1. JAVA
   1. Axis2
      1. ADB data binding

**Overwiev**

ADB data binding framework is intended to be rather simple and lightweight. It provides means to compile XML schemas and WSDL files into java classes. Use of this tool is pretty straightforward, but working with generated files may be cumbersome.

**Using ADB data binding**

ADB data binding is generally available through the WSDL2Java tool, by specifying proper databinding option. WSDL2Java may be accessed in various ways:

* As stand-alone command line tool
* As Ant task
* As maven plugin

Usage of all of the above is similar. It requires specifying location of the WSDL and XML Schema files, output location for the generated sources and package name for generated classes. Unless told differently, ADB generates one big stub class per one web service definition (or, alternatively, one stub java class per one wsdl file). Parameters are passed as command line options when running from command line. In case of invoking via Ant or Maven tool they are specified in adequate XML file (build.xml or pom.xml).

For the detailed parameters specification, see:

<http://axis.apache.org/axis2/java/core/tools/CodegenToolReference.html> for command line and Ant

<http://axis.apache.org/axis2/java/core/tools/maven-plugins/maven-wsdl2code-plugin.html> for Maven plugin

**Execution requirements**

As Sabre Web Services definitions (provided in form of WSDL files) use corresponding types of data structures, proper XML Schema files must be provided in same location as the WSDL file. It includes common schemas for SOAP communication protocol (available as *Common Schemas* asset at <https://drc.sabre.com/> ) and request/response object pairs definition (available also at <https://drc.sabre.com/> as asset for each service).

**Generated code**

If all of required files are available for code generation tool, after invocation it will by default generate one stub java class per service. The file will be placed in proper folder structure, according to specified output package and directory. The file will be named *<serviceName>ServiceStub.java*.

The stub will contain static inner classes for each complex type. Any element or attribute encapsulated by this complex type will become a field in that class. Simple types restrictions are handled by replacing the relevant type with the basetype. Generated classes follow the bean convention, with *setAttribute* and *getAttribute* methods for object fields and *addElement(Element e)* or *setElements(Element [] e)* for arrays.

Generated files may be quite large (up to hundreds of thousands of lines, depending on service definition complexity ) and therefore problematic to work with, for example some IDE settings changes may be necessary.