

WebReboot[®] Enterprise

XML Soap Web Service API Manual

API Version 2.0.0
December 15, 2007

Servprise International, Inc.
106 Huntoon Memorial Highway
Rochdale, MA 01542

Web: www.servprise.com
Phone: 800-832-3823
Email: support@servprise.com



Table of Contents

1	Introduction.....	1
2	Installation.....	1
3	Using the API.....	1
3.1	Getting Started	1
3.1.1	Service Call Architecture	2
3.1.2	Connecting to the WebReboot Enterprise.....	2
3.2	Retrieving Server Port Configuration	3
3.2.1	Getting a Server’s Port Number.....	3
3.2.2	Getting a Server’s Name	3
3.2.3	Getting a Server’s Status.....	4
3.2.4	What Reboot Methods Does the Server Support?	5
3.3	Rebooting Servers	5
3.3.1	Performing a Reset Switch Reboot	6
3.3.2	Performing a Power Switch Reboot.....	6
3.3.3	Power-off a Server	7
3.3.4	Power-on a Server.....	8
3.4	Server Administration Actions	8
3.4.1	Renaming a Server	9
3.4.2	Setting a Server to Use a Reset Switch Reboot Method	9
3.5	WebReboot Enterprise Administration Actions	10
3.5.1	Changing Network Configuration.....	10
3.5.2	Changing Authentication Credentials	11
3.5.3	Get System Information.....	11
3.5.4	Changing Log Configuration	12
	Appendix A – Example Command Line Application.....	13

1 Introduction

The WebReboot Enterprise API will allow you to programmatically control your WebReboot Enterprise product. Everything you can do with the built-in WebReboot Enterprise Web interface can be accomplished via the API, letting you completely integrate the WebReboot Enterprise into your own software. The purpose of this document is to introduce you to the API methods and provide illustrative examples of working with your WebReboot Enterprise.

2 Installation

The WebReboot Enterprise API is implemented as a Web service via the Web Services Description Language (WSDL). WSDL is standardized by the World Wide Web Consortium and as such, is supported by nearly all modern programming languages. The WSDL file representing the programmatic interface for the WebReboot Enterprise can be found at <http://dev.servprise.com/schemas/2007/12/WebRebootService.wsdl>.

In order to use the WSDL file in your own software, you will need to find an appropriate library for your language of choice. For illustrative purposes, we demonstrate use of the API in Java using the Apache Axis library¹.

3 Using the API

3.1 Getting Started

The first step to using the WebReboot Engine API is to install a library that is capable of interacting with WSDL for your language of choice. As indicated previously, the remainder of this document shall assume you are using Java with the Apache Axis library. The Axis library can be installed by simply adding the axis.jar and all dependencies to your project's classpath.

The next step is processing the WSDL file. Some languages, such as Python, handle this dynamically. Others, such as Java, require class file generation. Axis provides a utility named WSDL2Java that will generate proxies and skeleton code for interacting with the remote service. You may invoke it like so:

```
java org.apache.axis.wsdl.WSDL2Java WebRebootService.wsdl
```

Optionally, you may integrate the code generation directly into your ant build process with the following snippets:

```
<taskdef name="wsdl2java" classname="org.apache.axis.tools.ant.wsdl.Wsdl2javaAntTask"
  classpathref="project.classpath" />
:
```

¹ Axis libraries for both Java and C++ are available at <http://ws.apache.org/axis/>.

```
<wsdl2java debug="false" helperGen="false" noimports="false"
  output="{generatedSrc.dir}" serverside="false" skeletonDeploy="false"
  testcase="false" url="file://{conf.dir}/WebRebootService.wsdl" verbose="false"
/>
```

Please note that you will likely have to modify the paths to match those of your existing build environment.

Once you have generated the class files, you have everything you need to begin programming your WebReboot Enterprise. Once again, what follows is Java code, but the API defined by the WSDL file is identical across all languages. Each library may alter the generated code to produce something proper for that language (e.g., Axis generates camel-case method names), but the arguments, names, and overall calling conventions of the API are identical for all implementations.

3.1.1 Service Call Architecture

The WebReboot Enterprise deals with users on a call-by-call basis. That is to say, each service call is atomic and as such there is no concept of user “sessions.” A maximum of three operations may be performed at any given time by as many users.

The Web service is fundamentally a request/response architecture. A request object must be constructed by the client for each service call. The server will construct a corresponding response object that will be returned upon the end of the call. The WSDL file defines the structure for both sets of objects and Axis will generate stub classes for working with each.

Although each request and response object is unique, every response object contains a reference to a `CommandResult` instance. Each `CommandResult` contains a field indicating whether or not there was an error executing a given operation and a field that provides a textual status message that may be used to further define either the error or success condition. Please see the examples throughout this document on how you may use the `CommandResult` instance for error checking in your application.

3.1.2 Connecting to the WebReboot Enterprise

Before you perform any operation on the WebReboot Enterprise, you must first connect to it. The WebReboot Enterprise uses HTTP Basic authentication for all method calls. This is a standard means of authenticating with remote Web hosts, but it is also insecure as it requires clear-text transmission of your credentials. For this reason, we highly recommend that, unless on a trusted network, you use HTTPS rather than HTTP for communication. You may connect to the WebReboot Enterprise using the following code snippet:

```
WebRebootService service = new WebRebootServiceLocator();

URL url = new URL("http://" + ipAddress + ":" + port +
  "/admin/services/WebRebootService");
WebReboot wr = service.getWebRebootService(url);

((WebRebootSoapBindingStub) wr).setMaintainSession(true);
```

```

((WebRebootSoapBindingStub) wr)._setProperty(javax.xml.rpc.Stub.USERNAME_PROPERTY,
    "admin");

((WebRebootSoapBindingStub) wr)._setProperty(javax.xml.rpc.Stub.PASSWORD_PROPERTY,
    "password");

```

You should replace the connection URL with the appropriate scheme (HTTP or HTTPS) and hostname/IP address. Likewise, you should substitute the appropriate username for “admin” and the user’s corresponding password for “password.” Note that by default, the WebReboot Enterprise uses DHCP to retrieve its IP address, so you may need to coordinate deployment with your network administrator.

3.2 Retrieving Server Port Configuration

Information about each port in the WebReboot Enterprise can be retrieved via the `getWebRebootPort` method. The response object from the WebReboot Enterprise will contain a `WebRebootPort` instance, which includes the port number, associated server name, and enabled reboot types for the server, as well as server status information.

Alternatively, information about every port in the WebReboot Enterprise can be retrieved at once using the `getBatchWebRebootPort` method. In this case, the response object will contain an array of `WebRebootPort` instances. To access the information for a desired port, you will need to index into this array. NB: The array is zero-indexed, while the WebReboot Enterprise port numbers are one-indexed.

3.2.1 Getting a Server’s Port Number

The concept of a port number is a very important one when interacting with the WebReboot Enterprise. By definition, the port number is the unique identifier for each server connected to your WebReboot Enterprise. All server-related methods require supplying the port number in the request object.

Since keeping track of port numbers is such an important task, the API offers some assistance. Each `WebRebootPort` instance contains its identifying port number so that your application may reason at a high-level without having to worry about “house keeping chores,” such as `WebRebootPort` : port number mappings.

Please see the examples of using the port number information in Section 3.2.2 & throughout Section 3.3 to better understand how this information may be used.

3.2.2 Getting a Server’s Name

Every server connected to the WebReboot can have a textual name associated with it. The purpose of the name is to convey some meaning for users responsible for that server. The name of a server can be retrieved via the `getBatchWebRebootPort` method and can be used to enhance your application’s user interface. The following example shows how all the servers the current user has permissions to reboot can be printed out to the console.

```
GetBatchWebRebootPortRequest request = new GetBatchWebRebootPortRequest();
```

```

GetBatchWebRebootPortResponse response = wr.getBatchWebRebootPort(request);
WebRebootPort[] ports = response.getWebRebootPortArray().getWebRebootPort();
for (int i = 0; i < ports.length; i++)
{
    System.out.println(ports[i].getPortNumber() + ". " + ports[i].getName());
}

```

Note that the indexing starts at 0 while the WebReboot Enterprise port numbering starts at 1.

An alternative to the preceding code block would be to retrieve each port's information in a separate request-response cycle. Typically, you would not want to do this, since the large number of request-response cycles could introduce significant delay into your application. On the other hand, this approach may be beneficial if you need to retrieve information about a small range of ports, as the batch retrieval will include information for all ports the user has permissions for – even those you do not care about. As an example, the following code block demonstrates how to print the textual name associated with only the first ten ports on the WebReboot Enterprise.

```

for (int i = 0; i < 10; i++)
{
    GetWebRebootPortRequest request = new GetWebRebootPortRequest(i + 1);
    GetWebRebootPortResponse response = wr.getWebRebootPort(request);

    WebRebootPort port = response.getWebRebootPort();

    System.out.println(port.getPortNumber() + ". " + port.getName());
}

```

Note in this case that the loop indexing is 0-based, but the request object is 1-based; it accepts the actual WebReboot Enterprise port number as its single argument. Also, in this example it is assumed the user has permissions to access information about the first ten ports. If the user does not, the response object will contain information about the error state.

3.2.3 Getting a Server's Status

The `getWebRebootPort` method can be used to determine a remote server's status. Currently, the connection status, indicating whether or not a server is physically connected to a particular WebReboot port, as well as the power status, indicating whether a server is powered on or off, are retrievable.

```

GetBatchWebRebootPortRequest request = new GetBatchWebRebootPortRequest();
GetBatchWebRebootPortResponse response = wr.getBatchWebRebootPort(request);

WebRebootPort[] ports = response.getWebRebootPortArray().
getWebRebootPort();

for (int i = 0; i < ports.length; i++)
{
    WebRebootPort port = ports[i];

    if (true == port.getServerResetSwitchSupported())
    {
        System.out.println("Can reboot via reset switch: " + port.getName());
    }

    if (true == port.getServerPowerSwitchSupported())

```

```

    {
        System.out.println("Can reboot, power off, and power on via power switch: " +
            port.getName());
    }
}

```

Please note that the connection status of a port is only retrievable when using a Servprise Advanced Server Card attached to the port.

3.2.4 What Reboot Methods Does the Server Support?

The `getWebRebootPort` method can be used to determine which type of reboot method to call. Accordingly, it may be used to tailor your application's user interface. The following example outputs statements indicating the reboot types each port supports. For your own software, you may use similar code to enable or disable GUI buttons for the various reboot types.

```

GetBatchWebRebootPortRequest request = new GetBatchWebRebootPortRequest();
GetBatchWebRebootPortResponse response = wr.getBatchWebRebootPort(request);

WebRebootPort[] ports = response.getWebRebootPortArray().
getWebRebootPort();

for (int i = 0; i < ports.length; i++)
{
    WebRebootPort port = ports[i];

    if (true == port.getServerResetSwitchSupported())
    {
        System.out.println("Can reboot via reset switch: " + port.getName());
    }

    if (true == port.getServerPowerSwitchSupported())
    {
        System.out.println("Can reboot, power off, and power on via power switch: " +
            port.getName());
    }
}

```

Notice that reboot type status is indicated by boolean values. A value of “false” indicates that the reboot type is disabled, while “true” indicates that the reboot type is enabled.

3.3 *Rebooting Servers*

The WebReboot Enterprise supports two types of reboots for servers. The first type works if the connected server has a reboot switch. This is the traditional method for performing a reboot and satisfies the vast majority of rebooting use cases. The second type of reboot operates by turning the computer off for several seconds and then turning it back on. This method of reboot is necessary if the connected server does not allow a reset switch to be connected to the motherboard. It should be noted that the power-on/power-off is performed via the power switch connector on a motherboard and thus will not cause potentially damaging power surges to the server.

The WebReboot Enterprise API has the means of determining the configured reboot capabilities for a given server. Likewise, it has a set of methods for performing the various types of reboots. By default, only power switch reboots are enabled on all ports of the WebReboot Enterprise.

3.3.1 Performing a Reset Switch Reboot

As with all other server actions, the reset switch reboot action requires a properly formatted request object. Please be careful that you supply the correct port number in constructing the request, lest you inadvertently reboot the wrong computer. We recommend that your application's user interface provides a confirmation prompt prior to calling the reboot method. Likewise, you should verify that the port supports reset switch rebooting before calling the method on the server. For example:

```
// Prompt user for information.
int portNumber = getPortNumber();

GetWebRebootPortRequest infoRequest = new GetWebRebootPortRequest();
infoRequest.setPortNumber(portNumber);

GetWebRebootPortResponse infoResponse = wr.getWebRebootPort(infoRequest);

WebRebootPort port = infoResponse.getWebRebootPort();

// Check that the server supports reset reboot.
if (true == port.getServerResetSwitchSupported())
{
    System.out.println("You are about to reset switch reboot: " + port.getName());

    // Prompt user for confirmation of reboot action.
    boolean confirm = confirmAction();
    if (true == confirm)
    {
        ResetSwitchRebootRequest rebootRequest = new ResetSwitchRebootRequest();
        rebootRequest.setPortNumber(port.getPortNumber());

        ResetSwitchRebootResponse rebootResponse = wr.doResetSwitchReboot(rebootRequest);

        // Check if there was an error executing the command.
        CommandResult result = rebootResponse.getCommandResult();

        if (result.isError())
        {
            System.err.println(result.getResultMessage());
        }
    }
}
}
```

3.3.2 Performing a Power Switch Reboot

A power switch reboot action is very similar to a reset switch reboot action. As with the reset switch reboot, please be careful that you supply the correct port number in constructing the request, lest you inadvertently reboot the wrong computer. We recommend that your application's user interface provides a confirmation prompt prior to calling the reboot method. Likewise, you should verify that the port supports power switch rebooting before calling the method on the server. For example:

```
// Prompt user for information.
int portNumber = getPortNumber();

GetWebRebootPortRequest infoRequest = new GetWebRebootPortRequest();
infoRequest.setPortNumber(portNumber);

GetWebRebootPortResponse infoResponse = wr.getWebRebootPort(infoRequest);

WebRebootPort port = infoResponse.getWebRebootPort();
```

```

// Check that the server supports power operations.
if (true == port.getServerPowerSwitchSupported())
{
    System.out.println("You are about to power switch reboot: " + port.getName());

    // Prompt user for confirmation of reboot action.
    boolean confirm = confirmAction();
    if (true == confirm)
    {
        PowerRebootRequest rebootRequest = new PowerRebootRequest();
        rebootRequest.setPortNumber(port.getPortNumber());

        PowerRebootResponse rebootResponse = wr.doPowerReboot(rebootRequest);

        // Check if there was an error executing the command.
        CommandResult result = rebootResponse.getCommandResult();

        if (result.isError())
        {
            System.err.println(result.getResultMessage());
        }
    }
}
}

```

3.3.3 Power-off a Server

Servers that are configured to use the power reboot method may also be turned on and turned off via the WebReboot Enterprise. The `doPowerOff` method is used to turn a server off. Please note that this command will be executed upon request regardless of the connected server's actual status; it simply sends a power-off command to the server by holding the power switch pins for four seconds.

As with the reboot actions, great care should be taken to ensure that the power-off command is sent to the correct server. We recommend that your application's user interface provides a confirmation prompt prior to calling the `doPowerOff` method. Likewise, you should verify that the port supports power toggling before calling the method on the server. For example:

```

// Prompt user for information.
int portNumber = getPortNumber();

GetWebRebootPortRequest infoRequest = new GetWebRebootPortRequest();
infoRequest.setPortNumber(portNumber);

GetWebRebootPortResponse infoResponse = wr.getWebRebootPort(infoRequest);

WebRebootPort port = infoResponse.getWebRebootPort();

// Check that the server supports power operations.
if (true == port.getServerPowerSwitchSupported())
{
    System.out.println("You are about to power-off: " + port.getName());

    // Prompt user for confirmation of power toggle action.
    boolean confirm = confirmAction();
    if (true == confirm)
    {
        PowerOffRequest powerOffRequest = new PowerOffRequest();
        powerOffRequest.setPortNumber(port.getPortNumber());

        PowerOffResponse powerOffResponse = wr.doPowerOff(powerOffRequest);

        // Check if there was an error executing the command.
    }
}

```

```

        CommandResult result = powerOffResponse.getCommandResult();
        if (result.isError())
        {
            System.err.println(result.getResultMessage());
        }
    }
}

```

3.3.4 Power-on a Server

Servers that are configured to use the power reboot method may also be turned on and turned off via the WebReboot Enterprise. The `doPowerOn` method is used to turn a server on. Please note that this command will be executed upon request regardless of the connected server's actual status; it simply sends a power-on command to the server by momentarily holding the power switch.

As with the reboot actions, great care should be taken to ensure that the power-on command is sent to the correct server. We recommend that your application's user interface provides a confirmation prompt prior to calling the `doPowerOn` method. Likewise, you should verify that the port supports power toggling before calling the method on the server. For example:

```

// Prompt user for information.
int portNumber = getPortNumber();

GetWebRebootPortRequest infoRequest = new GetWebRebootPortRequest();
infoRequest.setPortNumber(portNumber);

GetWebRebootPortResponse infoResponse = wr.getWebRebootPort(infoRequest);

WebRebootPort port = infoResponse.getWebRebootPort();

// Check that the server supports power operations.
if (true == port.getServerPowerSwitchSupported())
{
    System.out.println("You are about to power-on: " + port.getName());

    // Prompt user for confirmation of power toggle action.
    boolean confirm = confirmAction();
    if (true == confirm)
    {
        PowerOnRequest powerOnRequest = new PowerOnRequest();
        powerOnRequest.setPortNumber(port.getPortNumber());

        PowerOnResponse powerOnResponse = wr.doPowerOn(powerOnRequest);

        // Check if there was an error executing the command.
        CommandResult result = powerOnResponse.getCommandResult();
        if (result.isError())
        {
            System.err.println(result.getResultMessage());
        }
    }
}
}

```

3.4 Server Administration Actions

The actions described in this section may only be performed by the WebReboot Enterprise administrator. These actions are the “set” analogues of the “get” actions described in Section 3.2. As the `getWebRebootPort` method is used to retrieve the WebReboot Enterprise port configurations, the `editWebRebootPort` method is used to edit those configurations.

3.4.1 Renaming a Server

An administrator can change the textual name associated with a given server port. This is the “set” action that corresponds to the “get” action described in Section 3.2.2. The `editWebRebootPort` method may be used in the following manner to change that name:

```
// Prompt user for information.
int portNumber = getPortNumber();
String newServerName = getNewServerName();

EditWebRebootPortRequest editPortRequest = new EditWebRebootPortRequest();
editPortRequest.setPortNumber(portNumber);
editPortRequest.setName(newServerName);

EditWebRebootPortResponse editPortResponse = wr.editWebRebootPort(editPortRequest);

// Check if there was an error executing the command.
CommandResult result = editPortResponse.getCommandResult();
if (result.isError())
{
    System.err.println(result.getResultMessage());
}
```

3.4.2 Setting a Server to Use a Reset Switch Reboot Method

By default, all the servers connected to a WebReboot Enterprise are configured to use a power switch reboot method. If you wish to enable or disable the reboot method for a given server, you may do so with the `editWebRebootPort` service call.

In the event that the server is already configured with the specified enable/disable value, this method is essentially a no-op. Otherwise, the reboot status is immediately updated to the value provided. Please note that simply enabling the reboot method is not sufficient enough for allowing reboots of this type -- you will have to ensure there is a physical connection from the WebReboot Enterprise to your motherboard that matches the reboot type as well. An example of the method’s usage follows:

```
// Prompt user for information.
int portNumber = getPortNumber();
final boolean switchEnabledValue = true;
final boolean switchDisabledValue = false;

EditWebRebootPortRequest editSupportedSwitchesRequest = new EditWebRebootPortRequest();
editSupportedSwitchesRequest.setPortNumber(portNumber);
editSupportedSwitchesRequest.setServerPowerSwitchSupported(switchDisabledValue);
editSupportedSwitchesRequest.setServerResetSwitchSupported(switchEnabledValue);

EditWebRebootPortResponse editSupportedSwitchesResponse = wr.editWebRebootPort
(editSupportedSwitchesRequest);

// Check if there was an error executing the command.
CommandResult result = editSupportedSwitchesResponse.getCommandResult();
if (result.isError())
{
    System.err.println(result.getResultMessage());
}
```

3.5 WebReboot Enterprise Administration Actions

The actions described in this section may only be performed by the WebReboot Enterprise administrator. Whereas the rest of this document details how to configure and manipulate servers attached to the WebReboot Enterprise, this section shows how the WebReboot Enterprise itself may be configured.

3.5.1 Changing Network Configuration

By default, the WebReboot Enterprise is configured to retrieve its network configuration via DHCP. In certain contexts, this may be inappropriate, so you may configure the network as fit for your network with the following approach.

As an example, if you would like to set up your WebReboot Enterprise to a static IP address, you may use the following code sample:

```
//
// First, we'll retrieve the existing configuration and print it
// to the console.
//

GetNetworkConfigurationRequest getRequest = new GetNetworkConfigurationRequest();

GetNetworkConfigurationResponse getResponse = wr.getNetworkConfiguration(getRequest);

// Check if there was an error executing the command.
CommandResult getResult = getResponse.getCommandResult();
if (getResult.isError())
{
    System.err.println(getResult.getResultMessage());
}

// Print network configuration details to the console
System.out.println("IP Address: " + getResponse.getIpAddress() );

System.out.println("Subnet Mask: " + getResponse.getSubnetMask() );

System.out.println("DHCP Enabled?: " + getResponse.getDhcpEnabled().toString() );

//
// Second, we'll edit the configuration to use a static IP address
//

EditNetworkConfigurationRequest editRequest = new EditNetworkConfigurationRequest();

// Disable DHCP (i.e., setDhcpEnabled to FALSE).
editRequest.setDhcpEnabled(dhcpDisabledValue);

// Set the new network information.
editRequest.setIpAddress(newIpAddress);
editRequest.setSubnetMask(newSubnetMask);

EditNetworkConfigurationResponse editResponse = wr.editNetworkConfiguration(editRequest);

// Check if there was an error executing the command.
CommandResult editResult = editResponse.getCommandResult();
if (editResult.isError())
{
    System.err.println(editResult.getResultMessage());
}

// At this point, the WebReboot Enterprise will begin to reboot with new
```

```
// new network configuration. Your application will need to re-establish
// the connection to the web service at the new address.

Thread.sleep(60000); // Wait for the WRE to reboot, then reconnect...
```

3.5.2 Changing Authentication Credentials

The WebReboot Enterprise has a single administrator account by default. The login name for this account is “admin” and the password is “password.” You will have most likely changed these defaults, however, during the initial configuration of your WebReboot Enterprise. Nevertheless, it is possible to programmatically change the authentication credentials via the following method:

```
EditPasswordRequest request = new EditPasswordRequest();

// Set the new password
request.setPassword(newPassword);

EditPasswordResponse response = wr.editPassword(request);

// Check if there was an error executing the command.
CommandResult result = response.getCommandResult();
if (result.isError())
{
    System.err.println(result.getResultMessage());
}
```

The password is transmitted in plaintext with no encryption or hashing. Please make sure that you use a secure transport layer (e.g., HTTPS) when using this method.

3.5.3 Get System Information

The system information for each WebReboot Enterprise describes how many physical ports are available on the device, as well as version information for the firmware, API, and configuration file.

The firmware, API, and configuration file are each versioned separately to allow minor changes to each in the future without affecting backward compatibility.

```
GetSystemInformationRequest request = new GetSystemInformationRequest();
GetSystemInformationResponse response = wr.getSystemInformation(request);

// Check if there was an error executing the command.
CommandResult result = response.getCommandResult();
if (result.isError())
{
    System.err.println(result.getResultMessage());
}

//Print system information details to console
System.out.println("Number of ports: " + response.getNumberOfPorts() );

System.out.println("Firmware version: " + response.getFirmwareVersion() );

System.out.println("API version: " + response.getApiVersion() );

System.out.println("Configuration version: " + response.getConfigurationFileVersion() );
```

3.5.4 Changing Log Configuration

The log configuration of the WebReboot Enterprise can be set for the local log file, remote Syslog server, as well as the error LED on the front of the chassis.

```
//
// First, we'll retrieve the existing log configuration and print it
// to the console.
//

GetLogConfigurationRequest getRequest = new GetLogConfigurationRequest();

GetLogConfigurationResponse getResponse = wr.getLogConfiguration(getRequest);

// Check if there was an error executing the command.
CommandResult getResult = getResponse.getCommandResult();
if (getResult.isError())
{
    System.err.println(getResult.getResultMessage());
}

// Get the aggregate element surrounding the log configuration details
LogConfiguration logConfig = getResponse.getLogConfiguration();

// Print log configuration details to console
System.out.println("Log level of local file: " +
    logConfig.getLocalFileLogLevel().toString() );

System.out.println("Log level of chassis error LED: " +
    logConfig.getChassisErrorLedLogLevel().toString() );

System.out.println("Log level of remote Syslog server: " +
    logConfig.getSyslogLogLevel().toString() );

System.out.println("IP address of remote Syslog server: " +
    logConfig.getSyslogServerIpAddress() );

//
// Second, we'll edit the configuration to log all messages (i.e., a log
// level of 'DEBUG') to a Syslog server at a static IP address.
//

EditLogConfigurationRequest editRequest = new EditLogConfigurationRequest();

// Set the IP address for the Syslog server
editRequest.setSyslogServerIpAddress("10.8.0.1");

// Set the log level to 'DEBUG' so all messages get logged, no matter how
// trivial they may be
editRequest.setSyslogLogLevel(LogLevelEnum.DEBUG);

EditLogConfigurationResponse editResponse =
    wr.editLogConfiguration(editRequest);

// Check if there was an error executing the command.
CommandResult editResult = editResponse.getCommandResult();
if (editResult.isError())
{
    System.err.println(editResult.getResultMessage());
}
}
```

Appendix A – Example Command Line Application

```

package com.servprise.documentcode.wreapimanual;

import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;

import javax.xml.rpc.ServiceException;

import com.servprise.dev.schemas._2007._06.WebRebootService_wsdl.*;
import com.servprise.dev.schemas._2007._06.WebRebootService_xsd.*;

public class ExampleCommandLineApplication
{
    /**
     * A bare-minimum command line application to connect to the WebReboot Enterprise via
     * the SOAP web service, and that pulls several of the API examples together.
     *
     * @param args - we need 1 argument... an IP address
     */
    public static void main(String[] args)
    {
        // Make sure we have an IP address as an argument
        if(1 != args.length)
        {
            System.out.println("We require an IP address as the one and only argument.");
            return;
        }

        String ipAddress = args[0];

        // Set up the connection to the service with default authentication
        // credentials of "admin" and "password".
        WebRebootService service = new WebRebootServiceLocator();
        WebReboot wr = null;
        try
        {
            wr = service.getWebRebootService(new URL("http://" + ipAddress +
                "/admin/services/WebRebootService"));
        }
        catch (MalformedURLException e)
        {
            System.err.println("The IP address we received was not properly formed. " +
                "Please double check the IP address.");
        }
        catch (ServiceException e)
        {
            System.err.println("We failed to connect to the web service. " +
                "Please double check the IP address.");
        }
        ((WebRebootSoapBindingStub) wr).setMaintainSession(true);
        ((WebRebootSoapBindingStub) wr)._setProperty(javax.xml.rpc.Stub.USERNAME_PROPERTY,
            "admin");
        ((WebRebootSoapBindingStub) wr)._setProperty(javax.xml.rpc.Stub.PASSWORD_PROPERTY,
            "password");

        //
        // Print the system information to the console...
        //

        GetSystemInformationRequest request = new GetSystemInformationRequest();
        GetSystemInformationResponse response = null;
        try
        {
            response = wr.getSystemInformation(request);
        }
        catch (RemoteException e)
    }
}

```

```
{
    // A remote exception was thrown, meaning something failed in the
    // actual transmission of XML/SOAP data back and forth.
    e.printStackTrace();
}

// Check if there was an error executing the command.
CommandResult result = response.getCommandResult();
if (result.isError())
{
    // An application-level problem occurred, meaning the WebReboot
    // Enterprise probably did not like the parameters it received
    // in the web service call.
    System.err.println(result.getResultMessage());
}

System.out.println("Number of ports: " + response.getNumberOfPorts() );

System.out.println("Firmware version: " + response.getFirmwareVersion() );

System.out.println("API version: " + response.getApiVersion() );

System.out.println("Configuration version: " + response.getConfigurationFileVersion()
);
}
}
```