

Legs4Mule

User Guide

Fady Moussallam

fady@legsem.com

LegSem

Introduction

LegStar is an open-source solution for mainframe integration. LegStar primary focus is on IBM COBOL-CICS programs but its architecture can be easily extended to support other legacy environments (such as IBM IMS for instance).

It is unique among legacy integration solutions in its open-source licensing model and bi-directional capabilities. It also fosters an innovative COBOL to XML Schema binding framework inspired by JSR 222 (JAXB Java Architecture for XML Binding).

This guide assumes you are familiar with the LegStar core product. You can learn more about it at <http://www.legsem.com/legstar>.

Legs4Mule is a Mule transport. Thanks to the Mule Transport architecture, Mule components can access remote functions, or be accessed, without any knowledge of the actual transport being used. With Legs4Mule, these same components, untouched, can have access to, or be accessed by, IBM CICS programs.

This guide should give you a good idea of how Legs4Mule works.

Installation

Before you install Legs4Mule, you will need Mule itself and a working installation of LegStar. The LegStar distribution is available at <http://www.legsem.com/legstar/modules/legstar-distribution/>.

You can download Legs4Mule from the MuleForge web site at <http://dist.muleforge.org/legstar-transport/>. Unzip the content in a location of your choice and read the README.txt file.

Because of the complexity related with mapping COBOL structures to Java interfaces, Legs4Mule comes with development tools that generate code and configuration files. These tools are available in two different formats:

- Ant scripts; These are part of the Legs4Mule standard distribution in the ant folder.
- Eclipse plugins; These are not part of the Legs4Mule distribution but you can use the Eclipse update mechanism and point to <http://www.legsem.com/legstar/eclipse/update/>.

The development tools that come with Legs4Mule complements those that come with the regular LegStar product. This means you will need to install the LegStar core product in order to get the complete set of development time features.

If you elected to use the ant scripts, then similarly you will need the ant scripts that come with the regular LegStar. If you prefer the Eclipse plugins, then you can get all the necessary tools from the same location (see above).

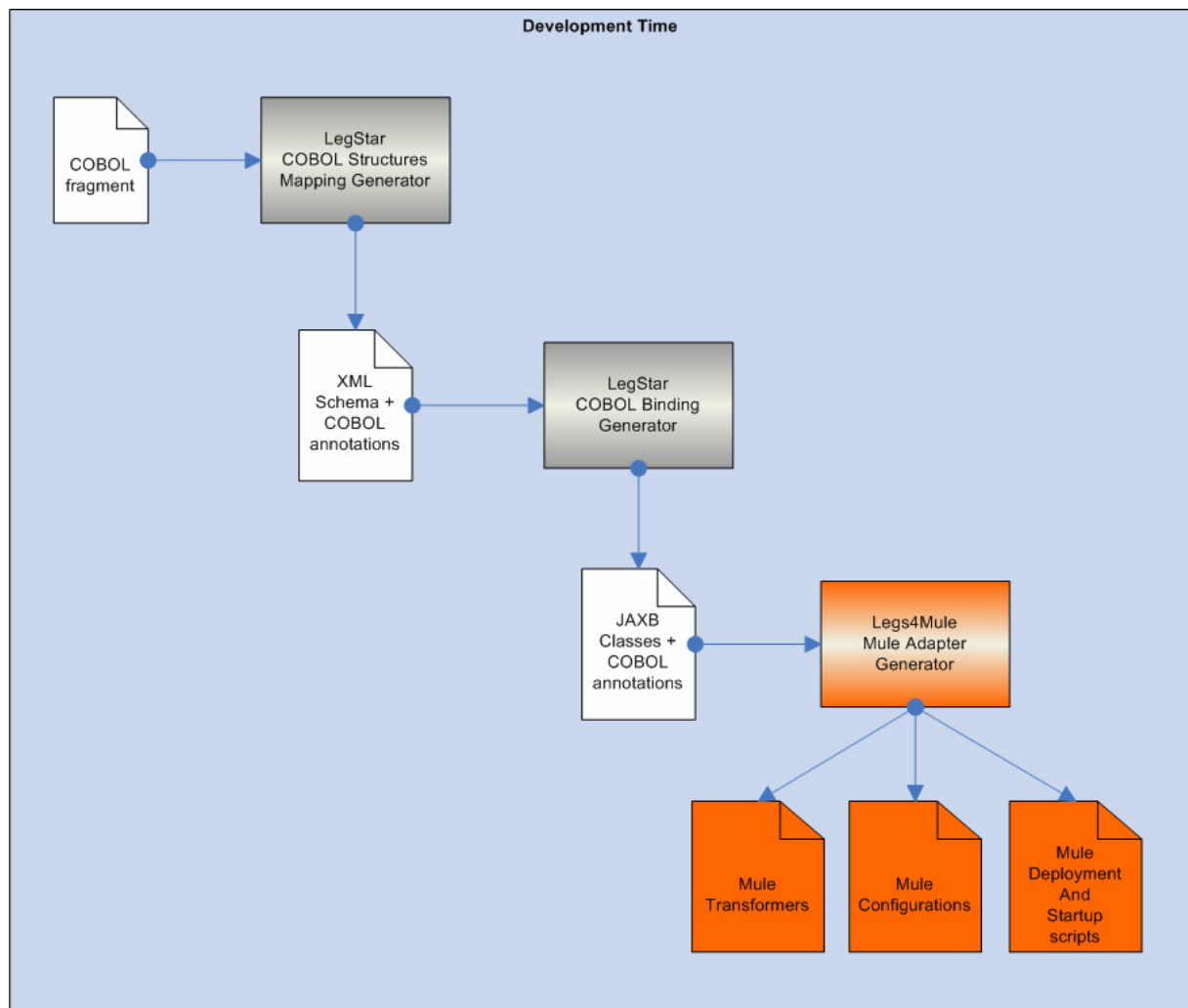
The rest of this document assumes you are using the Eclipse plugins. The README.txt file holds similar instructions on how to use ant scripts instead.

From a runtime perspective, the Legs4Mule distribution contains all the necessary Java-side components. The CICS runtime though is part of the regular LegStar distribution.

Expose a Mainframe program to Mule clients

In this use case, you want a mainframe program to be accessible by Mule clients.

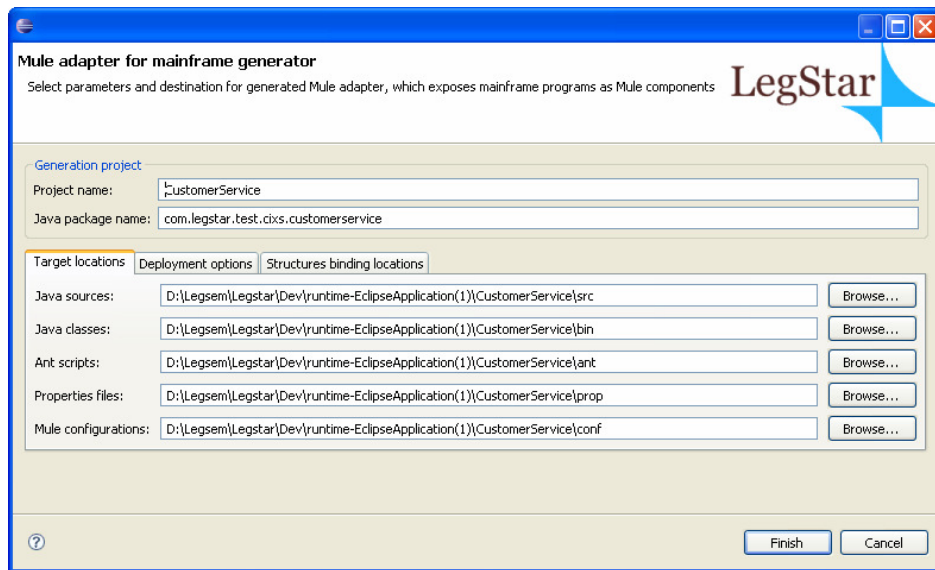
The following diagram shows the steps you will have to follow. Grey boxes represent development tools that are part of the core LegStar product and the orange boxes those that are part of Legs4Mule:



The LegStar COBOL Structures Mapping Generator and COBOL Binding Generator are described in the LegStar documentation <http://www.legsem.com/legstar/pdf/legstar-presentation.pdf>. Follow the steps in the “Expose a COBOL program as a Web Service” chapter until you get to generating a “Web Service adapter”. With the LegStar Mule Eclipse plugin installed, clicking on the generate button from the Operations Mapping editor displays a dialog similar to this:



As you can see, in addition the standard Web Service generator options, you can now select “Mule adapter for mainframe generator” which should display the generation dialog:



The generation process creates Mule Transformers in your Java sources folder. These are specialized transformers, that will process the COBOL structures that were bound in the previous steps.

Most of the options appearing here are derived from preferences, which you can change from Window→Preferences...→LegStar→Code generation.

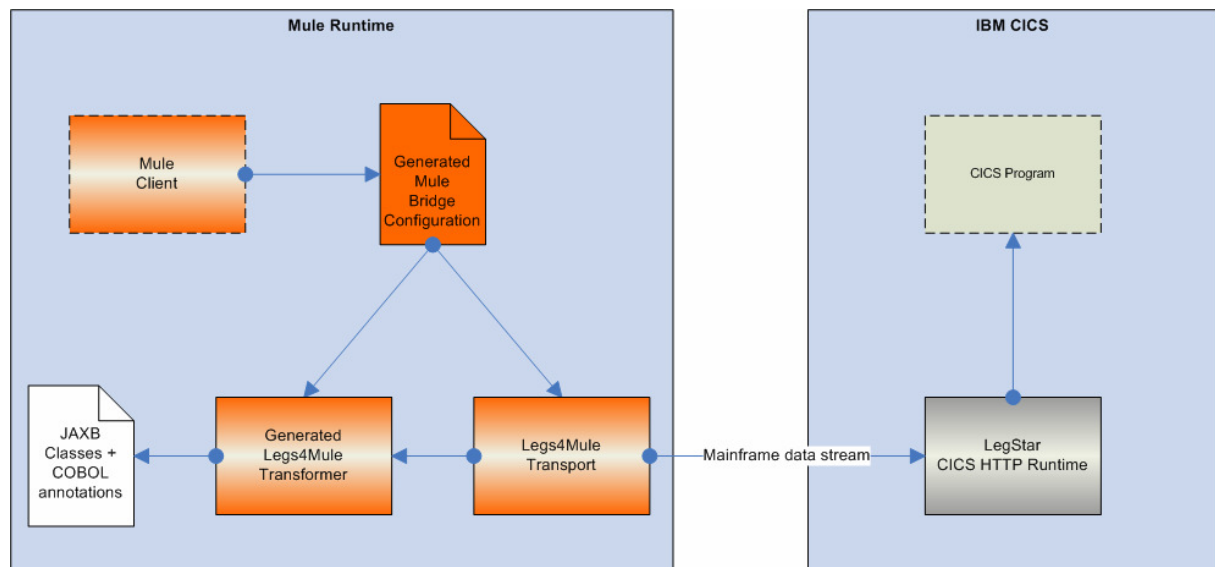
Clicking on the Finish button will generate ant scripts and Mule configuration files.

There are currently two different Mule configurations that are generated corresponding to two different deployment options:

Bridge configuration

This is the recommended configuration; it fully exploits the Legs4Mule transport. It uses the standard Mule BridgeComponent to expose the mainframe program to Mule Clients.

This diagram depicts how control flows at runtime between a Mule Client and a target CICS program (from left to right):



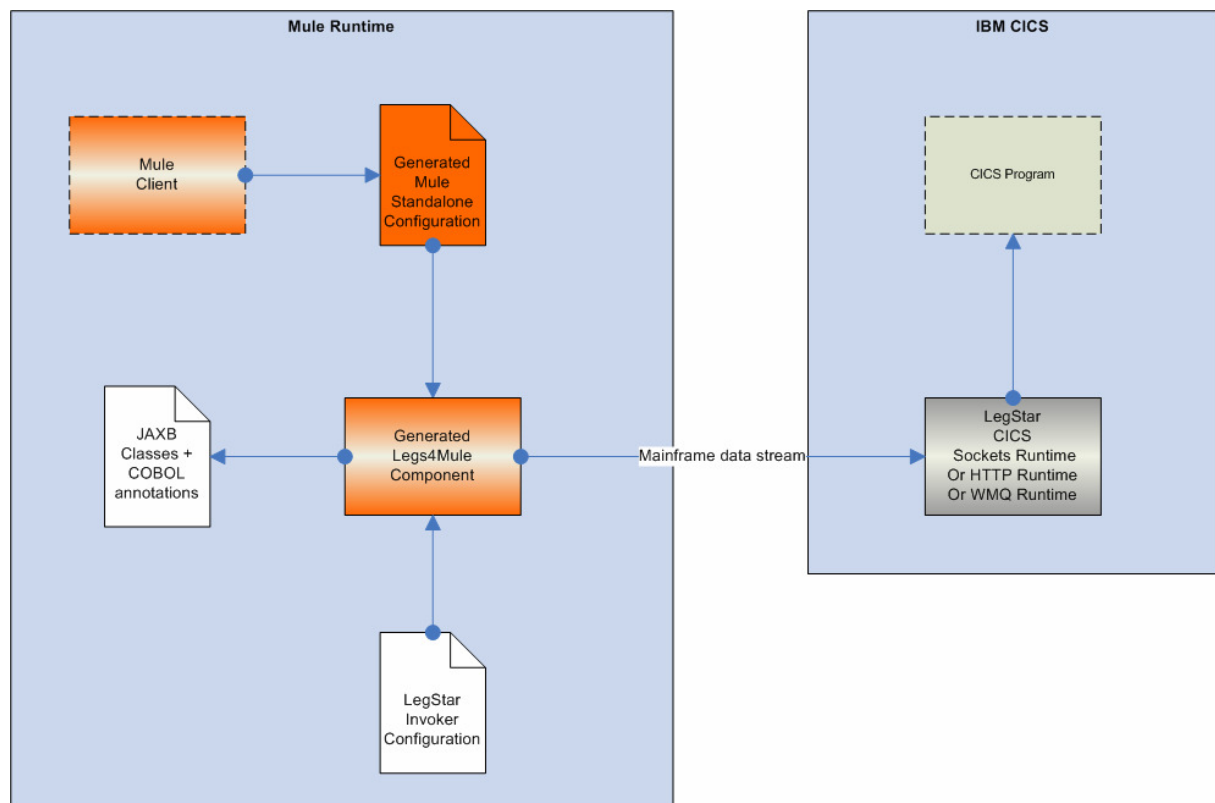
When you start Mule with this configuration, the underlying wire protocol is HTTP. There is no support yet for Sockets or Websphere MQ.

Standalone configuration

Because of the HTTP limitation, the standalone configuration is also provided if you would like to use Sockets or Websphere MQ as the underlying wire protocol to get to the mainframe program.

In addition to the Legs4Mule transformers, the Mule adapter generator also generates Java sources for a standalone Mule component, which internally uses the regular LegStar connectivity classes rather than the Legs4Mule transport. This means that Mule is not aware that this component is actually talking to a remote system.

In this configuration, the control flows like this between a Mule client and a CICS program:



There is more configuration work involved in this option but you can find details on how to setup this type of communication on the LegStar web site.

Deploy and run

As a convenience, the Legs4Mule adapter generator creates ant scripts that you can use to deploy and test your generated artefacts.

The build.xml script can be used to deploy the generated component to your Mule installation (Defined by the MULE_HOME environment variable).

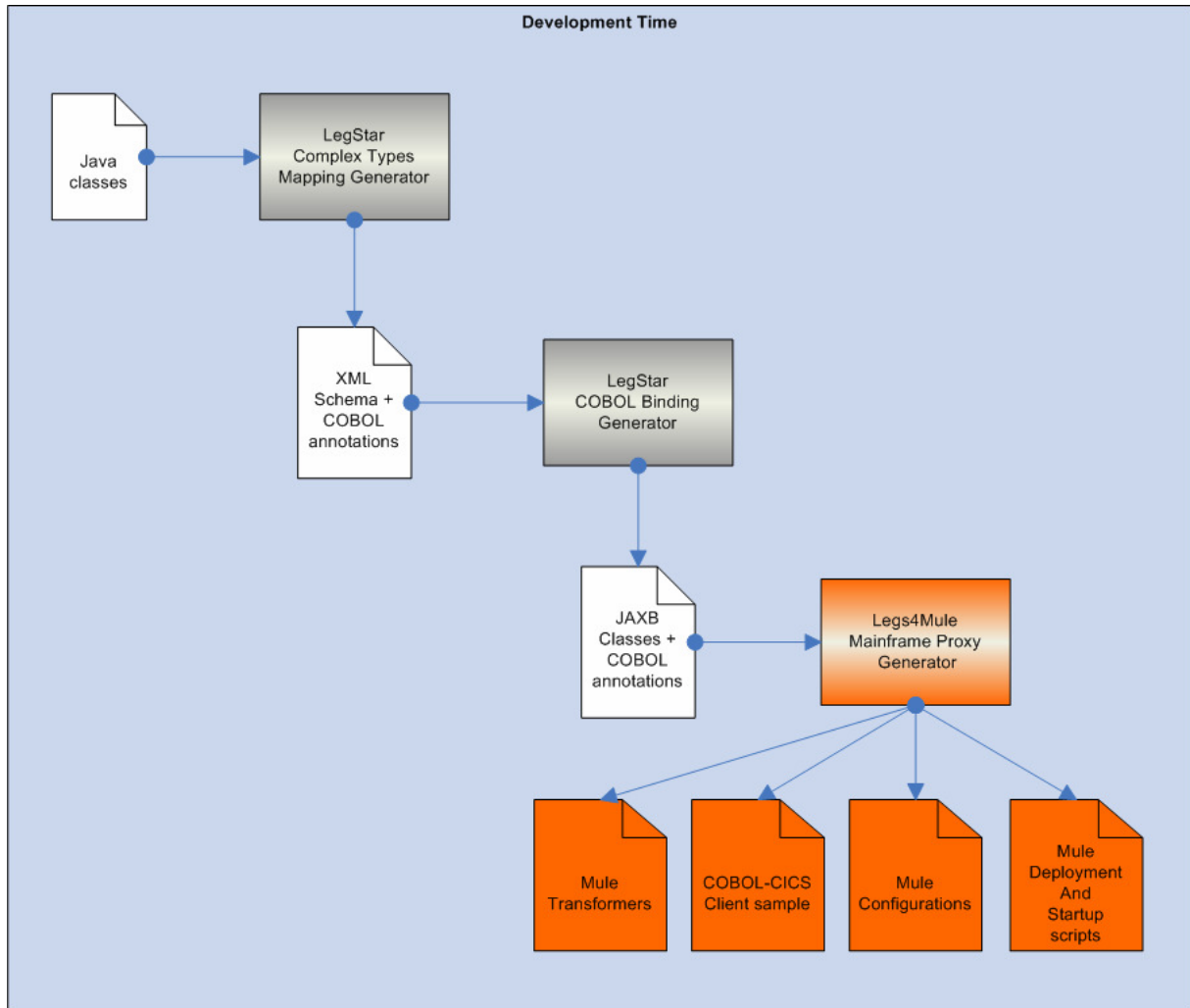
You can use the start-mule-x-config-y.xml scripts, where x is either bridge or standalone and y is your project name, to start a Mule instance on the target configuration. The generated script should have the classpath set so that all generated classes are made available to Mule.

Consume a Mule component from a Mainframe program

In this use case, we assume a Java object implementing the Transfer Object pattern is available to Mule. We want a CICS program to be able to call a method on the Java object passing a complex type as a request and receiving a complex type as the response.

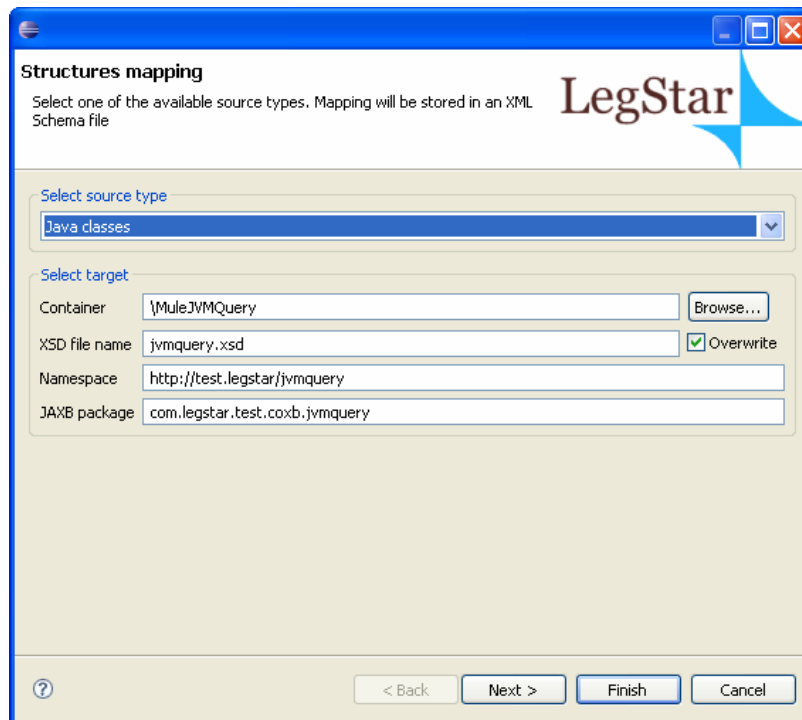
The target Java object must be available from the classpath. In Eclipse, this means you need to add a jar file to the referenced libraries or bring in the sources of the target Java object.

The following diagram shows the steps you will have to follow. Grey boxes represent development tools that are part of the core LegStar product and the orange boxes those that are part of Legs4Mule:

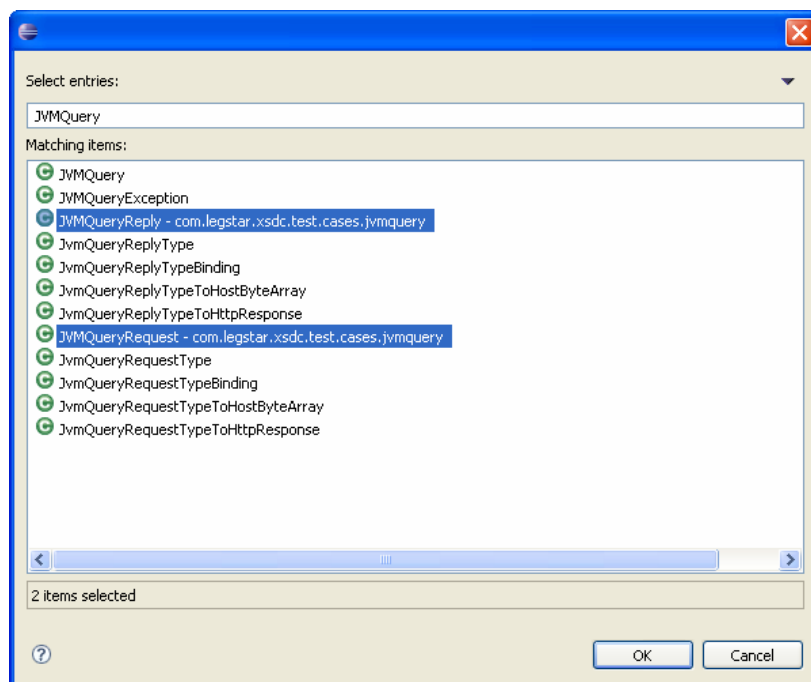


Complex Types Mapping:

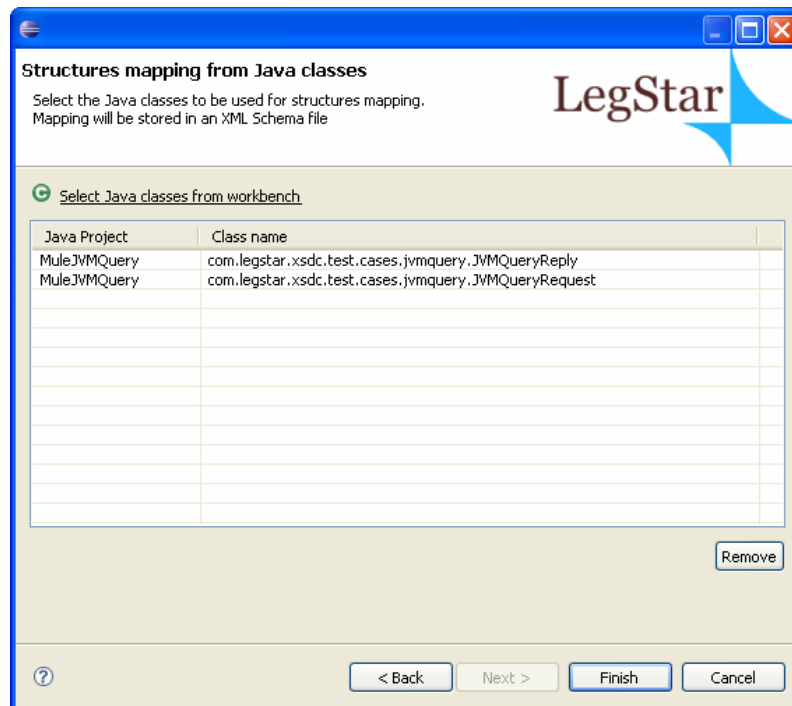
The first step in the process involves using the LegStar Structures Mapping plugin and selecting Java classes as the source type:



On the next page, you can select Java classes from the Eclipse project class path. This dialog allows you to key in the first characters of the target Java classes describing the complex request and reply. You should see a list of classes available from the classpath, which names start with the letter you typed. Next time around, the dialog remembers which classes you selected and the previously selected classes are listed:



After you selected the request and reply complex types, you are presented with the following dialog, which allows you to add more Java classes or remove some if needed:

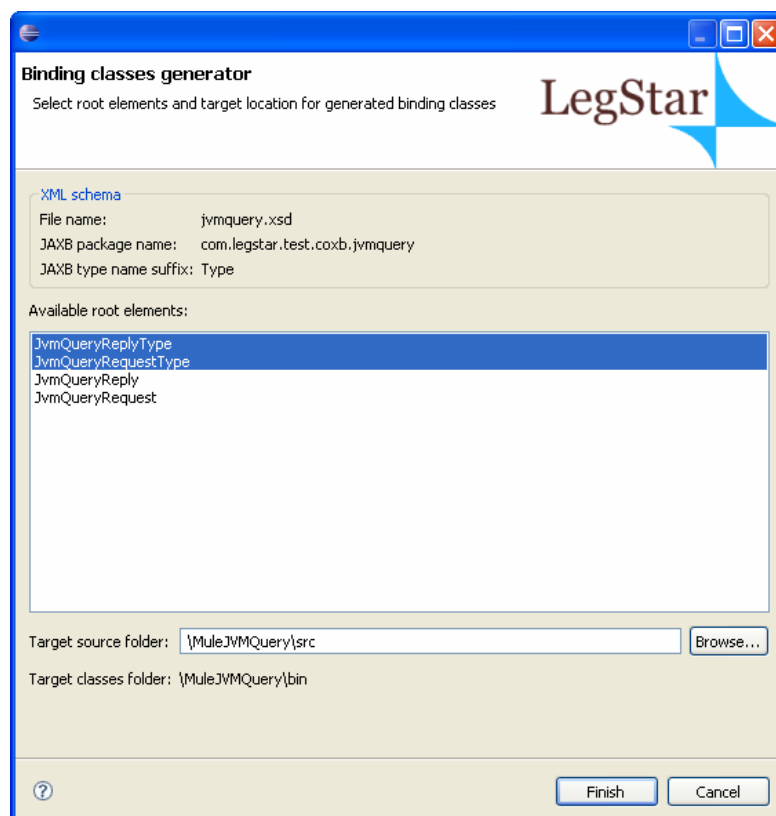


Finally, when you click on the Finish button, you are presented with the generated Mapping XML Schema with COBOL annotations that you can customize if necessary.

COBOL Binding classes generation:

The second step involves using the COBOL binding generator plugin by right clicking on the XML Schema file generated in the previous step.

The XML Schema contains elements and complex types for each Java class you previously selected. Make sure you select the complex types (should have a Type extension) here like so:

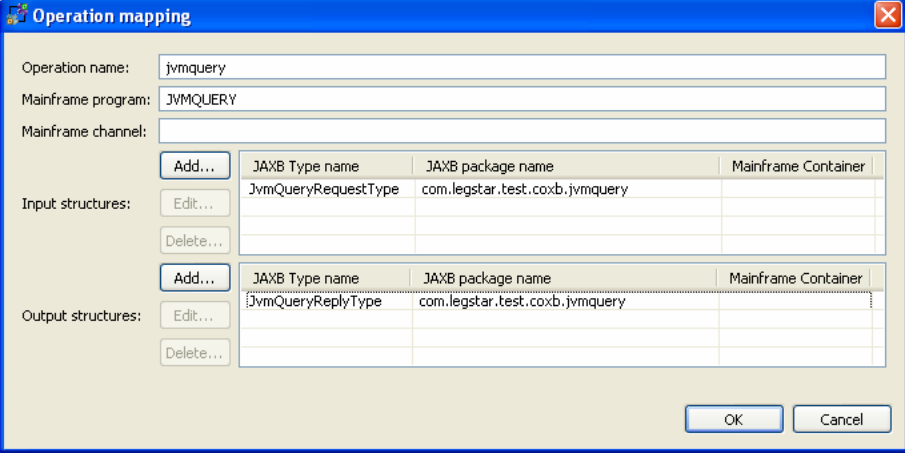


Clicking on Finish will generate the JAXB Classes with COBOL extensions.

Mainframe proxy generation:

When using the Eclipse plugins, before you can actually generate a Mainframe proxy, you first have to map the operations. With the ant scripts, this is done implicitly by customizing the ant script parameters.

This is how the new operations mapping dialog should look like:



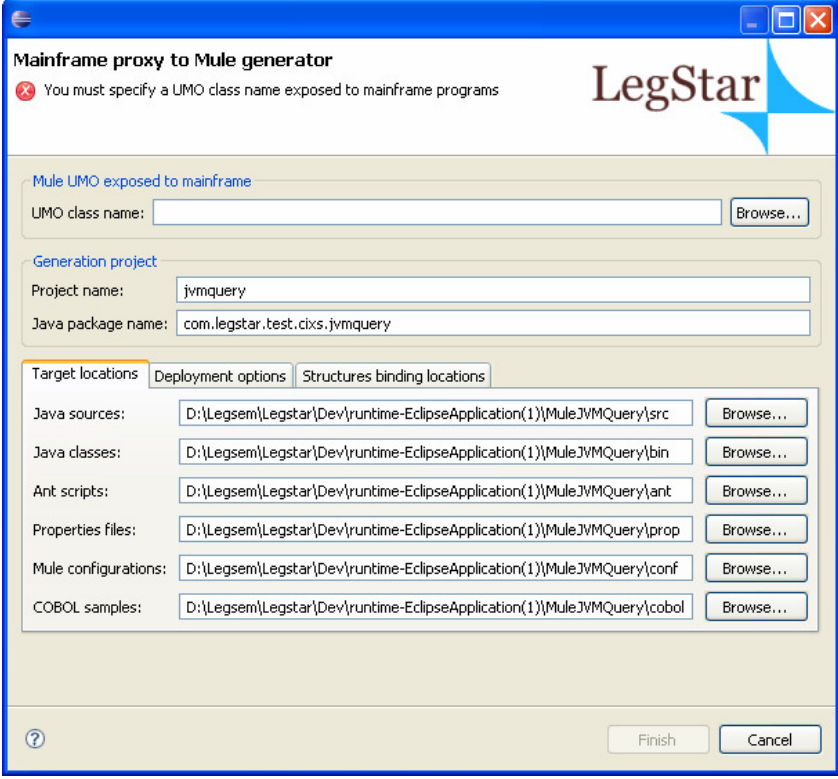
The "Operation mapping" dialog box is used to map JAXB types to Mainframe containers. It features a title bar with a close button. The main area is divided into two sections: "Input structures" and "Output structures". Each section has a table with columns for "JAXB Type name", "JAXB package name", and "Mainframe Container". The "Input structures" table contains one entry: "JvmQueryRequestType" from package "com.legstar.test.coxb.jvmquery". The "Output structures" table contains one entry: "JvmQueryReplyType" from package "com.legstar.test.coxb.jvmquery". Above each table are buttons for "Add...", "Edit...", and "Delete...". At the bottom right are "OK" and "Cancel" buttons.

Input structures:	JAXB Type name	JAXB package name	Mainframe Container
	JvmQueryRequestType	com.legstar.test.coxb.jvmquery	

Output structures:	JAXB Type name	JAXB package name	Mainframe Container
	JvmQueryReplyType	com.legstar.test.coxb.jvmquery	

Note that the mainframe program name is a sample that will be generated for you so keep the default value.

Now back to the operations mapping editor you can click on Generate and this time select the "Mainframe proxy to Mule generator". You will be presented with the following dialog:



The "Mainframe proxy to Mule generator" dialog box is used to configure the generation of a Mainframe proxy. It features a title bar with standard window controls. A message at the top states: "You must specify a UMO class name exposed to mainframe programs". Below this is a section for "Mule UMO exposed to mainframe" with a "UMO class name" field and a "Browse..." button. The "Generation project" section contains "Project name" (jvmquery) and "Java package name" (com.legstar.test.coxb.jvmquery) fields. The "Target locations" section has tabs for "Target locations", "Deployment options", and "Structures binding locations". Under "Target locations", there are fields for "Java sources", "Java classes", "Ant scripts", "Properties files", "Mule configurations", and "COBOL samples", each with a "Browse..." button. At the bottom are "Finish" and "Cancel" buttons.

Mainframe proxy to Mule generator

✖ You must specify a UMO class name exposed to mainframe programs

Mule UMO exposed to mainframe

UMO class name: Browse...

Generation project

Project name:

Java package name:

Target locations | Deployment options | Structures binding locations

Java sources: Browse...

Java classes: Browse...

Ant scripts: Browse...

Properties files: Browse...

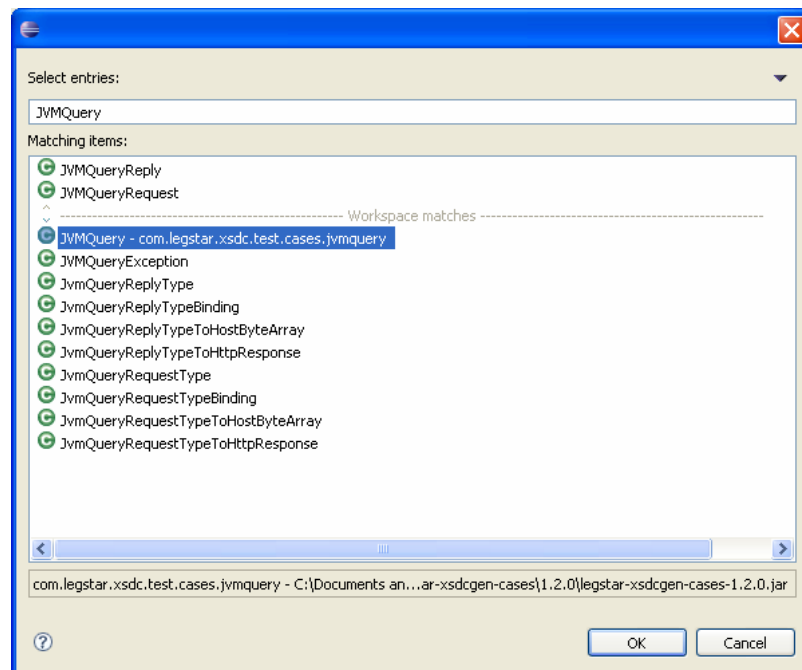
Mule configurations: Browse...

COBOL samples: Browse...

Finish Cancel

What you need to do here is to tell the generator what the target Java object class name is. That is, the Java class that implements the method taking as input the complex type mentioned and producing

the expected complex type response. Clicking on Browse allows you to pick up this class from the Eclipse classpath:

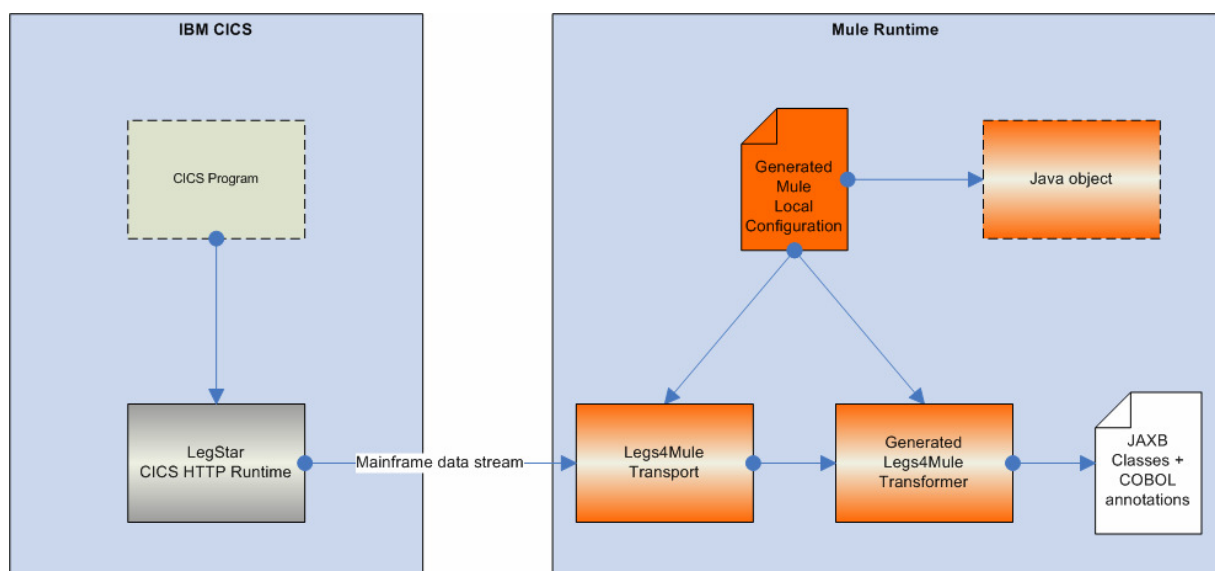


Once the target Java class is selected, you can click on the Finish button which will generate various Mule Transformers, Mule configurations, a sample COBOL-CICS client program and deployment ant scripts.

Local configuration

There is a single configuration file that exploits the Legs4Mule transport.

This is how control would flow from a CICS client program to the target Java object using this configuration (from left to right):



Deploy and run

As a convenience, the Legs4Mule adapter generator creates ant scripts that you can use to deploy and test your generated artefacts.

The build.xml script can be used to deploy the generated component to your Mule installation (Defined by the MULE_HOME environment variable).

You can use the start-mule-local-config-y.xml, scripts where y is your project name, to start a Mule instance on the target configuration.

The COBOL-CICS Sample program generated can be used as a template to write your own mainframe client programs.

Conclusion

Legs4Mule brings direct mainframe integration to Mule. There are no third-party proprietary layers involved. This is open-source from end to end.

The solution is bi-directional and allows mainframe programs to act as clients as well as servers. As the volume and criticality of Java applications increase in the enterprise, legacy mainframe programs need to interact in a peer-to-peer mode rather than client/server only.

The Legs4Mule transport implements a binary protocol with the mainframe, significantly reducing the overhead of the solution. This is in contrast with Web Services integration approaches, which imply heavyweight XML manipulations on both sides.

Finally, the COBOL binding framework brought in by LegStar gives you a lot of flexibility in mapping complex structures. The framework is customizable.

MuleSource offers Forums and Mailing lists that you are welcome to use. We will certainly appreciate your feedback.