LegSem
August 2007

LegStar

Presentation

I. Goals

LegStar provides development and runtime features for developers who need to integrate with z/OS legacy functions, such as COBOL-CICS programs.

With LegStar, existing legacy functions are exposed as Web Services or as POJOs (Plain Old Java Objects). Conversely, legacy functions can execute remote Web Services.

Unlike other solutions available, LegStar is open-source and relies on recent standards in three crucial areas:

- XML Schema as a universal meta-data language
- Annotation-based Language to Language mapping
- Web Services programming interfaces

LegStar leverages the ever-increasing power of open-source software by using familiar programming patterns (visitor, strategy…), frameworks (JAXB, JAXWS), tools (Apache Ant, Eclipse) and targeting wildly used J2EE runtimes such as Apache Tomcat.

By using standard Web Services APIs, LegStar will work with the most advanced Web Services stacks without having to provide a SOAP stack itself. Most other integration solutions have developed proprietary SOAP stacks they have trouble keeping current with the latest advances in Service oriented architectures (WS-Security, WS-ReliableMessaging, etc.)

Because there is less infrastructure to worry about, LegStar concentrates on the real issues facing integration developers (such as how to map COBOL weakly-typed variables to Java strongly-typed ones, deal with complex "REDEFINES" and variable size arrays or fully support CICS Containers).

At the core of LegStar is an XML schema to COBOL binding language. This is similar in spirit to the Java to XML Schema binding language introduced by the JAXB standard. This COBOL binding language materializes as XML schema annotations or Java annotations.

II. Use cases

The easiest way to present the LegStar architecture is to show how it supports two common integration use cases:

1. An existing legacy program, say a COBOL CICS program, needs to be exposed as a Web Service
2. A legacy application needs to execute a remote Web Service

The first use case is very common but the importance of the second one is growing rapidly as legacy sub-systems are being replaced by new applications running on J2EE and .Net platforms.

There is a large number of variations on these 2 main use cases, for instance developers might need to expose legacy functionalities as REST rather than plain Web Services, Developers might need to map complex structures to Java objects rather than XML. Developers might need to describe new structures in XML schema and then map these to Java and COBOL in support for two parallel developments (rather than integration), etc.

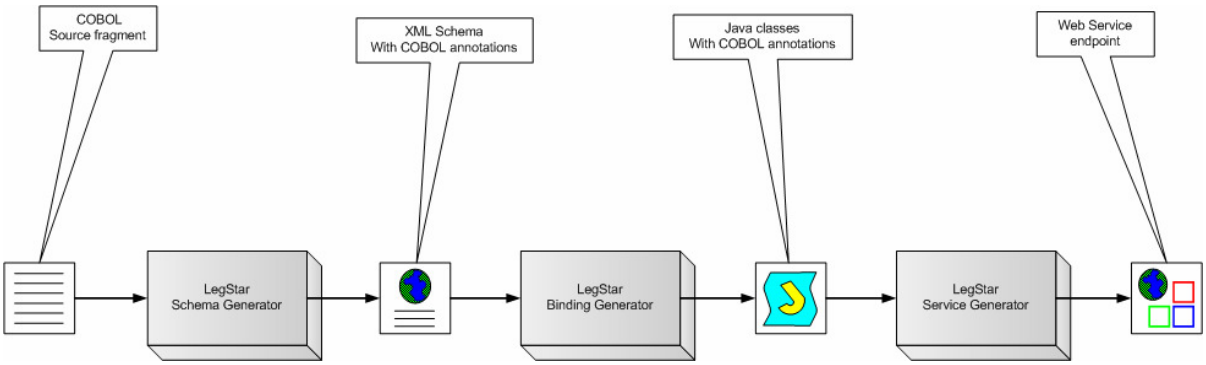
LegStar is modular so that features can be selected and combined as necessary.

III. Architecture

Expose a COBOL program as a Web Service

Development

Developers would follow these three steps to "Service-enable" a COBOL program:



In this use case, initial COBOL code fragments describe the legacy program input and output structures. When the COBOL program is a CICS Container-driven program there can actually be many such input structures and many output structures each described by a different code fragment.

The LegStar Schema Generator takes a COBOL fragment as input and creates an XML Schema with COBOL annotations. This step is repeated as necessary for each COBOL code fragment involved.

The generated XML Schema can be customized, and further annotated, by developers using standard XML Schema editors. In particular, developers can specify custom processing to deal with complex decisions related with COBOL REDEFINES for instance.

The LegStar binding generator takes a COBOL-annotated XML Schema as input and produces annotated Java Classes (using Java annotations introduced in J2SE 1.5). More precisely, LegStar uses the standard Java to XML Binding (JAXB) framework during this phase.

These generated classes are all that is needed for the runtime to perform XML to z/OS data and z/OS data to XML marshaling. z/OS data , described by the initial COBOL fragment, is typically encoded in a mainframe character set (EBCDIC). Conversion is totally bi-directional and completely independent of the origin or destination of the host data.

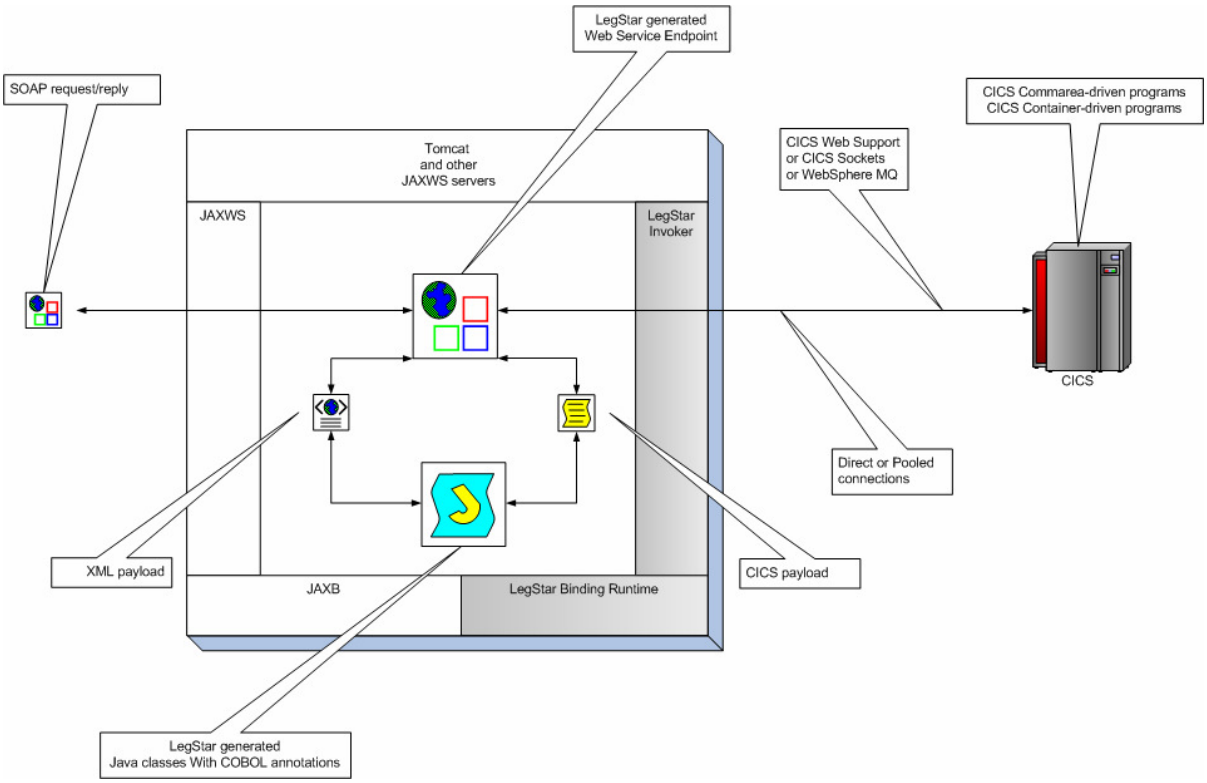
The LegStar Service Generator, maps a legacy program to a Web Service operation. The current version of LegStar supports CICS programs either Commarea or Container driven.

It is important to note that the tools behind each step are completely independent from each other. For instance, without an initial COBOL code fragment, developers could start from an XML Schema, edit the XML Schema (or use LegStar XML Schema Annotator) to add COBOL binding annotations and then continue the remaining steps. This is not an uncommon use case, where the legacy program is actually new and the starting point is an XML schema (An approach sometimes referred to as Contract-first).

For each of these steps, LegStar provides both Ant scripts and Eclipse plug-ins. Ant scripts are for pure java developers and plug-ins are for the Eclipse IDE.

Runtime

From a runtime perspective, this is how a request/reply message exchange pattern would flow in an IBM CICS environment:



SOAP request/replies are first processed by a standard Web Service stack. The LegStar-generated endpoint uses the JAXWS standard API to communicate with the SOAP stack.

The XML payload extracted from SOAP requests is handed over to the LegStar-generated Java classes which use the standard JAXB binding framework to parse the XML. The Java classes use the LegStar Binding runtime to convert data into the format expected by z/OS. This conversion includes Unicode to EBCDIC conversions, numeric conversions, REDEFINES decision-making, etc…

Once data is in z/OS format, the endpoint uses a transport independent layer called LegStar Invoker to invoke a remote CICS program. The actual transport is selected at runtime from a configuration file and offers the following options:

- Socket connectivity
- HTTP connectivity
- WebSphere MQ connectivity

Any of these transport options can be used in a direct or pooled fashion. Pooling of connections allows efficient connection reuse and enhances performances.

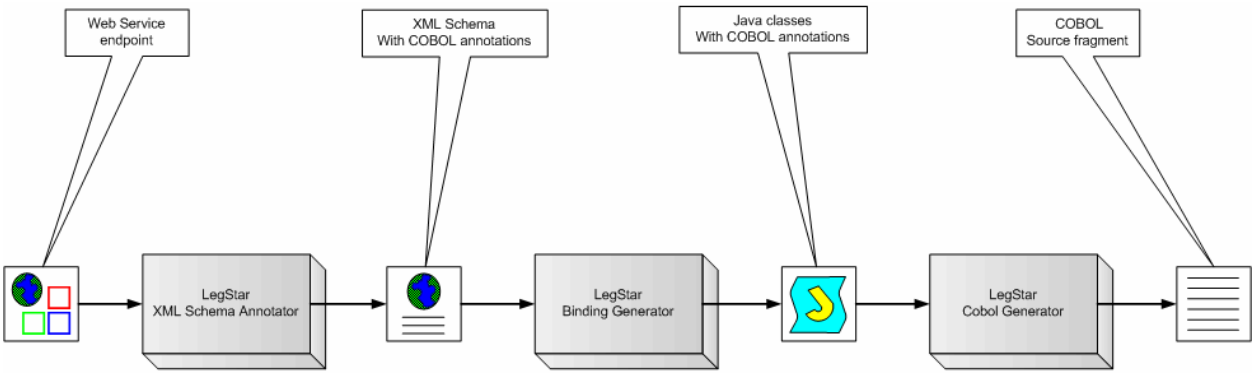
On the way back, z/OS data is converted to XML and then wrapped in a SOAP reply.

The CICS footprint of this architecture is minimal since all SOAP/XML processing occur off-host.

Call a remote Web Service from a COBOL program

Development

There is again a three steps process to achieve outbound access to Web Services:



The LegStar XML Schema annotator takes a WSDL or XML Schema file as input and produces a COBOL-annotated XML Schema.

The developer will typically edit the resulting XML schema to adjust such things as COBOL string sizes or maximum array sizes, which cannot always be inferred from XML Schema.

The second step, using the LegStar binding generator is the same as the one we went thru in the Web Service generation use-case. The result is a set of annotated Java classes, which provides the conversion capabilities from z/OS data to XML.

The third step, the LegStar COBOL Generator, produces COBOL code fragments from annotated Java classes. These code fragments contain structure definitions; they can be used as copybooks in regular COBOL programs. The structures describe the input parameters expected by the target Web Service as well as the reply message.

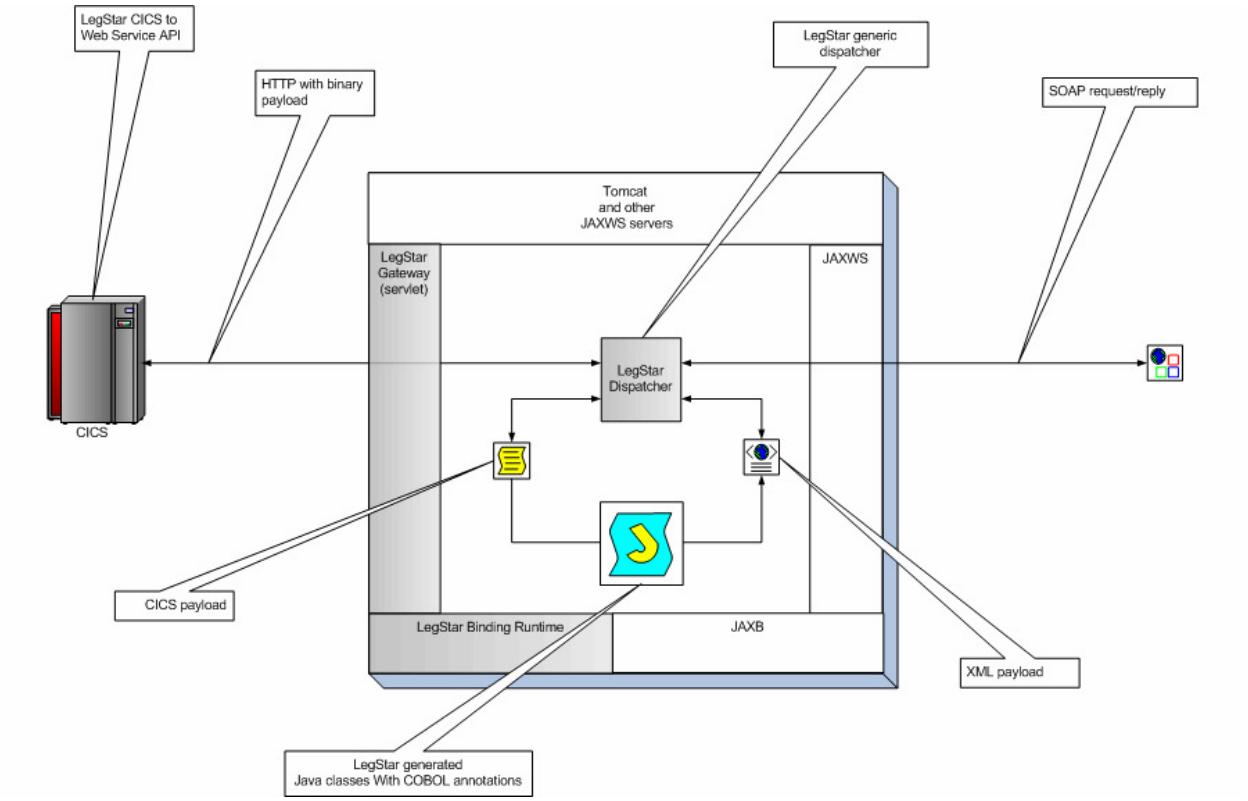
Runtime

LegStar provides a CICS to Web Service API that programmers can use from their COBOL programs to invoke the remote Web Service. They would typically use the generated COBOL descriptions to set the input parameters.

The CICS program does not directly call the target Web Service. Rather, a Servlet-based gateway receives the request, which is still in host (EBCDIC) format at this stage. Again, no conversion occurs on the host significantly reducing the z/OS footprint of this solution.

Conversion from z/OS format to XML is performed by the LegStar Binding runtime.

The request would flow as depicted in the following diagram:



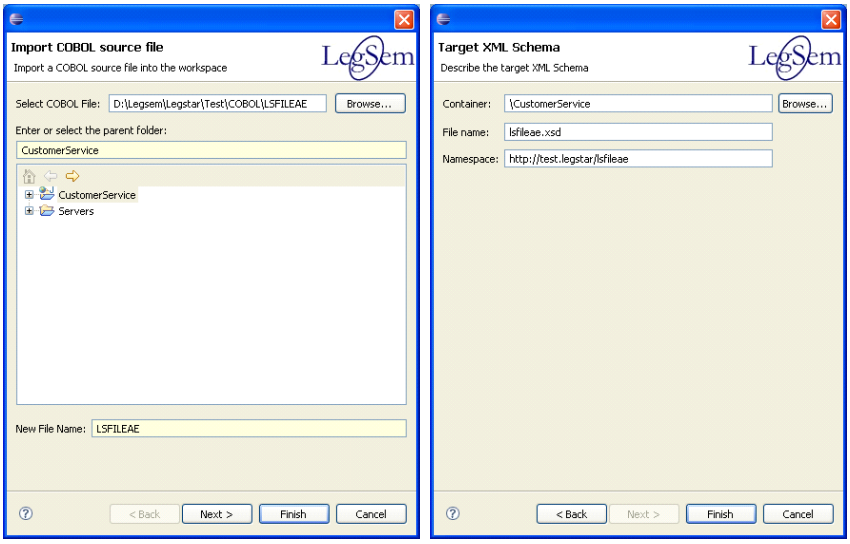
The LegStar dispatcher uses the standard JAXWS Client API to perform the call to the target Web Service. Again, use of a standard such as JAXWS shields LegStar from the ever-increasing complexity of SOA.

IV. Eclipse integration

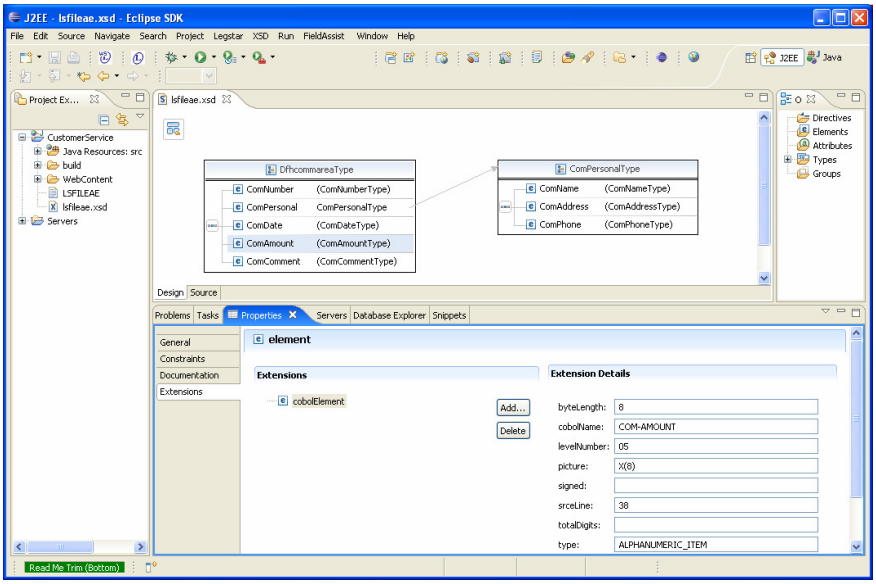
As already mentioned, the developers' tools are all available as Ant tasks. In addition, as a convenience for Eclipse developers, a set of plug-ins provide a user interface for some of the development steps:

XML Schema generation:

The process starts by importing some COBOL source into the Eclipse workspace. Then a few parameters are needed to describe the generated XML Schema:



The generated XML schema is editable with standard Eclipse editors:

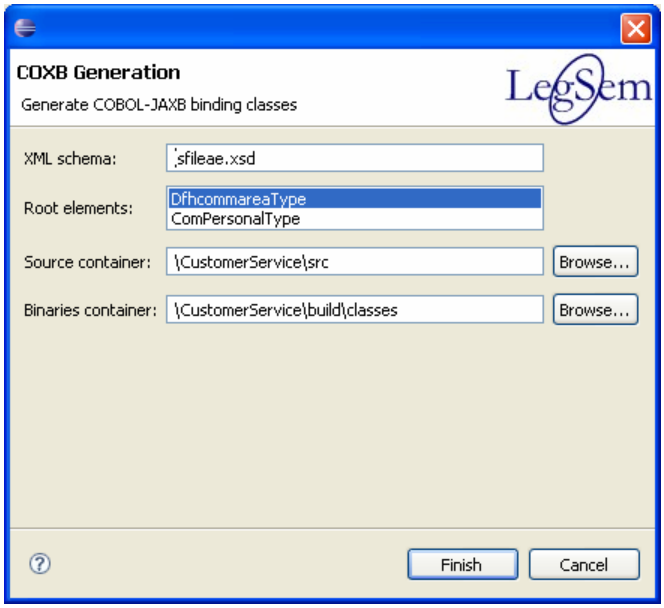


Observe the extensions used to annotate the XML Schema elements with COBOL meta-data.

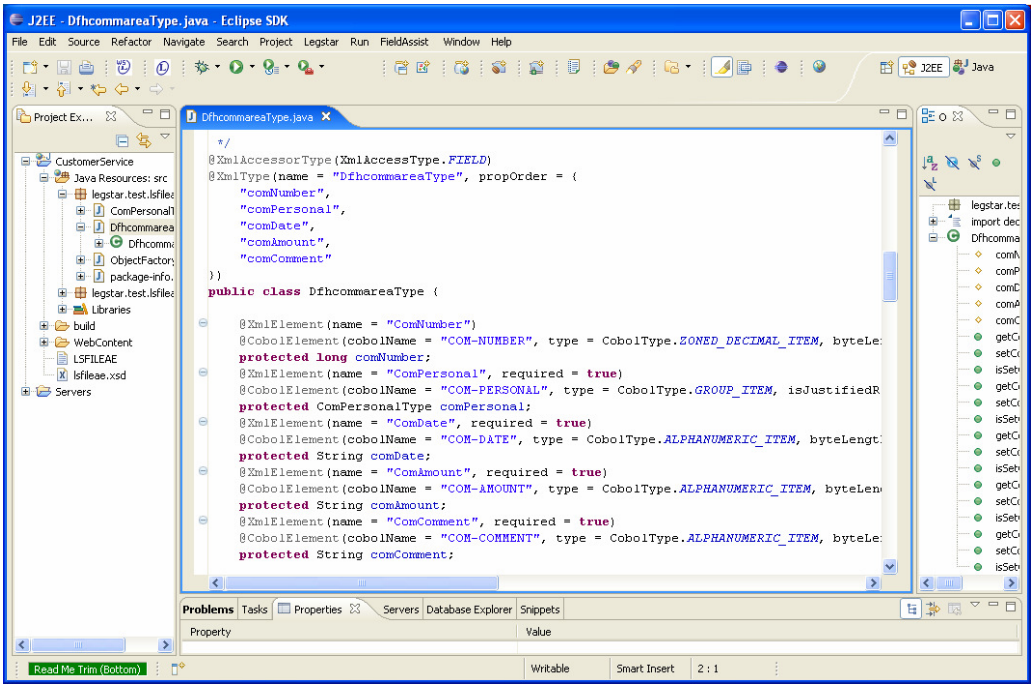
COBOL to XML binding generation:

The next step is to generate binding classes from the annotated XML Schema. These classes will be responsible for marshaling/un-marshaling XML into a z/OS data format as expected by CICS for instance.

The wizard is started by right clicking on a previously generated XML Schema:

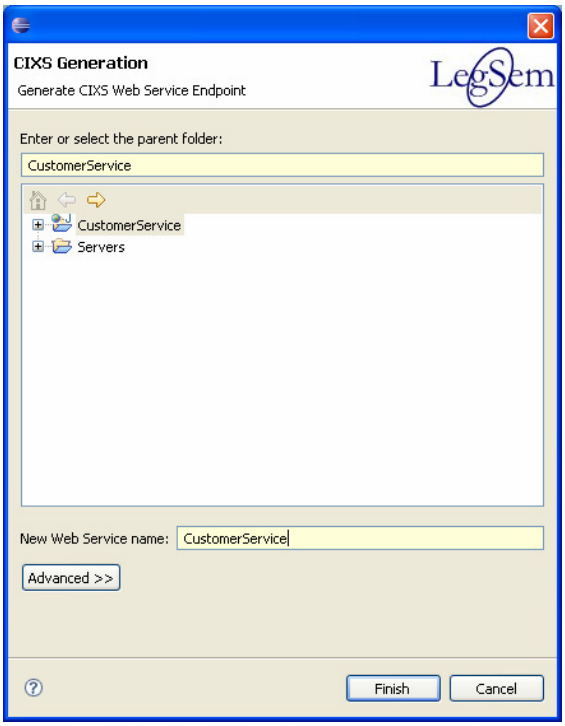


This dialog allows you to select multiple complex elements. Each complex element will be mapped to a standard JAXB class with additional COBOL annotations:

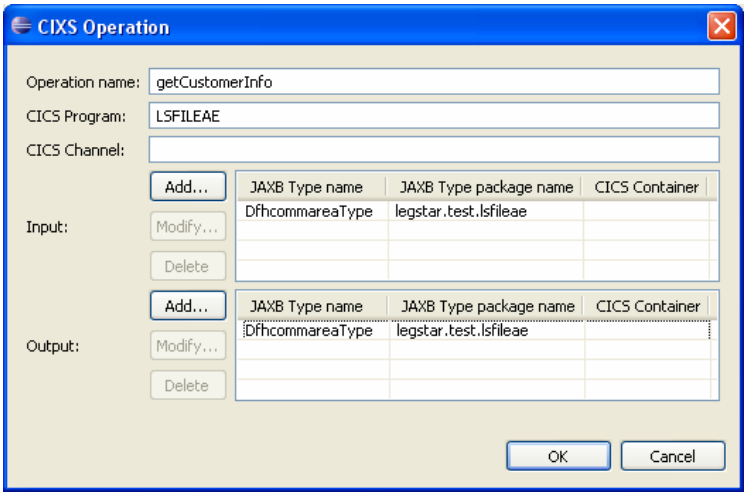


Web Service generation:

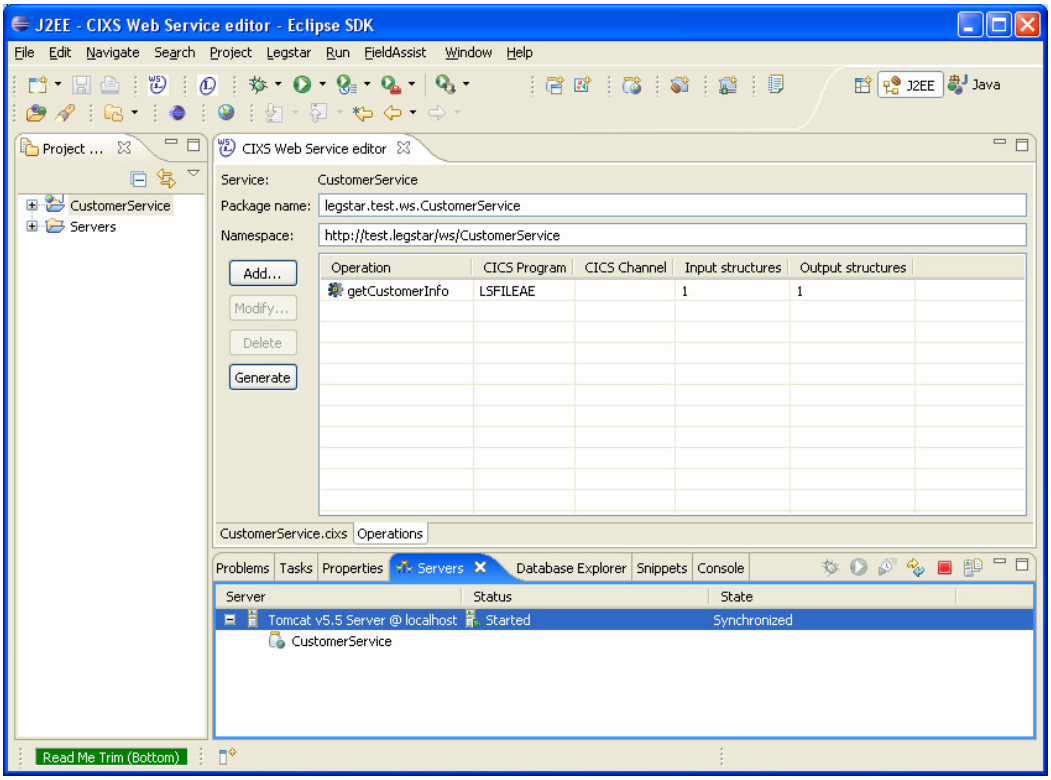
The final step in the process is to generate a JAXWS Endpoint where an operation maps to a remote CICS program.



After naming the Web Service endpoint, an editor allows you to create operations. Each operation corresponds to a CICS program:

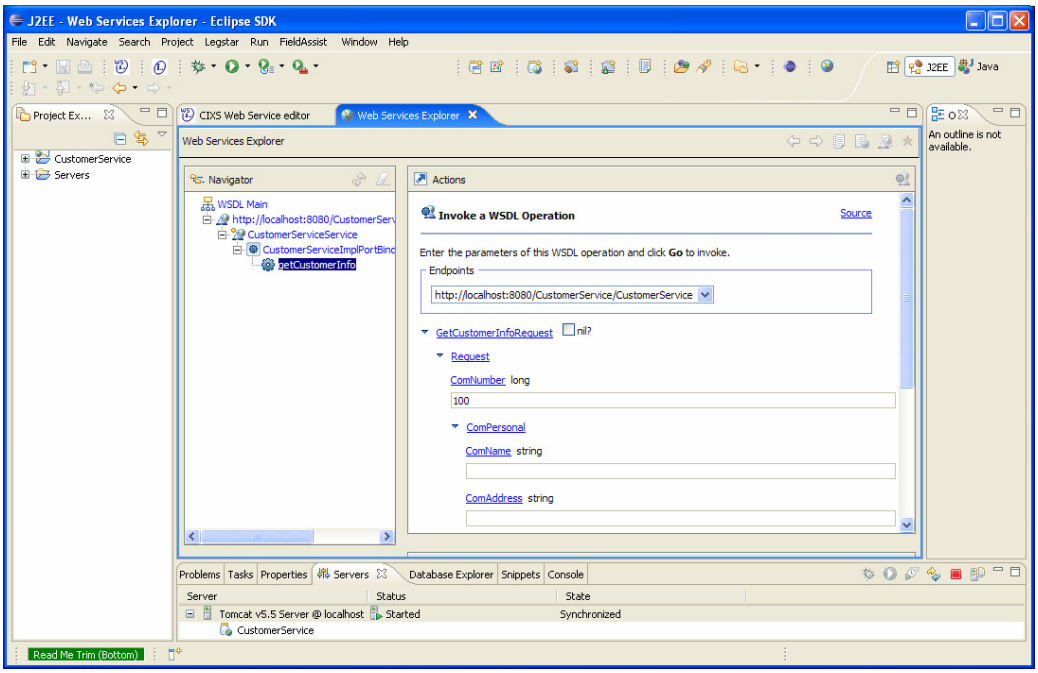


Input and output structures are selected from previously generated COBOL binding classes (Actually annotated JAXB types). You can select multiple input and output structures that can be mapped to CICS Containers.



The "Generate" button will create all the necessary artifacts for a Web Service Endpoint. In the context of Eclipse with WTP, you can directly drop the project onto an instance of a Server (in this case, Apache Tomcat):

Once the service is deployed, you can test it using the WTP Web Service Explorer:



V. Wrap up

LegStar is the first open-source initiative in the domain of legacy integration. It comes at a time where SOA and Web Services are maturing rapidly thanks to their wide adoption.

Sun, Apache foundation and Eclipse have fed the market with several, high quality, free, SOA frameworks and standards that keep evolving with the support of very large communities.

At the same time, the nature of legacy integration is changing. J2EE and .Net applications, which were only peripheral to the legacy applications, are now becoming more and more core to the IT infrastructure. This has triggered new integration needs where legacy applications now act as clients to J2EE applications for instance.

Open-source and the changing role of legacy applications have modified the landscape for integration solutions. We believe LegStar has the right architecture to meet the challenges of the future:

- By leveraging open-source and standards, LegStar can focus on bridging legacy applications with SOA as well as the next generation of architectures that will inevitably follow.
- With a high level of modularity, a large number of use cases can benefit from modules within LegStar.
- By being open-source itself, LegStar can attract Mainframe developers with a Java inclination as well as Java developers curious about legacy applications.

We also hope LegStar can inspire a standardization process among integration vendors, COBOL to XML binding does not have to be proprietary anymore than Java to XML binding. JAXB has become a standard through the Java Community Process, hopefully a similar process can be organized in the legacy integration world.