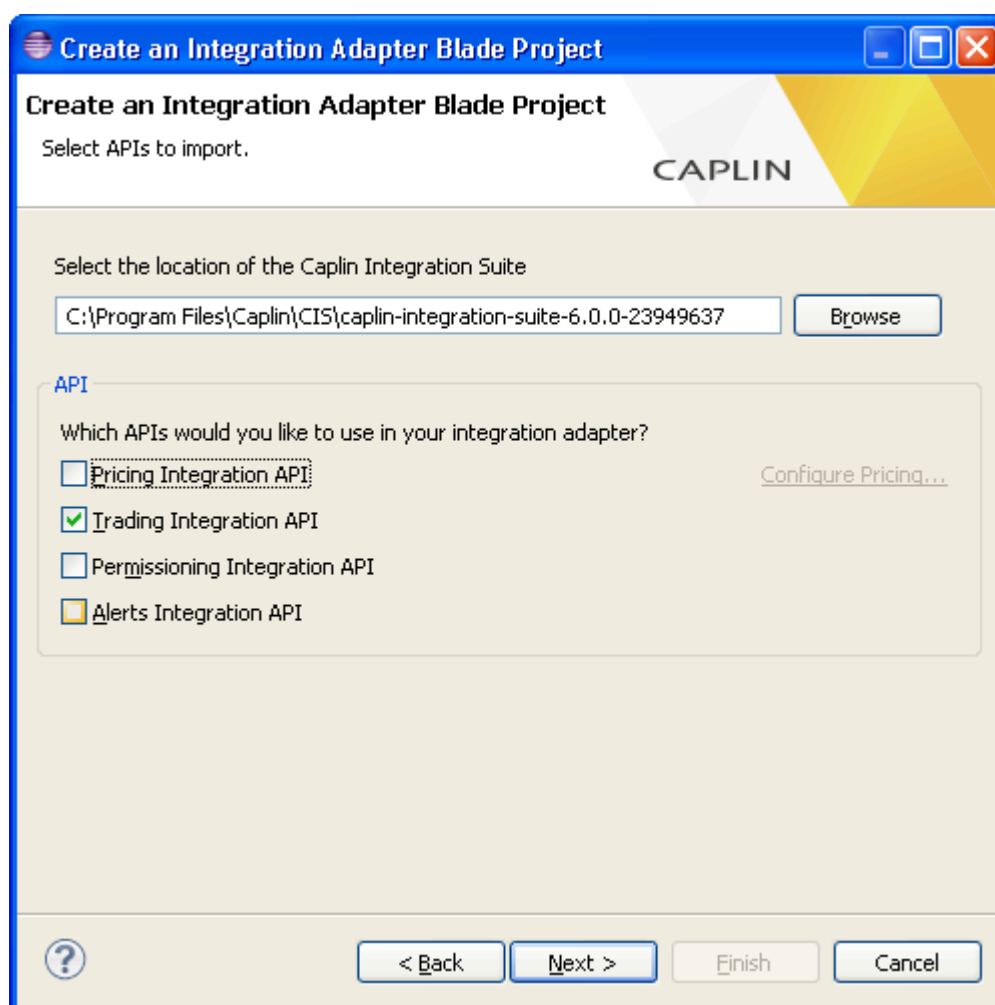
This is the name of the blade that is to be created. The name is also used in Java class names, so it must conform to Java naming standards.

The Use default location box is automatically checked when the wizard dialog is displayed. The blade is therefore created in the location shown in the grayed-out Location box.

- To change the blade location, uncheck the Use default location box, and either enter a file path in the Location box or browse to the desired location.
- You can optionally provide a Java package name for the generated blade Java classes. If you do not provide one, the classes are placed in the default (unnamed) package.

- You can optionally add the new project to an Eclipse working set. Consult your Eclipse documentation for information about Eclipse working sets.
- Select Next to move to step 2 of the wizard.

## 3.2 Wizard step 2:



- First, make sure that the Caplin Integration Suite has been unzipped to a suitable location in the file system.

For example, the location could be a folder called *caplin-integration-suite-[version]-[build]*, which was created by unzipping a Caplin Integration Suite kit of the same name.

- In the wizard, enter the location of the unzipped Caplin Integration Suite, either as a file path, or by browsing to the location.

The wizard reports an error if the required contents of the suite are missing.

- In the API section of the wizard, select which Caplin Integration APIs to include in the project. These are presented as Pricing Integration API, Trading Integration API, Permissioning Integration API, and Alerts Integration API, and you can select one or more to create a blade.

Your selections determine which libraries are included in your blade project, how the blade is configured, and which source files are created.

- If you have selected Pricing Integration API, you have the option to configure whether the Integration Adapter will connect to Transformer or to the Liberator:  
select the link [Configure Pricing...](#)

By default a Pricing Adapter will connect to Transformer to enable data filtering.

- Select Finish to create the blade.

### 3.3 Wizard step 3:

- If at step 2 you selected Trading Integration API, or Trading Integration API and Pricing Integration API, the wizard prompts for an Asset Class as shown below. A valid Asset Class must only contain alpha-numeric characters; for example, FX.

**Create an Integration Adapter Blade Project**

Create an Integration Adapter Blade Project

Enter namespace information.

**Adapter Configuration**

Enter the Asset Class to use for this Integration Adapter.

[What does this affect?](#)

Asset Class:

< Back Next > Finish Cancel

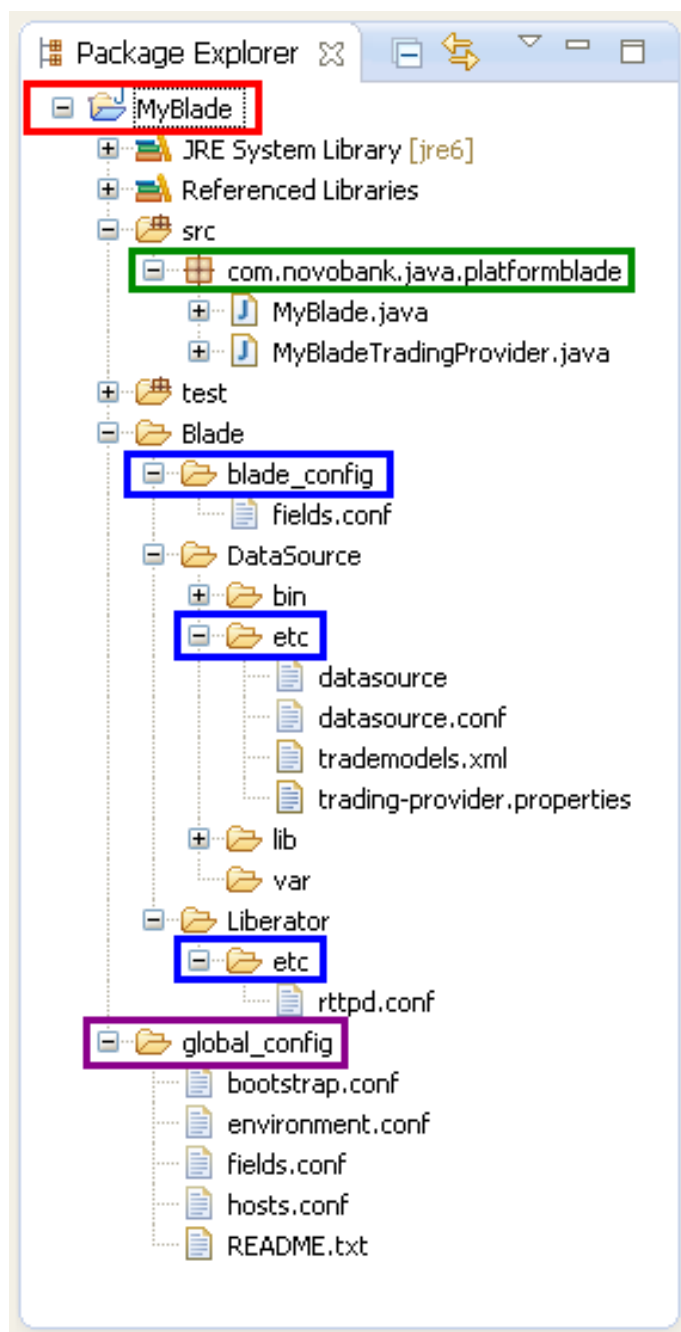
If you also selected Pricing Integration API, the subject prefix for received prices is derived from the asset class by adding a slash ('/') prefix and suffix. For example, if the entered asset class is FX, the derived subject prefix is /FX/


- If at step 2 you selected Pricing Integration API, but not Trading Integration API, the wizard prompts for a Subject Prefix instead.


A valid Subject Prefix must only contain alphanumeric characters. It must begin with a slash character ('/') and should end with a slash; for example, /FX/

### 3.4 Blade file structure

The following picture of Eclipse's Package Explorer shows the file structure of the newly created blade project resulting from the example Wizard settings shown in the previous section.



The blade files are located under a directory that has the name of the blade. In the example, this directory is *MyBlade*, marked **red**  in the picture of Eclipse Package Explorer.

The *src* subdirectory contains a Java package in which the source code files are located. In the example, the package is *com.novobank.java.platformblade*, marked **green**  in the picture of Eclipse Package Explorer, and the path of the corresponding subdirectory is *com/novobank/java/platformblade/*.

### Java source files

The Java source files created in this package are:

- ◆ *<BladeName>.java*

This file contains a main class and instantiates the providers used by your blade.

In the example, this file is *MyBlade.java*.

- ◆ The provider code stubs awaiting implementation.

There is a Java source file for each provider, with a name of the form


*<BladeName><AdapterType>Adapter.java*

In the example, there is just a single trading provider, whose code stub is in


*MyBladeTradingAdapter.java*.

### Blade component configuration

The configuration for the blade components is located in subdirectories of *<BladeName>/Blade*

In the example (marked **blue**  in the picture of Eclipse Package Explorer):

- ◆ The field definitions for the blade are in *MyBlade/Blade/blade\_config/fields.conf*
- ◆ The DataSource configuration is in *MyBlade/Blade/DataSource/etc/*
- ◆ The Liberator configuration is in *MyBlade/Blade/Liberator/etc/*

The *global\_config* subdirectory (marked **purple**  in the picture) contains configuration files that are used when the blade is connected to a remote host at run time (see section 4 “Running your blade from Eclipse”).



## 4 Running your blade from Eclipse

Caplin Platform blades are Java projects. However to run your blade in Eclipse, you need to use the Caplin blade launcher.

### Before running the blade:

An Adapter blade will usually connect to a Liberator or Transformer.

To ensure that the Liberator or Transformer is aware of the Adapter blade:

- Export the blade as a Config blade (see section 5.2)
- Deploy the exported Config blade to the Caplin Deployment Framework on the machine where the Liberator/Transformer is located (it could of course be the machine where you are developing the blade).

The hostnames configuration on your development machine (the machine where the blade is to run) must point to the correct location of the Liberator/Transformer:

- Edit the hostnames in `<BladeName>/global_config/hosts.conf` as required.

For example, edit `MyBlade/global_config/hosts.conf`

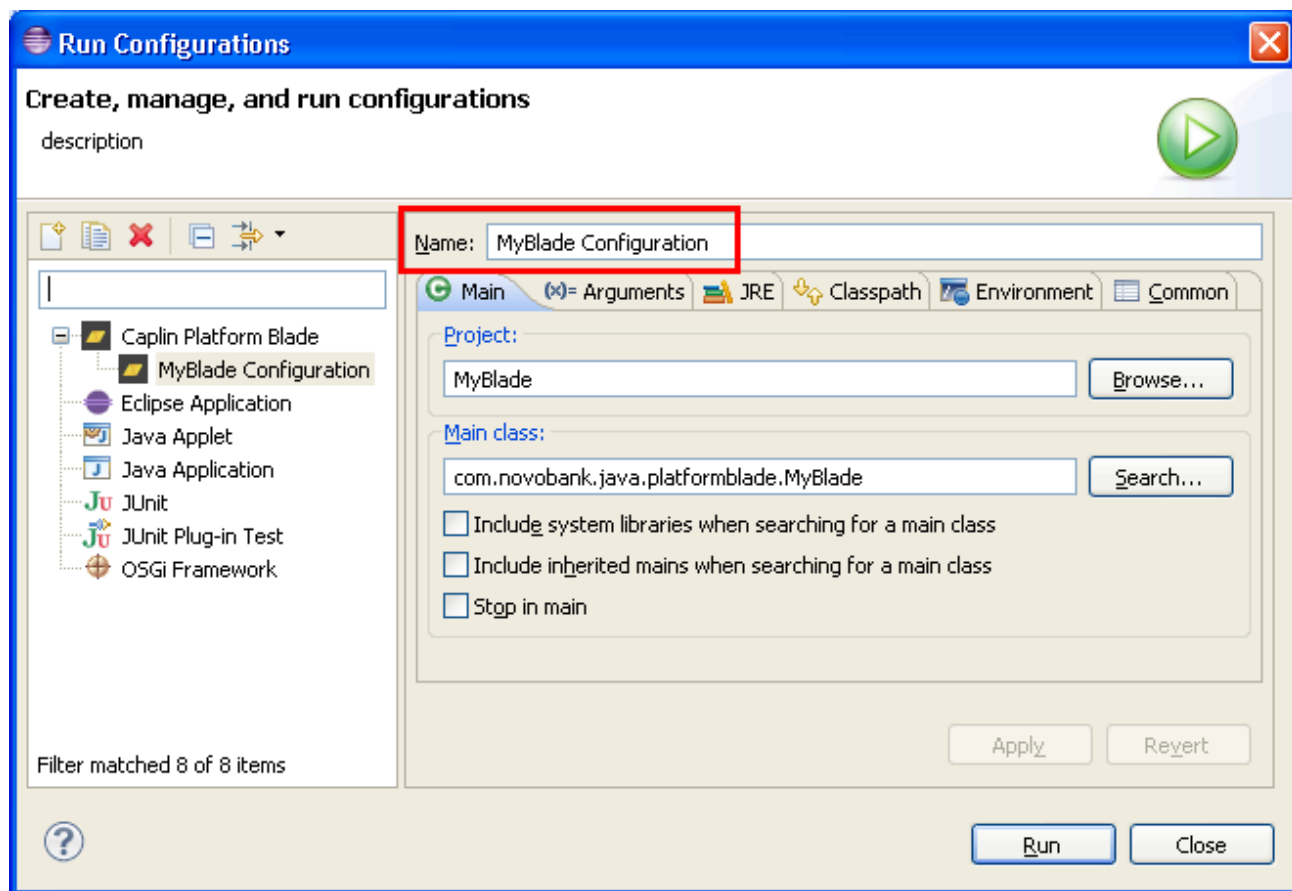
**Note:** If you omit these steps, when you run the blade, it will attempt to connect to the Liberator or Transformer but will fail to do so.

### To run the blade using the Caplin blade launcher:

- From the Run Configurations dialog in Eclipse (Run > Run Configurations), select the Caplin blade run configuration specific to your blade project.

The run configuration for your blade has a name of the form  
`<BladeName> Configuration`.

For example, `MyBlade Configuration`.

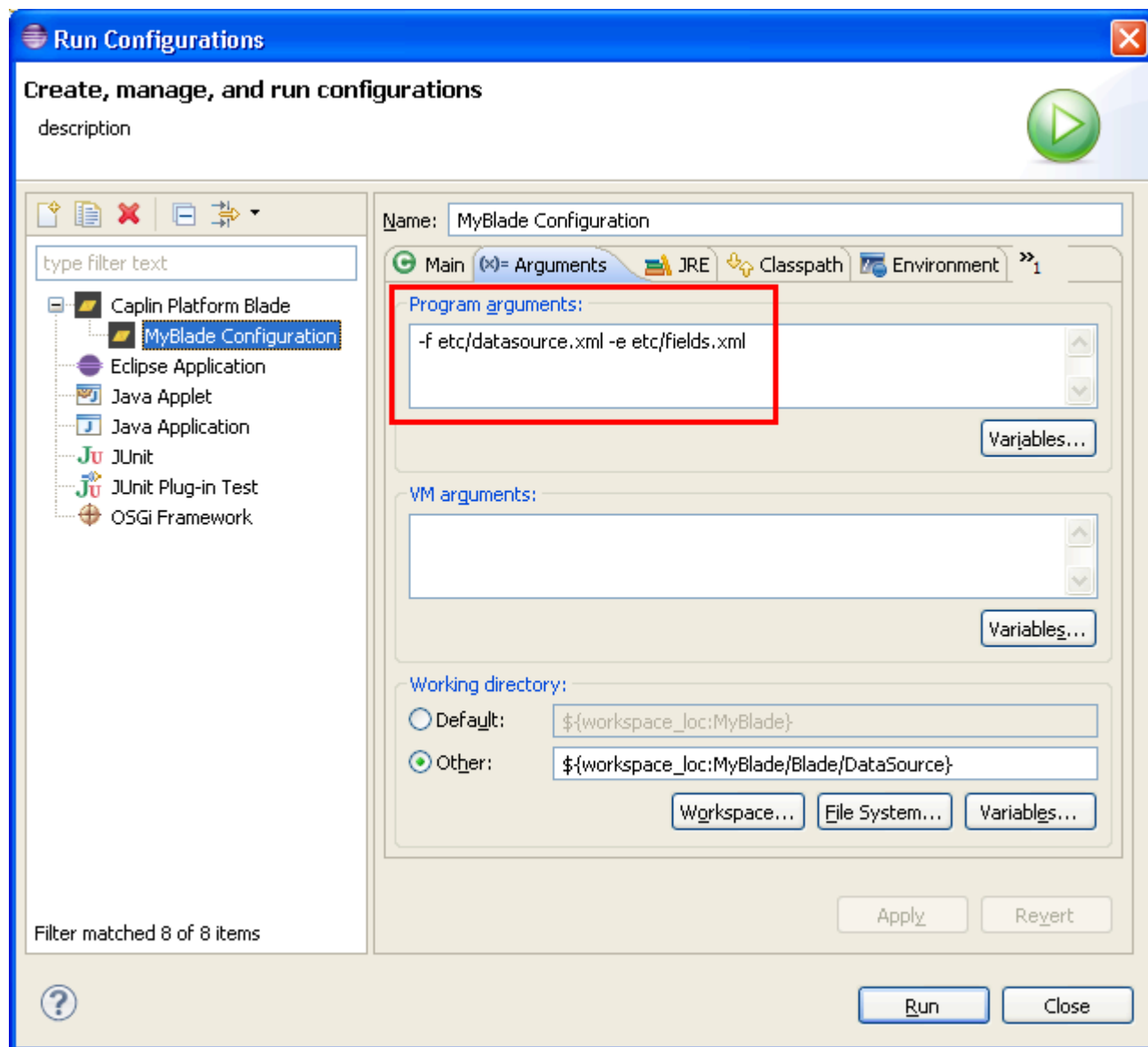


- Then select Run.

**Tip:** During the run, log files are created in your blade's *DataSource/var* directory. System output is written to the Eclipse console.

## 4.1 Changing the blade's run configuration

The Arguments tab of Eclipse's Run Configurations dialog shows pre-configured Program arguments that specify the location of configuration files for the blade:

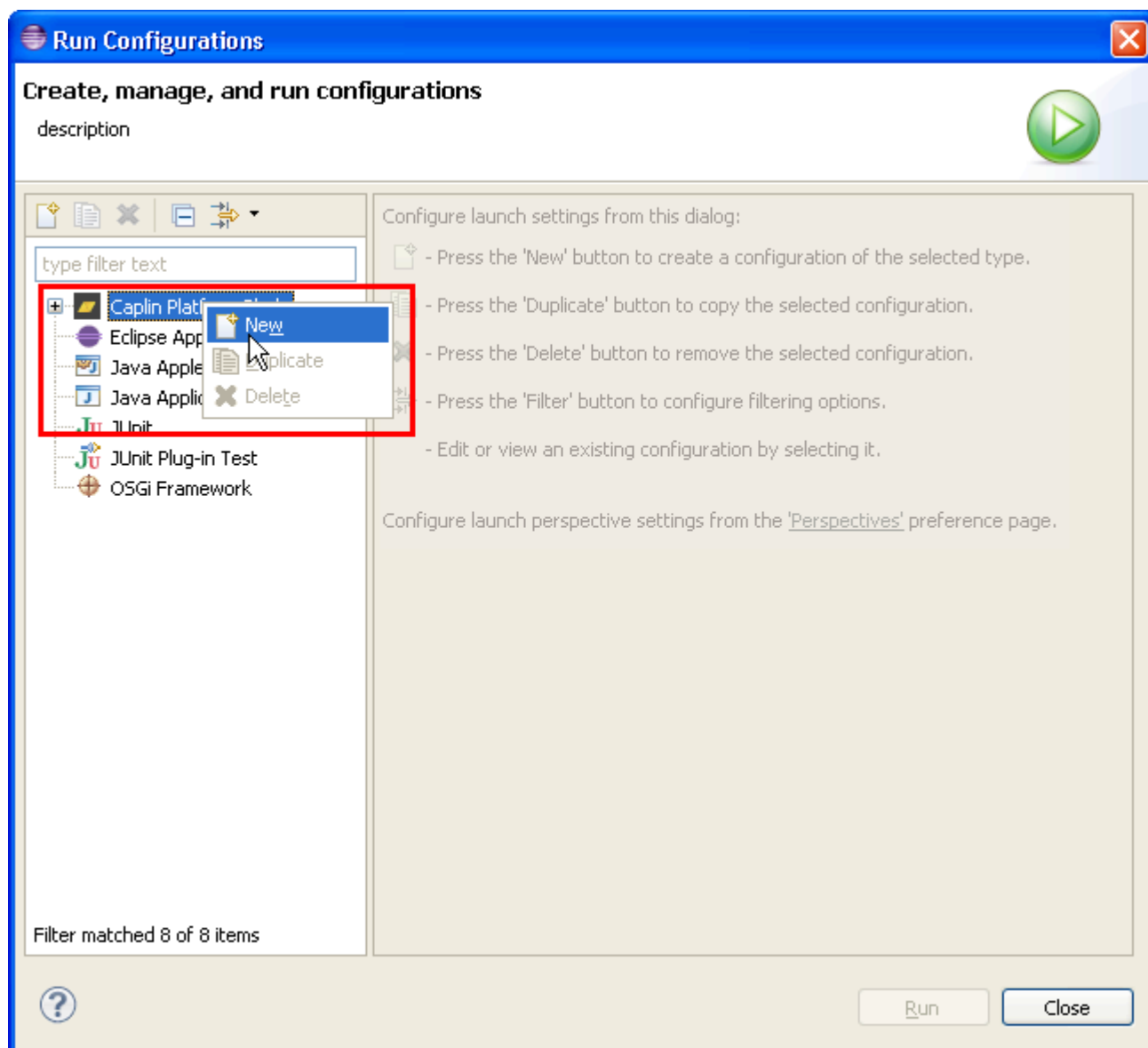


- Typically, you should not alter the program arguments automatically entered for you in the provided launcher, because the locations of these files must conform to the Caplin Platform Deployment Framework's expectation of a blade's directory structure. However, you may define additional program arguments required by your own code.

## 4.2 Creating a new run configuration

If a run configuration does not exist for your blade you can create one as follows:

- Right click on any file within your blade project and select Run As > Run Configurations...
- In the Run Configurations dialog, right click on the Caplin Blade icon and select New.



Eclipse creates a run configuration specific to your blade. You should name the run configuration so it is distinct from others that you create.

**Tip:** If you check out an existing blade project from a source code management system, there will probably be no run configuration defined for it, and you will have to create one.

### 4.3 Debugging the blade

Debugging your blade in Eclipse works in a similar way to running it:

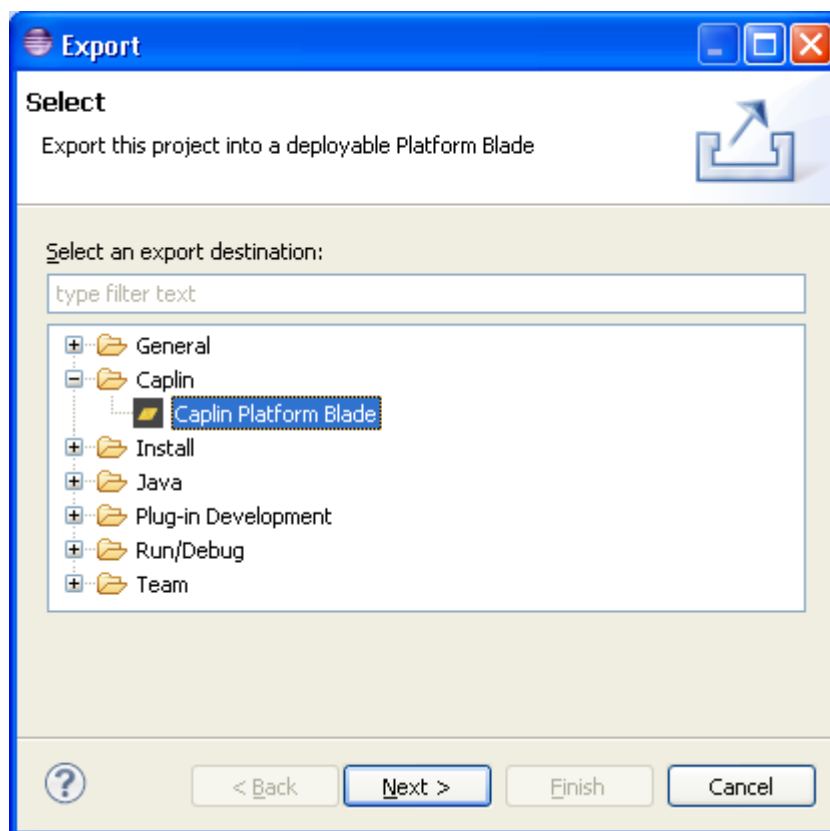
- From the Debug Configurations dialog in Eclipse (Run> Debug Configurations), select the Caplin blade run configuration specific to your blade project.
- Then select Debug.

## 5 Exporting your blade as a kit

To create a deployment blade that can be deployed to the Caplin Platform Deployment Framework, you must export your blade project using the Caplin Blade Export Wizard. Deploy the resulting kit by following the instructions in the document **Caplin Platform: Deployment Framework Overview**.

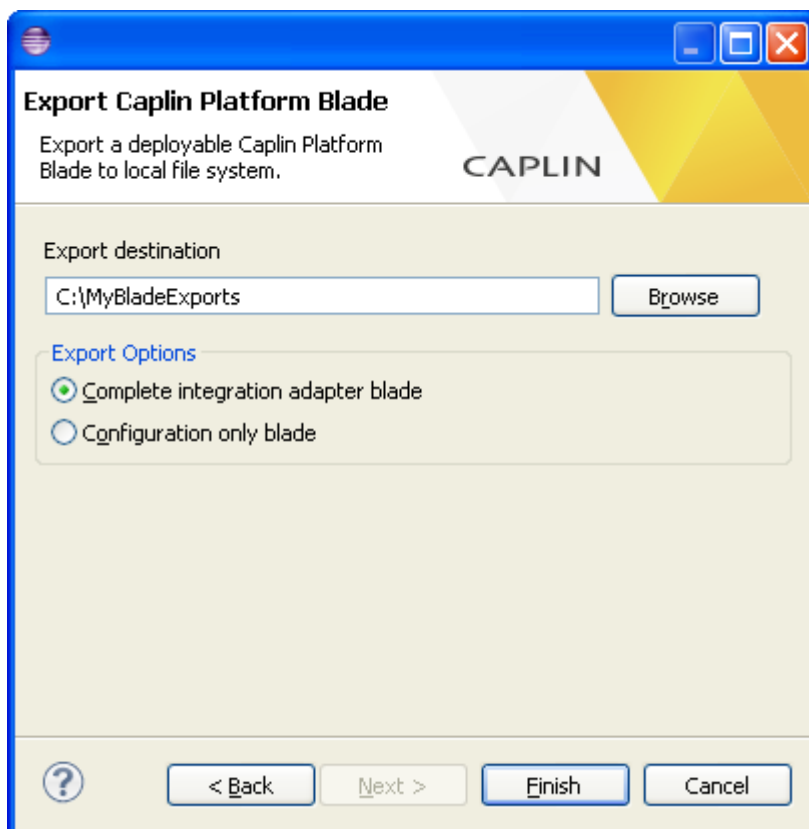
### 5.1 To export an Adapter blade:

1. Select any file within your blade project.
2. Right click and select Export.
3. In the Export dialog, navigate to Caplin and select Caplin Platform Blade.



4. Select Next.

The Export Caplin Platform Blade wizard is displayed:



5. In the Export Location box enter (or browse to) the path of the directory where the blade is to be exported to.
6. Ensure the Export option Complete integration adapter blade is selected.
7. Select Finish.

The blade is exported as a Zip archive.



## 5.2 To export a Config blade:

1. Select any file within your blade project in the navigator pane.
2. Right click and select Export.
3. In the Export dialog, navigate to navigate to Caplin and select Caplin Platform Blade.
4. Select Next.
5. In the Export Location box enter (or browse to) the path of the directory where the blade is to be exported to.
6. Ensure the Export option Configuration only blade is selected.
7. Select Finish.

The blade is exported as a Zip archive containing configuration for Liberator and/or Transformer as appropriate.

## 6 Creating a blade from the command line

The Caplin Integration Suite Toolkit can be used at the command line to create a new blade project. The command creates the directory tree, and generates blade configuration and source files.

The command line utility allows you to:

- ◆ Create a new blade project (**create** command – see section 6.1).
- ◆ Build a blade project on your development machine (**build** command – see section 6.2).
- ◆ Run a blade project on your development machine (**run** command – see section 6.3).
- ◆ Export an existing blade project as a Caplin Platform blade so it can be deployed to the Caplin Deployment Framework (**export** command – see section 6.4).

**Tip:** The command line utility is located in the tools directory of the unzipped Caplin Integration Suite.

- To run the Toolkit's command line utility, enter a command string of the form:

```
java -jar cis-blade-toolkit_<ver>.<build_id>.jar <command> <arguments>
```

For example:

```
java -jar cis-blade-toolkit_5.0.0.230420.jar create <arguments>
```

Sections 6.1 to 6.4 detail the <arguments> applicable to each command, and show example commands.

Additionally:

- To display basic usage instructions for the command line utility, type:

```
java -jar cis-blade-toolkit_<ver>.<build_id>.jar help
```

For example:

```
java -jar cis-blade-toolkit_5.0.0.230420.jar help
```

- For detailed help about each command, type:

```
java -jar cis-blade-toolkit_<ver>.<build_id>.jar help <command>
```

For example:

```
java -jar cis-blade-toolkit_<ver>.<build_id>.jar help create
```

## 6.1 Command for creating a new blade project

**Tip:** If the blade is to be developed in Eclipse, it is recommended that you create the blade project using the Caplin Integration Adapter project wizard, as described in section 3 of this document.

Command: **create**

Argument	Required/Optional	Description
-n --name	Required	The name of the Java blade (and hence the name of its main class).
-k --kit	Required	Path to the unzipped Caplin Integration Suite kit.
-d --directory	Optional	The location of the blade project. By default, the project structure is created in the current working directory.
-p --package	Optional	The package containing the source code. For example: <code>com.novobank.adapter</code>
-i --include	Required	The Caplin Integration APIs to include. The selection can be one or more of <code>pricing</code> , <code>trading</code> , <code>alerts</code> , or <code>permissioning</code> , separated by spaces.
-s	Optional, but see Description	<b>Subject Prefix or Asset Class</b>  If <code>pricing</code> is among the API selections, but not <code>trading</code> you must specify a Subject Prefix. A valid Subject Prefix must only contain alphanumeric characters. It must begin with a slash and should end with a slash; for example, <code>/FX/</code>  If <code>trading</code> is among the API selections, you must specify an Asset Class. A valid Asset Class must only contain alpha-numeric characters; for example, <code>FX</code>  If both <code>trading</code> and <code>pricing</code> are amongst the API selections, you must specify an Asset Class, as defined above. In this case, the subject prefix for received prices is derived from the asset class by adding a slash ('/') prefix and suffix. For example, if the entered asset class is <code>FX</code> , the derived subject prefix is <code>/FX/</code>

Argument	Required/Optional	Description
-L	Optional	Ensures that the Integration Adapter connects directly to the Liberator. Use this argument when <code>pricing</code> is specified in the <code>-i</code> argument.

**Example: Create a new pricing and trading blade from scratch:**

```
java -jar cis-blade-toolkit_5.0.0.<build_id>.jar create ^  
-n MyBlade -k ../../CaplinIntegrationSuite/ -d NewDirectory ^  
-i pricing -s /FX/ -p com.novobank.adapter
```

This example creates a pricing-only blade called `MyBlade` that uses the integration libraries from the given `CaplinIntegrationSuite`. The blade project is created in a subdirectory called `NewDirectory` within the current working directory.

## 6.2 Command for building a blade

Command: **build** <blade-name>

The **build** command allows you to build a blade that can run on your development machine. The command generates the configuration files necessary for running the blade locally. The only argument required is the name of the blade, which can include its path.

Compilation errors are output to the command line.

Argument	Required/Optional	Description
-cp -classpath	Optional	The class-path. By default the builder uses all jars it finds in <i>Blade/Datasource/lib</i> to create a class-path (in addition to the class-path defined by the environment, if any). This argument can be used to override the default with the jars in the specified class-path.

**build command example:**

```
java -jar cis-blade-toolkit_<ver>.<build_id>.jar build <blade-name>
```

## 6.3 Command for running a blade

Command: **run** <blade-name>

This command runs the blade on your development machine once you have built it.

### Before running the blade:

An Adapter blade will usually connect to a Liberator or Transformer.

To ensure that the Liberator or Transformer is aware of the Adapter blade:

- Export the blade as a Config blade (see the `export` command in section 6.4)
- Deploy the exported Config blade to the Caplin Deployment Framework on the machine where the Liberator/Transformer is located (it could of course be the machine where you are developing the blade).

The hostnames configuration on your development machine (the machine where the blade is to run) must point to the correct location of the Liberator/Transformer:

- Edit the hostnames in <BladeName>/global\_config/hosts.conf as required.

For example, edit *MyBlade/global\_config/hosts.conf*

**Note:** If you omit these steps, when you run the blade, it will attempt to connect to the Liberator or Transformer but will fail to do so.

### run command details

The only argument required to `run` is the name of the blade. If the blade is not in the current working directory, use the `-d` or `--directory` argument to specify its path.

While the blade is running you can stop it at any time by pressing the key combination `Control+C` (shown as `^C` in the command example below).

Any runtime errors are output to the command line.

Argument	Required/Optional	Description
<code>-cp</code> <code>-classpath</code>	Optional	The class-path. By default the runner uses all jars it finds in <i>Blade/Datasource/lib</i> to create a class-path (in addition to the class-path defined by the environment, if any). This argument can be used to override the default with the jars in the specified class-path.
<code>-d</code> <code>--directory</code>	Optional	The path to the blade. If no path is specified, the blade is assumed to be in the current working directory.

Argument	Required/Optional	Description
-m --mainclass	Optional	The name of the blade's main class. If the blade's main-class name has been changed since you created the blade project, you must supply this argument.  The default is <blade-name>DataSource.

**run command example:**

```
java -jar cis-blade-toolkit_<ver>.<build_id>.jar run <blade-name> ^
-d C:\Projects

<blade-name> blade is running (^C to exit)

^C

<blade-name> blade is shutting down
```

## 6.4 Command for exporting an existing blade project

Command: **export**

Argument	Required/Optional	Description
-n --blade-name	Required	The blade name.
-p --class-path	Required	A class path string that specifies the jars of any libraries used by the blade that are not in the Caplin Integration Suite.  Separate the individual jar paths by spaces, and enclose the whole string in quotes ("..."). Include this argument, if you want to invoke the exported jar with the command <code>java -jar</code>
-A --compiled-classes	Optional	The path to the compiled Java classes. The default is the <i>bin</i> directory in the root directory of the project.
-o --output	Optional	The output location of the archive. The default is the current working directory.
--config-only	Optional	Export a <b>Config blade</b> .

**Note:** Before exporting your project as a blade you must first compile the blade application's source code, otherwise the export operation will fail. For information on building your blade, see section 6.2.

**Tip:** The export command expects the compiled classes to be in the bin directory within the root directory of the project. However, if they have been created in a different location, you can use the -A (or --compiled-classes) argument to tell the export command where to find them.



**Export command example usage:**

```
java -jar cis-blade-toolkit_<ver>.<build_id>.jar export X:\CaplinBlades\
```

This example exports a blade called `MyPricingAdapter` located in `X:\CaplinBlades\`

## 7 Glossary of terms and acronyms

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

Term	Definition
<b>Adapter blade</b>	A <b>blade</b> for the <b>Caplin Platform</b> that consists of an <b>Integration Adapter</b> and associated configuration.
<b>Alerts Integration API</b>	An integration API that allows you to send alert notifications into the <b>Caplin Platform</b> from other systems.
<b>Blade</b>	A re-usable software module containing the code and resources needed to implement a business feature. In this document the term blade specifically means a <b>Caplin Platform blade</b> .
<b>Blade toolkit</b>	A set of commands to create, build and export <b>Caplin Platform blades</b> . See sections 3, 4, 5, and 6 of this document.
<b>Caplin Integration Suite (CIS)</b>	A set of APIs and tools for creating adapters that integrate the <b>Caplin Platform</b> with external systems. It includes the <b>blade toolkit</b> .
<b>Caplin Liberator</b>	A financial internet hub that delivers data and messages in real time to and from subscribers over any network.
<b>Caplin Platform</b>	An integrated suite of software that supports the services and distribution capabilities needed for web trading. It consists of <b>Caplin Liberator</b> , <b>Caplin Transformer</b> , Caplin KeyMaster, Caplin Director, and Caplin Management Console.
<b>Caplin Platform blade</b>	A <b>blade</b> designed for use with the <b>Caplin Platform</b> . A Caplin Platform blade can be an <b>Adapter blade</b> , <b>Config blade</b> , or <b>Service blade</b> .
<b>Caplin Platform Deployment Framework</b>	A configuration and deployment environment for the <b>Caplin Platform</b> that supports <b>Caplin Platform blades</b> .
<b>Caplin Transformer</b>	An event-driven, real-time data transformation engine optimised for web trading services. These services are implemented in <b>Transformer Modules</b> .
<b>Config blade</b>	A <b>Caplin Platform blade</b> that enables a feature through configuration.

Term	Definition
<b>DataSource</b>	<p>DataSource is the messaging infrastructure used by the <b>Caplin Platform</b> and <b>Integration Adapters</b>.</p> <p>In some older documents DataSource is also used as a synonym (but <i>non-preferred term</i>) for <b>DataSource application</b>.</p>
<b>DataSource API</b>	An API that allows server applications (including <b>Integration Adapters</b> ) to communicate with the <b>Caplin Platform</b> .
<b>DataSource application</b>	An application that uses the <b>DataSource API</b> . <b>Caplin Liberator</b> , <b>Caplin Transformer</b> , and <b>Integration Adapters</b> are all DataSource applications.
<b>Eclipse</b>	<p>An integrated software development environment (IDE) that supports Java.</p> <p>For more information about Eclipse, see <a href="http://www.eclipse.org">www.eclipse.org</a></p>
<b>Integration Adapter</b>	A server application that allows an external system to communicate with the <b>Caplin Platform</b> . An Integration Adapter is a <b>DataSource application</b> and is created using the <b>Caplin Integration Suite</b> .
<b>Liberator</b>	Short for <b>Caplin Liberator</b> .
<b>Liberator Auth Module</b>	A software module in <b>Caplin Liberator</b> that performs authentication and permissioning functions.
<b>Permissioning</b>	<p>The process of determining the access rights that an end-user has to resources, such as data and functionality.</p> <p>Also known as “authorization”.</p>
<b>Permissioning Integration API</b>	An API for sending <b>permissioning</b> information to the <b>Caplin Platform</b> . The Permissioning Integration API is part of the <b>Caplin Integration Suite</b> .
<b>Pricing Integration API</b>	An API for sending pricing information to the <b>Caplin Platform</b> . The Pricing Integration API is part of the <b>Caplin Integration Suite</b> .
<b>Service blade</b>	A blade for the <b>Caplin Platform</b> that includes a <b>Transformer module</b> or a <b>Liberator Auth module</b> .
<b>Trade model</b>	The definition of a particular trading workflow. It specifies all the states that a trade can be in, and the transitions between those states.

Term	Definition
<b>Trading Integration API</b>	An API for creating an <b>Integration Adapter</b> for trade messages that intelligently manages the <b>Trade model</b> for each trade. The Trading Integration API is part of the <b>Caplin Integration Suite</b> .
<b>Transformer</b>	Short for <b>Caplin Transformer</b> .
<b>Transformer Module</b>	A software module in <b>Caplin Transformer</b> that implements a service. For example, the Refiner module provides a Container filtering and sorting service.

## Contact Us

Caplin Systems Ltd

Cutlers Court

115 Houndsditch

London EC3A 7BR

Telephone: +44 20 7826 9600

**[www.caplin.com](http://www.caplin.com)**

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

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

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

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

This publication may contain links to third-party web sites; Caplin Systems Limited is not responsible for the content of such sites.