

DOCUMENTATION

Liferay Portal

[Liferay Portal 6.1](#)

[Liferay Portal 6.0](#)

[Liferay Portal 5.2](#)

[Liferay Portal 5.1](#)

Liferay Social Office

[Liferay Social Office 2.0](#)

[Liferay Social Office 1.5](#)

Additional Resources

[Whitepapers](#)

[Web Event Recordings](#)

Contact Us

[Request a Demo](#)

[Download EE Trial](#)

Documentation

Liferay provides a rich store of resources and knowledge to help our community better use and work with our technology.

[User Guide](#)

[Development](#)

[Community Resources](#)

Liferay Portal 6.1 - Development Guide

[Previous - Liferay Frameworks](#)

[Table of Contents »](#)

[Liferay Frameworks »](#)

[Service Builder](#)

[Next - Security and Permissions](#)

[Get Enterprise Pricing](#)

[Learn More](#)



Service Builder

Service Builder is a model-driven code generation tool built by Liferay to automate the creation of interfaces and classes for database persistence and a service layer. Service Builder will generate most of the common code needed to implement find, create, update, and delete operations on the database, allowing you to focus on the higher level aspects of service design.

The service layer generated by Service Builder, has an implementation class that is responsible for retrieving and storing data classes and adding the necessary business logic around them. This layer can optionally be composed of two layers, a local service and a remote service layer. The local service contains the business logic and accesses the persistence layer. It can be invoked by client code running in the same Java Virtual Machine. The remote service usually has additional code to check security and is meant to be accessible from anywhere over the Internet or your local network. Service Builder automatically generates the code necessary to allow access to the remote services using SOAP, JSON and Java RMI.

Define the Model

The first step in using Service Builder is to define your model classes and their attributes in a `service.xml` file. For convenience, we will define the service within the *my-greeting* portlet, although it should be placed inside a new portlet. Create a file named `service.xml` in `portlets/my-greeting-portlet/docroot/WEB-INF` within the Plugins SDK and add the following content:

```
<?xml version="1.0"?>
<!DOCTYPE service-builder PUBLIC
"-//Liferay//DTD Service Builder 6.1.0//EN"
"http://www.liferay.com/dtd/liferay-service-
builder_6_1_0.dtd">
<service-builder package-path="com.sample.portlet.library">
<namespace>Library</namespace>
```

```

<entity name="Book" local-service="true" remote-
service="true">

<!-- PK fields -->

<column name="bookId" type="long" primary="true" />

<!-- Group instance -->

<column name="groupId" type="long" />

<!-- Audit fields -->

<column name="companyId" type="long" />
<column name="userId" type="long" />
<column name="userName" type="String" />
<column name="createDate" type="Date" />
<column name="modifiedDate" type="Date" />

<!-- Other fields -->

<column name="title" type="String" />
</entity>
</service-builder>

```

Overview of *service.xml*

```
<service-builder package-path="com.sample.portlet.library">
```

This specifies the package path to which the class will be generated. In this example, classes will generate to WEB-INF/src/com/sample/portlet/library/

```
<namespace>Library</namespace>
```

The namespace element must be a unique namespace for this component. Table names will be prepended with this namespace.

```
<entity name="Book" local-service="true" remote-  
service="false">
```

The entity name is the database table you want to create.

```
<column name="title" type="String" />
```

Columns specified in `service.xml` will be created in the database with a data type appropriate to the specified Java type. Accessors in the model class will automatically be generated for these attributes.



Tip: Always consider adding two `long` fields called

groupId and *companyId* to your data models. These two fields will allow your portlet to support the multi-tenancy features of Liferay so that each organization (for each portal instance) can have its own independent data.

Generate the Service

Next, we'll build a service using our `service.xml`. We can do this by using either of the following methods: using *Liferay Developer Studio* or using the terminal window.

Using Developer Studio: From the *Package Explorer*, open your `service.xml` file found in your `my-greeting-portlet/docroot/WEB-INF` folder. By default, the file opens up in the *Service Builder Editor*. Make sure you are in *Overview* mode. Then, select *Build Services*.

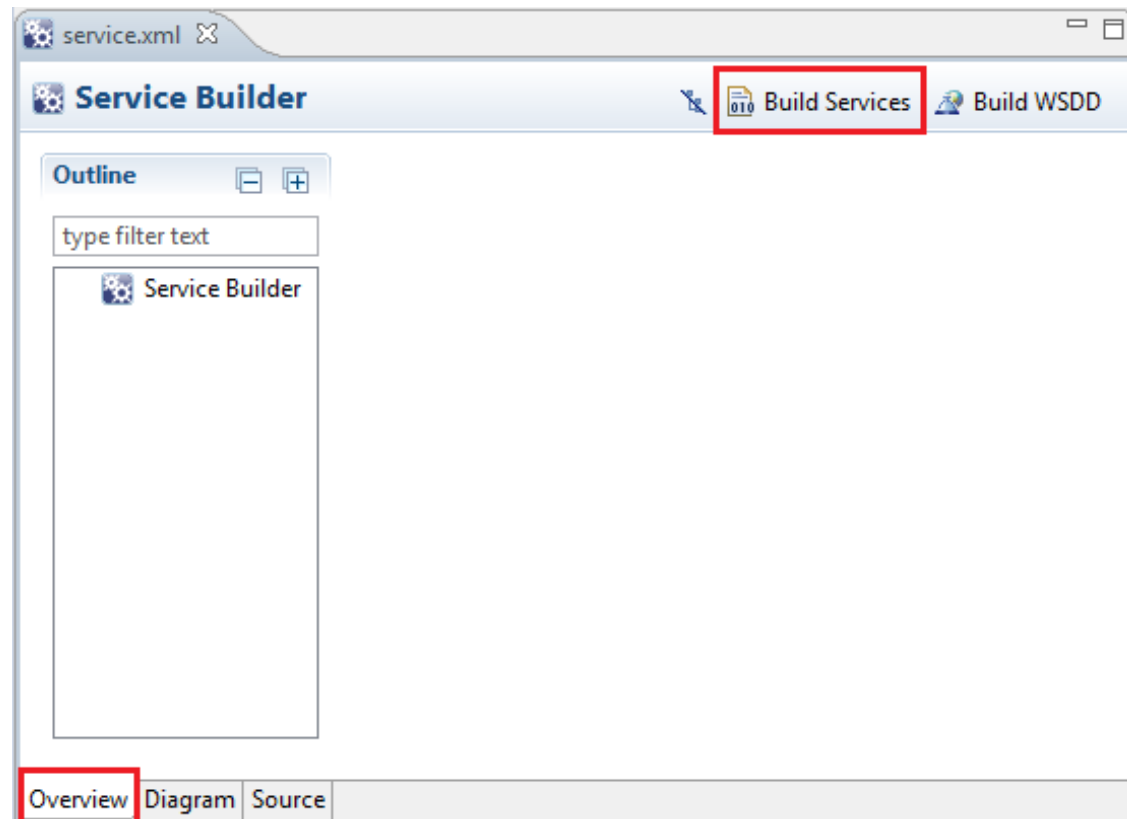


Figure 10.1: Overview mode in editor

You should receive a message in your console stating BUILD SUCCESSFUL along with a list of generated files. An overview of these files can be found later on in this section.

The *Overview* mode in Developer Studio's *Service Builder Editor* can be used to build services or build web service deployment descriptors (WSDOs). The editor also has *Diagram* and *Source* modes. If we select *Diagram*, we are given a graph structured background onto which we can add entities and

relationships from the *Palette* available on the right hand side of the editor. Here is a view of the editor in *Diagram* mode:

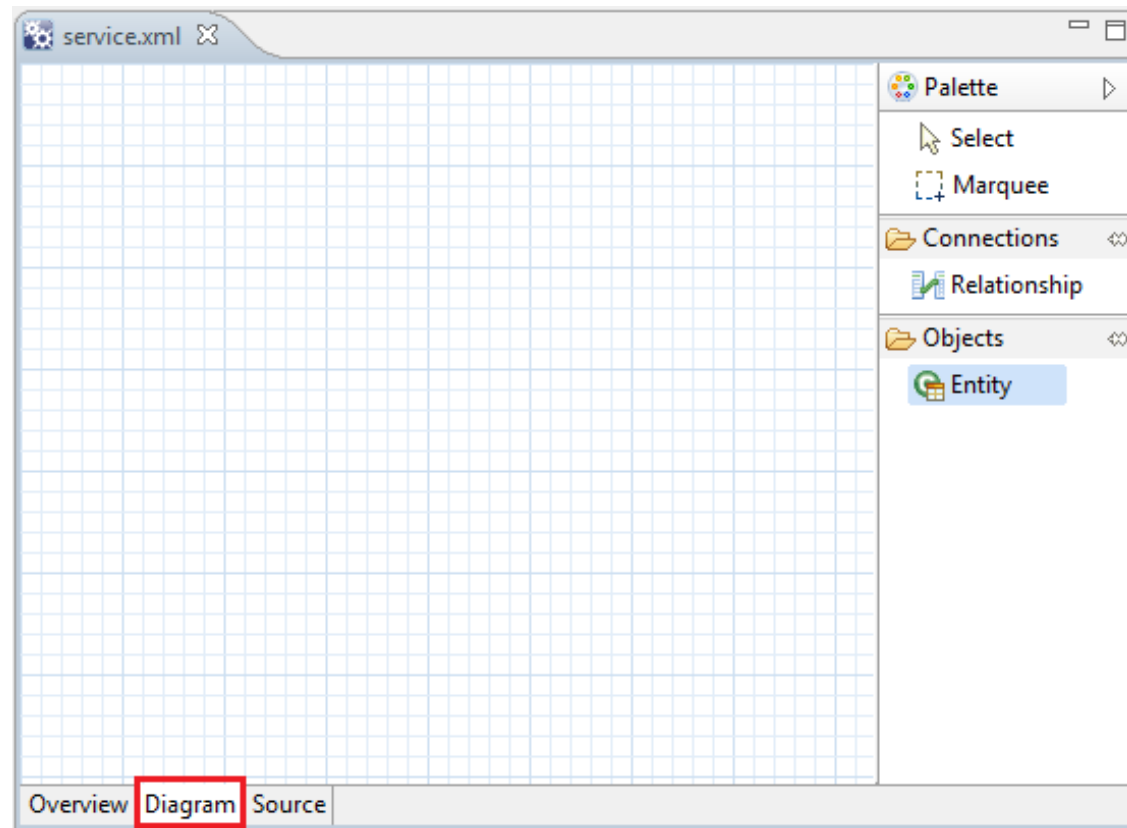


Figure 10.2: *Diagram* mode in editor

Lastly, select *Source* mode to edit the XML source directly. Here is an example `service.xml` shown in *Source* the editor's mode:



Figure 10.3: Source mode in editor

Developer Studio offers many options to help automate creating interfaces and classes for your database persistence and service layers.

Using the terminal: Open a terminal window in your `portlets/my-greeting-portlet` directory and enter this command:

```
ant build-service
```

The service has been generated successfully when you see BUILD

SUCCESSFUL in the terminal window, you should see that a large number of files have been generated. An overview of these files is provided below:

- Persistence

- BookPersistence - **book persistence interface** @generated
- BookPersistenceImpl - **book persistence** @generated
- BookUtil - **book persistence util**, instances BookPersistenceImpl @generated

- Local Service

- BookLocalServiceImpl - **local service implementation**. This is the only class within the local service that you will be able to modify manually. Your business logic will be here.
- BookLocalService - **local service interface** @generated
- BookLocalServiceBaseImpl - **local service base** @generated @abstract
- BookLocalServiceUtil - **local service util**, instances BookLocalServiceImpl @generated
- BookLocalServiceWrapper - **local service wrapper**, wraps BookLocalServiceImpl @generated

- Remote Service

- BookServiceImpl - **remote service implementation**. Put here the code that adds additional security checks and invokes the local service.
- BookService - **remote service interface** @generated
- BookServiceBaseImpl - **remote service base** @generated @abstract
- BookServiceUtil - **remote service util**, instances BookServiceImpl @generated

- BookServiceWrapper - **remote service wrapper**, wraps BookServiceImpl @generated
- BookServiceSoap - **soap remote service**, proxies BookServiceUtil @generated
- BookSoap - **soap book model**, similar to BookModelImpl, **does not implement Book** @generated
- BookServiceHttp - **http remote service**, proxies BookServiceUtil @generated
- **Model**
 - BookModel - **book base model interface** @generated
 - BookModelImpl - **book base model** @generated
 - Book - **book model interface** @generated
 - BookImpl - **book model implementation**. You can use this class to add additional methods to your model other than the auto-generated field getters and setters.
 - BookWrapper - **book wrapper**, wraps Book @generated

Out of all of these classes only three can be manually modified: BookLocalServiceImpl, BookServiceImpl and BookImpl.

Write the Local Service Class

In the file overview above, you will see that BookLocalService is the interface for the local service. It contains the signatures of every method in BookLocalServiceBaseImpl and BookLocalServiceImpl. BookLocalServiceBaseImpl contains a few automatically generated methods providing common functionality. Since this class is generated, you should never modify it, or your changes will be overwritten the next time you run Service Builder. Instead, all custom code should be placed in BookLocalServiceImpl.

Open the following file:

```
/docroot/WEB-INF/src/com/sample/portlet/library/service/impl/BookLocalServiceImpl.java
```

We will add the database interaction methods to this service layer class. Add the following method to the `BookLocalServiceImpl` class:

```
public Book addBook(long userId, String title)
throws PortalException, SystemException {
    User user = UserUtil.findByPrimaryKey(userId);
    Date now = new Date();
    long bookId =
        CounterLocalServiceUtil.increment(Book.class.getName());

    Book book = bookPersistence.create(bookId);

    book.setTitle(title);
    book.setCompanyId(user.getCompanyId());
    book.setUserId(user.getUserId());
    book.setUserName(user.getFullName());
    book.setCreateDate(now);
    book.setModifiedDate(now);
    book.setTitle(title);

    return bookPersistence.update(book, false);
}
```

Before you can use this new method, you must add its signature to the `BookLocalService` interface by running service builder again.

Using Developer Studio: As we did before, open your `service.xml` file and

make sure you are in the *Overview* mode. Then, select *Build Services*.

Using the terminal: Navigate to the root directory of your portlet in the terminal and run:

```
ant build-service
```

Service Builder looks through `BookLocalServiceImpl` and automatically copies the signatures of each method into the interface. You can now add a new book to the database by making the following call

```
BookLocalServiceUtil.addBook(userId, "A new title");
```

Overview of *service.properties*

Service Builder generates the properties file `service.properties` in the `src` directory of your service. Liferay Portal uses these properties to alter your service's database schema and load Spring configuration files to support deployment of your service. You should not modify this file, but rather make any necessary overrides in a `service-ext.properties` file in the `src` folder.

The only property that you may need to override from this file is `build.auto.upgrade`. Setting `build.auto.upgrade=false` in your `service-ext.properties` prevents Liferay from trying to automatically apply any changes to the database model when a new version of the plugin is deployed. This is needed in projects in which it is preferred to manually manage the changes to the database or in which the SQL schema has been modified manually after generation by Service Builder.

Built-In Liferay Services

In addition to the services you create using Service Builder, your portlets may also access a variety of services built into Liferay. These include the following:

- UserService
- OrganizationService
- GroupService
- CompanyService
- ImageService
- LayoutService
- PermissionService
- UserGroupService
- RoleService

For more information on these services, see Liferay's Javadocs at <http://docs.liferay.com/portal/6.1/javadocs/>.

[View »](#)

[Previous - Liferay Frameworks](#)

[Table of Contents »](#)

[Liferay Frameworks »](#)

Service Builder

[Next - Security and Permissions](#)


 NEWS

 BLOGS

 TWITTER

 FACEBOOK

 LINKEDIN

Powered by Liferay Portal 

[PRIVACY POLICY](#)

[SITEMAP](#)

[CONTACT US](#)

[ABOUT US](#)

[CAREERS](#)

© 2012 LIFERAY INC. ALL RIGHTS RESERVED