XService XLinker Userguide
version 2.0

1 XService **XLinker 2.0 introduction**

1.1 XSERVICE XLinker introduction

XSERVICE XLinker 2.0 consists of Web Service Server、Web Service Client、Management console and Developmnet toolkits

1.2 XSERVICE XLinker components

Web Service Server

Web Service Server receives requests from the client and send the response back to the. Client. It also provides a graphic management tool to manage the server. The server can be embedded into Web server such as Tomcat, WebLogic, and WebSphere. The client application can intergrate the web servies into the web application through the webapplication API provided by the web service server.

This server includes the following features

- Support for SOAP 1.2 , Service Styles –RPC message, binding with SOAP over various protocols such as SMTP, FTP, message-oriented middleware , Datatypes decleard in the specification。
- Support for WSDL1.1, Automatic WSDL generation from deployed services, validation and Semantic analysis of the soap message based WSDL
- Provides a management console for monitor and control the system
- Flexibility- support for configurable process to conform to the web services varies
- Support for reusable components that implement common patterns of processing for your applications, or to distribute to partners.

Web Service Client

Web Service Client is program interface for client application . The client constructs a soap message, and send it request to a web service and get the response from the requested web service. Web Service Client is a set of Java API and conforms to JAX-RPC1.1 .

the invoke styles that The web service client supports for are as follow:

- Support for Muti-datatypes
- Support for RPC and Document style services
- Support for SOAP with Header
- Support for SOAP with attachment

Developmnet toolkits

XSERVICE XLinker also provides the web services Developmnet toolkits to implement the transform between Java class and WSDL。

◆ Java2WSDL

Java2wsdl is a command tool, the tool can build wsdl from the java class.there are 5 input parameters: four required and one optional.

The inpu parameters are as follows:

Parameter name	property	default
“WSDL filename”	required	
“Java classname”	required	
“Default NameSpace”	required	
“Unified Resource Identifier”	required	
Service invoke style	optional	Default is “rpc”

The command is like this:

java java2wsdl C:\temp\Echo.wsdl org.act.Echo http://act.org http://localhost:8080/

“C:\temp\Echo.wsdl”	the output WSDL file;
“org.act.Echo”	the java class;

“http://act.org”	Default NameSpace;
http://localhost:8080 Unified Resource Identifier，	“rpc” is the default service style。

The service style can be modified to be “document”, in that case, the command is like this: *java java2wsdl C:\temp\Echo.wsdl org.act.Echo http://act.org http://localhost:8080/ document*

- WSDL2Java

XService Linker WSDL2Java tool can build java class from wsdl. The tool can aotomaticly generate a sataic stub for the WSDL。

The command tool is used like this:

java org.act.soap.wsdl.wsdl2java.WSDL2Java [wsdlfilename] [outputDir]

for example:

java org.act.soap.wsdl.wsdl2java.WSDL2Java

D:\\SOAP\\src\\org\\act\\soap\\wsdl\\wsdl2java\\echo.wsdl

D:\\output_java_stub

this example generate the static Stub according for echo.wsdl and savethe stub in the director “D:\\output_java_stub”。

Another example is :

Java org.act.soap.wsdl.wsdl2java.WSDL2Java

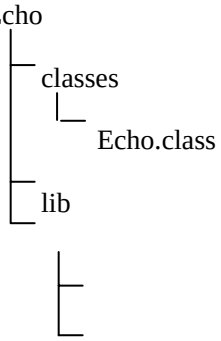
http://202.112.137.7:8080/axis/services/Echo?wsdl d:\\output

this example get the wsdl as input Stream from the internet and generate the static Stub andsave it in the directory “ D:\\output “。

- **WSAR automatic generation tool**

WSAR tool is a package **automatic generation tool, and the tool build a WSAR packagfor a java class . the structure of the package is as follow:**

Echo



META-INF

service.xml ((automatically generated)

echo.wsdl (automatically generated)

Managerment console

XSERVICE XLinker management console Is a graphic web application. The manager can mange the server and the deployed web services by the console.

The management console has the following features :

1. management the user profile
2. management the state of the server and cofigiration
   - query the state of the server engine
   - management the lifecycle of the the server engine(startup /stop)
   - query and config the property of the server engine
3. management the deployed Web services
   - get the list of deloyed web services
   - get the operation collection of a deployed web service
   - get the information about the service invoke
   - Security support subsystem

The Security support of the XLinker includes 3 parts :soao message transport security, different types authentificatons , RBAC (role-base access control) and the security for the server. The security system is to make sure the soap message midware is safe at run time .

Develop and invoke a webservice with XLinker

Web services is a web application wapper , and the web application can be a lot of things such as Java class、EJB、CORBA and COM。The logic process of the application is the

concern of the Web services runtime platform。 By now the XLinker only support for Web Services written in Java 。

The development of a web service provided by XLinker is to generate a WSDL and WSDD for a java application in forms of “servicename .wsdl” and “service.xml”and package them into a WSAR (a web service deployment package) 。The webservices invoke provided by XLinker is to create a web service client stub to invoke the service .

A simple example is given below to demonstrate how to develop and invoke the web service with XLinker., and function of the example is to output a simple string .

2.1 Develop a “ Echo” web service with Xlinker

The steps are as follow ：

- Create a Java source code file
- Compile the Java file
- Create a directory structure for a wsar file
- Generate a wsdl (**Web Service Descriptor**)
- Create a wsdd (**Web Service Deployment Descriptor**)
- Create a deployment packet
- Deploy the Web service

◆ Create a Java code file

Create a java file named “ Echo.java “ , and save it in the directory C:\. Of course, you can edit the java file with any editor. The code is as follow :

```
public class Echo {
    public String echoString (String s) {
        return s;
    }
}
```

◆ Complile the Java source code - “ *Echo.java* “

Use the command “javac ‘ complile the “ Echo.java “， and the “Echo.class” will be generated 。 the command is as follow

```
C:\>javac Echo.java
```

◆ Create the directories for the Websevice deployment package

Use the console commands to create the directories

C:\>md Echo	create “c:\ Echo “ directory
C:\>cd Echo	enter “c:\ Echo “ directory
C:\Echo>md classes	create “c:\ Echo \ classes” directory
C:\Echo>md lib	create “c:\ Echo \ lib” directory
C:\Echo>md META-INF	create “c:\ Echo \ META-INF” directory
C:\Echo>copy ../Echo.class classes	copy the compiled Echo.class into “c:\ Echo \ classes”

- WSDL2Java tool for building Java proxies and skeletons from WSDL documents
- Java2WSDL tool for building WSDL from Java classes.

◆ *building WSDL from the Java classes-“Echo.class”*

XSERVICE XLinker provides Java2WSDL tool for building WSDL from Java classes. You can build the wsdl with the tool and save it in the directory “c:\ Echo \ META-INF”. the command is :

```
C:\Echo> java org.act.soap.wsdl.java2wsdl.Java2WSDL META-INF\echo.wsdl Echo http://act.org/
http://localhost:8080/ACTSoap/Echo
```

The generated wsdl file is :

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://act.org/"
   xmlns:tns="http://act.org/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<wsdl:message name="echoStringRequest">
   <wsdl:part name="param0" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="echoStringResponse">
   <wsdl:part name="echoStringReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="Echo">
   <wsdl:operation name="echoString" parameterOrder="param0">
      <wsdl:input name="echoStringRequest" message="tns:echoStringRequest"/>
      <wsdl:output name="echoStringResponse" message="tns:echoStringResponse"/>
   </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="EchoSOAPBinding" type="tns:Echo">
   <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="echoString">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="echoStringRequest">
         <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://act.org/"/>
      </wsdl:input>
      <wsdl:output name="echoStringResponse">
         <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://act.org/"/>
      </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
<wsdl:service name="EchoService">
   <wsdl:port name="Echo" binding="tns:EchoSOAPBinding">
```

```
        <wsdlsoap:address location="http://localhost:8080/ACTSoap/Echo"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions><?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
        targetNamespace="http://act.org/" xmlns:tns=http://act.org/
         xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">
   <wsdl:message name="echoStringRequest">
      <wsdl:part name="param0" type="xsd:string"/>
   </wsdl:message>
   <wsdl:message name="echoStringResponse">
      <wsdl:part name="echoStringReturn" type="xsd:string"/>
   </wsdl:message>
   <wsdl:portType name="Echo">
      <wsdl:operation name="echoString" parameterOrder="param0">
         <wsdl:input name="echoStringRequest" message="tns:echoStringRequest"/>
         <wsdl:output name="echoStringResponse" message="tns:echoStringResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="EchoSOAPBinding" type="tns:Echo">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="echoString">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="echoStringRequest">
            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://act.org/"/>
```

```
        </wsdl:input>
        <wsdl:output name="echoStringResponse">
          <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://act.org/"/>
        </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="EchoService">
      <wsdl:port name="Echo" binding="tns:EchoSOAPBinding">
        <wsdlsoap:address location="http://localhost:8080/ACT XLinker/Echo"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

◆ *Building the Web service deployment descriptor*

"service.xml "-Web service deployment descriptor will be created and saved in the directory : C:/echo/META-IN ， the content is as follow ：

```
<Deployment     xmlns="http://act.buaa.edu.cn/soap/wsdd"
                xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
                xsi:schemaLocation="http://act.buaa.edu.cn/soap/wsdd/schema.xsd"
                ServiceName="Echo"
                WSDLLocation="echo.wsdl">
   <Documentation>ACT Sample Web Service</Documentation>
   <Transports>
      <Transport Type="http">
        <Parameter Name="URLPattern" Value="/Echo"/>
      </Transport>
   </Transports>
   <Provider Type="java">
      <Parameter Name="ClassName" Value="Echo"/>
```

```
  </Provider>
</Deployment>
```

The directory of the web service is as follows：

```
Echo
├── classes
│   └── Echo.class
├── lib
└── META-INF
    ├── service.xml
    └── echo.wsdl
```

◆ *Building the web sevice deployment package*

Use the command –“jar ”-below to package the service into a .wsar . additionally you can use the deployment tool provided by XSERVICE XLinker to deploy the service. now the development work has finished , and the .wsar can be deployed into Xlinker .

```
C:\Echo>jar cvf Echo.wsar .
```

WSAR 的自动生成工具的用法请参考“附录 A.3”。

◆ *Packaging and Deploying the Service*

1. copy the “Echo.wsar” into the web service deployment directory。The command is as follow :

```
C:\ Echo >copy Echo.wsar %XLinker%\deploy
```

The % XLinker % is the Xlinker Install path . the deployment package will automatically Be deployed if the Xlinker is running or next time when the xlinker starts up .

。

2. debug and run

You should now be able to access the service at the following URL :http://localhost:8080/ACTSoap/Echo?wsdl ,and will get the “Echo” servicesinformation；

2.2 invoke ”echo” service with XLinker

The user can invoke the “Echo service” by web service Client after the services are deployed correctly. Before invoked the examples, you'll need to make sure that your CLASSPATH includes

1. lib for XML parsing

   you can use the Apache Xerces published with XSERVICE XLinker。The lib includes these files：

   - jaxp-api-1.2.jar XML parsing interface
   - dom_2.jar W3C DOM interface
   - sax_2.jar SAX interaface
   - xercesImpl_2.2.1.jar Apache Xerces，XMLparser
   - wsdl4j-1_4.jar wsdl parsing
   - saaj-1_2-fr-api.jar parsing SOAP with attachment
   - jaxrpc-1_1-fr-spec-api.jar JAX-RPC specification interface
   - jax-1_1-fr-qname-class.jar Namespace
   - activation.jar
   - mail.jar

2. lib for XSERVICE SOAP client ：
   - act_xlinker.jar

Note: make sure that “echo.wsar” has been correctly deployed into XSERVICE XLinker (in the directory: %XLinker%\deploy)

2.2.1RPC based Web service invoke

2.2.1.1 create a java code named “SampleClient.java”

The code is as follow

```
1. import javax.xml.namespace.QName;
2. import javax.xml.rpc.Call;
3. import javax.xml.rpc.encoding.XMLType;
```

```
4.  import javax.xml.rpc.ParameterMode;
5.  import javax.xml.rpc.Service;
6.  import javax.xml.rpc.ServiceFactory;


7.  public class SampleClient {
8.  private final static String SERVICE_FACTORY_PROPERTY ="javax.xml.rpc.ServiceFactory";
9.  private final static String SERVICE_FACTORY_IMPL_ACT ="org.act.soap.client.ServiceFactory";
10. private final static String MESSAGE_FACTORY_PROPERTY ="javax.xml.soap.MessageFactory";
11. private final static String
12.     MESSAGE_FACTORY_IMPL_ACT ="org.act.soap.message.MessageFactoryImpl";
13. private final static String
14.     SOAP_CON_FACTORY_PROPERTY ="javax.xml.soap.SOAPConnectionFactory";
15. private final static String
16.     SOAP_CON_FACTORY_IMPL_ACT ="org.act.soap.message.transport.ConnectionFactoryImpl";


17. public static void main (String[] args) throws Exception {
18.         testEcho();
19. }



20. public static void testEcho () throws Exception {
21. // setProperty of Factory
22. System.setProperty(SERVICE_FACTORY_PROPERTY, SERVICE_FACTORY_IMPL_ACT);
23. System.setProperty(MESSAGE_FACTORY_PROPERTY, MESSAGE_FACTORY_IMPL_ACT);
24. System.setProperty(SOAP_CON_FACTORY_PROPERTY, SOAP_CON_FACTORY_IMPL_ACT);


25. Service service = ServiceFactory.newInstance().createService(new QName(""));


26. // construct a call    object .
27. Call call = service.createCall(new QName(""));
```

```
28.  call.setTargetEndpointAddress("http://localhost:8080/ACT XLinker/Echo");
29.  call.setOperationName(new QName("echoString"));
30.  call.setReturnType(XMLType.XSD_STRING);
31.  call.addParameter("arg1", XMLType.XSD_STRING, ParameterMode.IN);
32.  String strRtn = (String) call.invoke(new Object[] {new String("hello world!")});
33.  System.out.println("return String = " + strRtn);
34.  }
35.  }
```

2.2.1.2compile and run the “SampleClient.java”

Before running the “SampleClient.java”, make sure that XSERVICE XLinker has run correctly and “echo.wsar” has been deployed normally. After compile and run “SampleClient.java” successfully, you will see the result message “Hello world”. The result is below:

```
Hello world!
```

2.2.2 Message based Web services invoke

2.2.2.1 Create a java code named “SampleClient.java”

the code is as follow：

```
1.  import java.net.URL;
2.  import javax.xml.soap.MessageFactory;
3.  import javax.xml.soap.Name;
4.  import javax.xml.soap.SOAPBody;
5.  import javax.xml.soap.SOAPConnection;
6.  import javax.xml.soap.SOAPConnectionFactory;
```

```
7.  import javax.xml.soap.SOAPElement;
8.  import javax.xml.soap.SOAPMessage;
9.  import javax.xml.soap.SOAPEnvelope;


10. public class SampleClientM {
11. private final static String MESSAGE_FACTORY_PROPERTY = "javax.xml.soap.MessageFactory";
12. private final static String
13.         MESSAGE_FACTORY_IMPL_ACT ="org.act.soap.message.MessageFactoryImpl";
14. private final static String
15.         SOAP_CON_FACTORY_PROPERTY ="javax.xml.soap.SOAPConnectionFactory";
16. private final static String
17.         SOAP_CON_FACTORY_IMPL_ACT ="org.act.soap.message.transport.ConnectionFactoryImpl";


18. public static void main (String[] args) throws Exception {
19. testEcho();
20. }


21. public static void testEcho () throws Exception {
22. / se    Property    of    Factory
23. System.setProperty(MESSAGE_FACTORY_PROPERTY, MESSAGE_FACTORY_IMPL_ACT);
24. System.setProperty(SOAP_CON_FACTORY_PROPERTY, SOAP_CON_FACTORY_IMPL_ACT);
25. //construct a empy    SOAPMessage
26. SOAPMessage reqMsg = MessageFactory.newInstance().createMessage();
27. reqMsg.setProperty(SOAPMessage.WRITE_XML_DECLARATION, "true");
28. reqMsg.setProperty(SOAPMessage.CHARACTER_SET_ENCODING, "gb2312");
29. reqMsg.setProperty("org.act.soap.output-indent", "true");


30. //get the SOAPMessage Body
31. SOAPEnvelope reqEnv = reqMsg.getSOAPPart().getEnvelope();
32. SOAPBody body = reqEnv.getBody();
```

```
33.  / create    A    Name    object
34.  Name n1 = reqEnv.createName("echoString");
35.  //add a child to the Body element
36.  SOAPElement e1 = body.addBodyElement(n1);
37.  //add textnode to each    Body element
38.  e1.addTextNode("hello world!");

39.  SOAPConnection con = SOAPConnectionFactory.newInstance().
40.  createConnection();
41.  URL url = new URL("http://localhost:8080/ACT XLinker/Echo");
42.  SOAPMessage resMsg = con.call(reqMsg,url.toString());
43.  resMsg.writeTo(System.out);

44.  }
45.  }
```

2.2.2.2compile and run the “SampleClient.java”

Before compiling and running the “SampleClient.java”, make sure that XSERVICE XLinker has run correctly and “echo.wsar” has been deployed normally. After compile and run “SampleClient.java” successfully, you will see the result as follow:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><soapenv:Header/>
   <soapenv:Body>
      <echoString>
      hello world!
      </echoString>
   </soapenv:Body>
```

```
</soapenv:Envelope>
```

XSERVICE XLinker’ advanced features

XSERVICE XLinker is extensible , and can be integrated into a lot of web server such as tomcat WebLogic,SunOne， moreover, it supports for number of binding with SOAP over various protocols such as SMTP, FTP, message-oriented middleware 。