**Introduction**

With so many options available to developers today in the web services space, one of the questions worth asking is why should I use CXF? In no particular order, here are some of the reasons you might want to use CXF:

**JAX-WS Support**

CXF implements the JAX-WS APIs which make building web services easy. JAX-WS encompasses many different areas:

1. Generating WSDL from Java classes and generating Java classes from WSDL
2. Provider API which allows you to create simple messaging receiving server endpoints
3. Dispatch API which allows you to send raw XML messages to server endpoints

Much more...

**Spring Integration**

Spring is a first class citizen with Apache CXF. CXF supports the Spring 2.0 XML syntax, making it trivial to declare endpoints which are backed by Spring and inject clients into your application.

**Aegis Databinding**

Aegis Databinding (2.0.x) is our own databinding library that makes development of code-first web services incredibly easy. Unlike JAXB, you don't need annotations at all. It also works correctly with a variety of datatypes such as Lists, Maps, Dates, etc. right out of the box. If you're building a prototype web services that's really invaluable as it means you have to do very little work to get up and running (and one of the primary reasons XFire was started a while back)

**RESTful services**

CXF enables the development of RESTful services via annotations using the HTTP Binding. Using URI templates and annotations you can bind a service operation to arbitrary URL/verb combinations. For instance, you can annotate a getCustom method with @Get @HttpResource("/customers/{id}"). CXF will then listen for GET requests on that URL and using the parameter at the {id} location as a parameter to the service.

**WS-\* Support**

CXF supports a variety of web service specifications including WS-Addressing, WS-Policy, WS-ReliableMessaging and WS-Security.

**Apache Licensed**

CXF comes under the liberal Apache license, making CXF great for all types of applications.

**Developing a Service using JAX-WS**

You can develop a service using one of two approaches:

1. Start with a WSDL contract and generate Java objects to implement the service (WSDL First Development)
2. Start with a Java object and service enable it using annotations (Java First Development)

**2. Java First Development**

To create a service starting from Java you need to do the following:

1. Create a Service Endpoint Interface (SEI) that defines the methods you wish to expose as a service.

Tip

You can work directly from a Java class, but working from an interface is the recommended approach. Interfaces are better for sharing with the developers who will be responsible for developing the applications consuming your service. The interface is smaller and does not provide any of the service's implementation details.

1. Add the required annotations to your code.
2. Generate the WSDL contract for your service.

Tip

If you intend to use the SEI as the service's contract, it is not necessary to generate a WSDL contract

1. Publish the service.

**Creating the SEI**

The service endpoint interface (SEI) is the piece of Java code that is shared between a service and the consumers that make requests on it. When starting with a WSDL contract, the SEI is generated by the code generators. However, when starting from Java, it is the up to a developer to create the SEI.

There are two basic patterns for creating an SEI:

* Green field development   
  You are developing a new service from the ground up. When starting fresh, it is best to start by creating the SEI first. You can then distribute the SEI to any developers that are responsible for implementing the services and consumers that use the SEI.

|  |  |
| --- | --- |
| https://cwiki.apache.org/confluence/images/icons/emoticons/warning.gif | **Note** The recommended way to do green field service development is to start by creating a WSDL contract that defines the service and its interfaces. |

* Service enablement   
  In this pattern, you typically have an existing set of functionality that is implemented as a Java class and you want to service enable it. This means that you will need to do two things:
  1. Create an SEI that contains **only** the operations that are going to be exposed as part of the service.
  2. Modify the existing Java class so that it implements the SEI.

|  |  |
| --- | --- |
| https://cwiki.apache.org/confluence/images/icons/emoticons/warning.gif | **Note** You can add the JAX-WS annotations to a Java class, but that is not recommended |

REST (**Re**presentational **S**tate **T**ransfer) Architectural Style: