Brekeke PBX Web Service

User Guide

Brekeke Software, Inc.

Version
Brekeke PBX Web Service User Guide
Revised October 16, 2006

Disclaimer
Brekeke Software, Inc. reserves the right to change any information found in this document without any written notice to the user.

Table of Contents

1.0 Purpose 4
2.0 About Brekeke PBX Web Service 4
3.0 Installation 5
4.0 Configuring the Axis Soap Engine 5
5.0 Web Service Deployment Descriptor (WSDD) 5
5.1 Brekeke PBX WSDD 5
5.2 Scoped Services 5
6.0 Starting Web Service 5
7.0 Web Service Description Language 6
7.1 Obtaining WSDL from Deployed Service 6
7.2 Creating Web Service Clients and Proxy-Generation 6
7.3 Generating the Client Stubs from the WSDL 7
7.4 Issues with the Proxy Generation approach 7
8.0 Brekeke PBX Web Service Methods 8
8.1 getUserPropertyNames 8
8.2 getUserProperties 8
8.3 setUserProperties 9
8.4 getUserProperty 9
8.5 setUserProperty 10
8.6 createUser 11
8.7 createUser (with default password) 12
8.8 deleteUser 13
8.9 callControl 14
9.0 User Settings Properties 15
10.0 Sample Programs 16
10.1 ArrayGetterClient 16
10.2 ArraySetterClient 17
10.3 GetterClient 18
10.4 SetterClient 19
10.5 CreateUserClient 20
10.6 ThirdPartyCC 21
11.0 Web Service Security 22
12.0 Developing with PHP and other languages 22

1.0 Purpose

This document describes the Brekeke PBX Web Service. The PBX Web Service allows third party applications to view and modify PBX options and user settings.

Currently, the Web Service only has methods to view/modify user settings, create/delete users, and implement click-to-call feature.

2.0 About Brekeke PBX Web Service

The Brekeke PBX Web Service was implemented using Axis Version 1.3. Axis is essentially a *SOAP engine* -- a framework for constructing SOAP processors such as clients, servers, gateways, etc. Axis also includes:

- A server which plugs into servlet engines such as Tomcat,
- Support for the *Web Service Description Language (WSDL)*,
- An emitter tooling that generates Java classes from WSDL.

Please view the latest Axis documentation at:

http://ws.apache.org/axis/java/index.html

3.0 Installation

The files for the Axis Engine and the Brekeke Web Service are automatically installed when the Brekeke PBX software is installed.

4.0 Configuring the Axis Soap Engine

The web.xml file in the /pbx/WEB-INF subdirectory contains information necessary to deploy the Brekeke PBX servlet and the AXIS servlet.

5.0 Web Service Deployment Descriptor (WSDD)

A deployment descriptor contains configuration information for Web Services available through the Axis engine.

5.1 Brekeke PBX WSDD

For the Brekeke PBX Web Service, the WSDD file is named “server-config.wsdd” which is located in the /pbx/WEB-INF subdirectory.

The Brekeke PBX Web Service is named “UserImpl” in the WSDD file.

5.2 Scoped Services

Axis supports scoping service objects three ways. "Request" scope will create a new object each time a SOAP request comes in for the web service. "Application" scope will create a singleton shared object to service **all** requests. "Session" scope will create a new object for each session-enabled client who accesses the web service. To specify the scope option, open the WSDD file for editing and change the scope parameter in the “UserImpl” service as follows:

```
<service name=" UserImpl"...>
  <parameter name="scope" value="value"/>
  ...
</service>
```

Valid entries for "*value*" are “request”, “session”, or “application”.

6.0 Starting Web Service

The web service automatically starts when Tomcat is started.

7.0 Web Service Description Language

WSDL describes Web Services in a structured way. A WSDL describes, in a machine-understandable way, the interface to the web service, the data types it uses, and where the web service is located.

7.1 Obtaining WSDL from Deployed Service

When you make a service available using Axis, there is typically a unique URL associated with that service. For the Brekeke PBX Web Service, this is usually:

http://<host>/pbx/services/UserImpl

where <host> is the host server name or host ip address.

Attach a "?wsdl" to the end of the URL, Axis will automatically generate a service description for the deployed service, and return it as XML in your browser. Type:

http://<host>/pbx/services/UserImpl?wsdl

where <host> is the host server name or host ip address.

The resulting xml description may be saved or used as input to proxy-generation (Section 7.3).

7.2 Creating Web Service Clients and Proxy-Generation

There are several ways to create clients for the web service. One way is to have Axis generate a wrapper class for the web service. This is done by taking the WSDL description of the service and generating Java classes that make the low level calls appropriate to building the SOAP requests for each operation, then post-processing the results into the declared return values. Axis also takes note of any URL of the service included in the WSDL and compiles this in to the classes. Thus the client will automatically bind to the URL that the WSDL talks about -which is often the URL of the (development) server that the WSDL was retrieved from.

The steps to create a web service client using proxy generation are:

1. Start the web service by running Tomcat.
2. Generate the WSDL file using the browser. See Section 7.1.
3. Create the client stubs using the WSDL2Java tool. See Section 7.3
4. Compile the generated stub classes.
5. Write the client using the stub classes.

7.3 Generating the Client Stubs from the WSDL

You'll find the Axis WSDL-to-Java tool in "org.apache.axis.wsdl.WSDL2Java". The basic invocation form looks like this:

java org.apache.axis.wsdl.WSDL2Java userimpl.wsdl

Alternatively, you can use the wsdl directly from the web service itself.

java org.apache.axis.wsdl.WSDL2Java http://<host>/pbx/services/UserImpl?wsdl

where <host> is the host server name or host ip address.

This will generate only those bindings necessary for the client. Axis follows the JAX-RPC specification when generating Java client bindings from WSDL.

The WSDL2Java tool will create files in a directory structure that depends on the hostname. For instance, on a host with the default name of “localhost”, the stub or proxy classes were generated in “localhost\pbx\services\UserImpl” because that is the target namespace from the WSDL and namespaces map to Java packages. Compile the generated proxy classes and they will be ready for use.

7.4 Issues with the Proxy Generation approach

This automatic generation of *proxy classes* is convenient, as it makes calling a remote Web Service look almost like calling a local object. However, the developer should be aware of the following issues:

- These generated classes are only compatible with Axis. This is allowed by the JAX-RPC specification, which has a notion of compile time compatibility but not run-time compatibility. If you want stub classes that work with other organizations’ SOAP implementation, you would need to generate stub classes from the WSDL using their platform's tools. The stub classes should all have the same names and methods, so the rest of the code should not change.
- Remote Web Services are not the same as local objects. Pretending that they are is going to lead you astray. In particular, a method call to a local object often takes a few microseconds, while a call to a remote service can take tens of seconds, and fail with an obscure network error in the process, leaving the caller unsure if the call was successful or not. Making blocking calls to a Web Service from a web service will lead to a very unhappy end user experience.
- You have a more complex build process, as you need the WSDL before compiling the client.

8.0 Brekeke PBX Web Service Methods

In the future, the Brekeke PBX Web Service will make more of the PBX Admin available to third party applications. However, the current web service only exposes methods to view and modify property related to User Settings.

There is also a Third Party Call Control function that will allow you to create applications that make calls through the PBX Server (See Section 8.9).

8.1 getUserPropertyNames

Signature:
public String[] getUserPropertyNames(String userName) throws RemoteException,UserImplException;

Description:
Retrieves the names of properties that are currently saved in the user's property file.

Parameter:
username is a username as defined in the Brekeke PBX Admin.

Returns:
String Array containing the property names. If an error occurs, this method generates either a RemoteException or a UserImplException.

8.2 getUserProperties

Signature:
public String[] getUserProperties(String userName, String[] propertyNameArray) throws RemoteException,UserImplException;

Description:
Retrieves the Property values of properties specified in the propertyNameArray parameter.

Parameter:
username - a string representing a username as defined in the Brekeke PBX Admin.
propertyNameArray - a string array of property names whose values are to be retrieved.

Returns:
String array containing property values. If an error occurs, this method generates either a RemoteException or a UserImplException.

8.3 setUserProperties

Signature:
public void setUserProperties(String userName, String[] propertyNameArray, String[] propertyValues) throws RemoteException,UserImplException;

Description:
Sets the properties listed in the propertyNameArray to the corresponding values listed in the propertyValuesArray.

Parameters:
username - a string representing a username as defined in the Brekeke PBX Admin.
propertyNameArray - a string array of property names whose values are to be set.
propertyValuesArray- a string array containing values to be set.

Returns:
None. If an error occurs, this method generates either a RemoteException or a UserImplException.

8.4 getUserProperty

Signature:
public String getUserProperty(String userName, String propertyName) throws RemoteException,UserImplException;

Description:
Retrieves a property value.

Parameters:
username - a string representing a username as defined in the Brekeke PBX Admin.
propertyName - a string containing a property name.

Returns:
A string containing the property value. If an error occurs, this method generates either a RemoteException or a UserImplException.

8.5 setUserProperty

Signature:
public void setUserProperty(String userName, String propertyname, String value) throws RemoteException,UserImplException;

Description:
Sets the property listed in the propertyName parameter to the corresponding value in the propertyValue parameter.

Parameters:
username - a string representing a username as defined in the Brekeke PBX Admin.
propertyName - a string containing a property name.
propertyValuesArray- a string containing a property value.

Returns:
None. If an error occurs, this method generates either a RemoteException or a UserImplException.

8.6 createUser

Signature:
public boolean createUser(String userName, String password) throws RemoteException,UserImplException;

Description:
Creates a PBX User (extension).

Parameters:
userName - a string representing a username as defined in the Brekeke PBX Admin.
password - a string representing the corresponding password.

Returns:
True. If an error occurs, this method generates either a RemoteException or a UserImplException.

8.7 createUser (with default password)

Signature:
public boolean createUser(String userName) throws
RemoteException,UserImplException;

Description:
Creates a PBX User (extension).

Parameters:
userName - a string representing a username as defined in the Brekeke PBX Admin.

Returns:
True. If an error occurs, this method generates either a RemoteException or a UserImplException.

8.8 deleteUser

Signature:
public boolean deleteUser(String userName) throws RemoteException,UserImplException;

Description:
Deletes a PBX User (extension).

Parameters:
userName - a string representing a username as defined in the Brekeke PBX Admin.

Returns:
True. If an error occurs, this method generates either a RemoteException or a UserImplException.

8.9 callControl

Signature:
public String callControl(String user, String from, String[] to, String type) throws RemoteException,UserImplException;

Description:
Allows the creation of third party applications that can create calls through the PBX Server.

Parameters:
users – String representing the call owner. Usually this is an extension number.
from – String representing the from-url.
to – String array representing the to-url. Calls can be simultaneously made to different to-urls.
type – String “1” or string “2”. Type “1” will simultaneously call the from-url and the to-url then connect them. Type “2” will call the from-url first. When the from-url picks up, The to-url is called, then the two calls are connected.

Returns:
Success or Error message.

9.0 User Settings Properties

The table below contains property names that may be viewed or altered using the web service methods.

Property Name	Description	Value
Admin	Is the user an admin user type?	true or false
Allowjoin	Allow others to join conversation?	true or false
Canjoin	Allowed to join other's conversation?	true or false
Defaultpickup	Enable one touch Call Pickup for the predetermined group extensions by assigning the group number	group number
dtmfcommand	Allow this user to use call transfer/hold features?	true or false
Email	Email for forwarding voice mail info	email address
emailattachment	Attach WAV file to email?	true or false
emailnotification	Enable email notification?	true or false
Greetingtype	Greeting message	1 = Personal 2 = Alternative 3 = System
Index	Pattern Setting	number
Language	Language	en or ja
maxsessioncount	Maximum number of sessions	-1 means unlimited or 0 thru 6
messageforward	The extension number(s) to which received voicemail messages will be forwarded	list of extensions
Name	The description of this user	text
noanswerforward	number or SIP-URI to which the call will be forwarded when Ringer timeout has occurred	number or uri
Password	password	text
Personalivr	IVR Type	aa,setup,fm,ptn
phoneforward	phone number(s) or SIP-URI to directly forward all calls	comma separated list
Recording	Activate Call Recording?	True or false
Ringertime	Ringing timeout to wait for the called user to answer (in seconds)	number
Rtprelay	RTP Relay	true,false,true_codec, false_g711u,default
Userplugin	Type of Call Forwarding	user,fw,schedule,conf

10.0 Sample Programs

Sample Client Applications can be downloaded from the Brekeke website. The example programs all use the Proxy-generation approach to create the client applications. There are many other methods to create client applications.

10.1 ArrayGetterClient

This is sample program demonstrating how to retrieve property values using a string array of property names.

```
import localhost.sipadmin.services.UserImpl.*;

public class ArrayGetterClient {

	public static void main(String[] args) {
		try {
			String[] propNameArray = { "name", "password", "email",
					"ringertime", "userplugin", "rtprelay", "admin", "demo",
					"ptn.index", "index" };		//fill in an array of properties to retrieve
			String[] propValueArray;

			UserImplService afs = new UserImplServiceLocator();
			UserImpl af = afs.getUserImpl();
			propValueArray = af.getUserProperties("test", propNameArray); //retrieve the properties

			// results are returned as a String Array
			// loop through and display the contents of the String Array
			for (int i = 0; i < propNameArray.length; i++) {
				System.out.println(propNameArray[i] + "=" + propValueArray[i]);
			}

		} catch (Exception e) {
			System.err.println("Execution failed. Exception: " + e);
		}
	}
}
```

10.2 ArraySetterClient

This is a sample program demonstrating how to set property values using string arrays of property names and property values.

```
import localhost.sipadmin.services.UserImpl.*;

public class ArraySetterClient {

	public static void main(String[] args) {
		try {
			String[] propNameArray = { "name", "email" };		//fill in the an array of property names
			String[] propValueArray = { "Bob Dole1", "bob1@senator.org" }; //fill in an array of corresponding
values

			UserImplService afs = new UserImplServiceLocator();
			UserImpl af = afs.getUserImpl();

			af.setUserProperties("test", propNameArray, propValueArray); //sample method call

			System.out.println("Ok");

		} catch (Exception e) {
			System.err.println("Execution failed. Exception: " + e);
		}
	}
}
```

10.3 GetterClient

This is a sample program demonstrating how to retrieve a single property value.

```
import localhost.sipadmin.services.UserImpl.*;
import org.apache.axis.utils.Options;

public class GetterClient {

	public static void main(String[] args) {
		try {
			Options options = new Options(args);

			String userName = "test";
			String propName = "";

			args = options.getRemainingArgs();
			if ((args == null) || (args.length < 2)) {
				System.out.println("GetterClient <User name> <property name>");
				System.out.println("if <property name> is set to 'all', this will show all property names
and values.");
				return;
			} else {
				userName = args[0];
				propName = args[1];
			}

			UserImplService afs = new UserImplServiceLocator();
			UserImpl af = afs.getUserImpl();

			if (propName.equalsIgnoreCase("all")) {
				String[] propNameArray = af.getUserPropertyNames(userName);  //example method call

				for (int i = 0; i < propNameArray.length; i++) {
					System.out.println(propNameArray[i] + "="
							+ af.getUserProperty(userName, propNameArray[i]));
//example method call
				}
			} else {
				System.out.println(propName + "="
						+ af.getUserProperty(userName, propName));		//example
method call
			}

		} catch (Exception e) {
			System.err.println("Execution failed. Exception: " + e);
		}
	}
}
```

10.4 SetterClient

This is a sample program demonstrating how to set a single property value.

```
import localhost.sipadmin.services.UserImpl.*;
import org.apache.axis.utils.Options;

public class SetterClient {

	public static void main(String[] args) {
		try {
			Options options = new Options(args);

			String userName = "test";
			String propName = "";
			String propValue = "";

			args = options.getRemainingArgs();
			if ((args == null) || (args.length < 3)) {
				System.out.println("Usage: SetterClient <username> <property name> <property
value>");
				return;
			} else {
				userName = args[0];
				propName = args[1];
				propValue = args[2];
			}

			UserImplService afs = new UserImplServiceLocator();
			UserImpl af = afs.getUserImpl();

			af.setUserProperty(userName, propName, propValue);

			System.out.println("Ok");

		} catch (Exception e) {
			System.err.println("Execution failed. Exception: " + e);
		}
	}
}
```

10.5 CreateUserClient

This is a sample program demonstrating how to create a PBX user.

```
import localhost.sipadmin.services.UserImpl.*;
import org.apache.axis.utils.Options;

public class CreateUserClient {

	public static void main(String[] args) {
		try {
			Options options = new Options(args);

			String userName = "test";
			String password = "test";

			args = options.getRemainingArgs();
			if ((args == null) || (args.length < 2)) {
				System.out.println("Usage: CreateUserClient <username> <password>");
				return;
			} else {
				userName = args[0];
				password = args[1];
			}

			UserImplService afs = new UserImplServiceLocator();
			UserImpl af = afs.getUserImpl();

			if ( af.createUser(userName, password) ){
				System.out.println("Ok");
			}


		} catch (Exception e) {
			System.err.println("Execution failed. Exception: " + e);
		}
	}
}
```

10.6 ThirdPartyCC

This is a sample program demonstrating how to create a third party call control application.

```
import localhost.sipadmin.services.UserImpl.*;
import org.apache.axis.utils.Options;

public class ThirdPartyCC {

	public static void main(String[] args) {
		try {
			String[] destination = { "556", "557" };  //fill in the an array of destination numbers


			UserImplService afs = new UserImplServiceLocator();
			UserImpl af = afs.getUserImpl();

			System.out.println(af.callControl(“555”,”555”,destination,”2” ));

		} catch (Exception e) {
			System.err.println("Execution failed. Exception: " + e);
		}
	}
}
```

11.0 Web Service Security

Access to the Web Service is controlled using the PBX Administration Tool. Only Clients with IP Addresses that match the regular expression defined in the PBX Administration Tool will be allowed to consume from the web service.

To define the valid client ip addresses:

1. Browse to the PBX Administration tool.
2. Login with Admin privileges.
3. Select Options from the menu.
4. Enter a regular expression for the text box labeled “Valid Client IP Pattern”. For example, “192.168.*” or “192.168.0.5”.

12.0 Developing with PHP and other languages

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. WSDL is the standard language in which you describe Web services. It is an XML-based language that allows you to map a particular service in a language-neutral manner. It is therefore possible to access the Brekeke PBX web service using any programming language that has facilities to use the SOAP and WSDL standards.

In PHP 5, for instance, one can enable the soap extension and create the following application:

```
<?php
$client = new SoapClient( "http://< host >/pbx/services/UserImpl?wsdl");
$array = array("4007");
print($client->callControl("555","555",$array,"2"));
?>
```

Please consult with the documents for PHP on how to enable the soap extension.