



CHAPTER 1

SOAP Architecture and Application Programming Interface

This chapter describes the SOAP adapter architecture and application programming interface (API) for the Cisco BTS 10200 Softswitch. This chapter includes the following sections:

- [SOAP Adapter Architecture, page 1-1](#)
- [Cisco BTS 10200 Softswitch API, page 1-7](#)

SOAP Adapter Architecture

The SOAP adapter interface leverages the adapter architecture of the Element Management System (EMS) component in the Cisco BTS 10200 Softswitch. This architecture allows for a variety of adapters to provide operations, administration, management, and provisioning (OAM&P) by adapting the external interface to a common infrastructure in the EMS.

Login sessions expire in 10 minutes with no activity. This means that a command must traverse each session once every 10 minutes to keep a session alive. This is important for any client application that deploys the use of connection pools.

SOAP Interface Specifications

The SOAP interface adapter conforms to SOAP 1.2 specifications.

Compiler Tools

The SOAP adapter utilizes the J2SE Development Kit (JDK)–1.4.1. The Cisco BTS 10200 uses JDK1.4.2 for compilation. JDK1.5.0 or JDK 5 for the Java Runtime Environment (JRE).

Client-side applications may require the following tools:

- Xerces parsers
- ECS Report Builder

WSDL Stub Generation

The SOAP adapter uses the Apache AXIS toolkit to generate skeletons and stubs. The skeletons and stubs abstract the SOAP interface from the BTS 10200 middleware.

This code example uses the Java package tree as developed in the Cisco BTS 10200 Softswitch product. This code can vary. Other clients can specify a different package tree to contain the WSDL interface objects. See the SDK for a detailed breakdown of this script.

```
#!/bin/sh
#####
# Copyright (c) 2002, 2006 by Cisco Systems, Inc.
#
#
#####
set -e
set -a
#set -x

#
# List required jar files
#
AXISLIB=/opt/Tomcat/webapps/axis/WEB-INF/lib
CLASSPATH=$AXISLIB/axis.jar:$AXISLIB/commons-logging-1.0.4.jar:$AXISLIB/jaxrpc:$AXISLIB/commons-discovery-0.2.jar:$AXISLIB/saaj:$AXISLIB/wsdl4j-1.5.1.jar

export CLASSPATH

java -classpath $CLASSPATH org.apache.axis.wsdl.WSDL2Java -o . -p
com.sswitch.oam.soap.client bts10200.wsdl
```

Java files are generated in a local directory tree specified in the package directory. This package path is required in the bind logic to find the object interface implementation.

```
#!/bin/sh
#####
# Copyright (c) 2002, 2006 by Cisco Systems, Inc.
#
#
#####
set -e
set -a
#set -x
#
# List required jar files
#
AXISLIB=/opt/Tomcat/webapps/axis/WEB-INF/lib
CLASSPATH=$AXISLIB/axis.jar:$AXISLIB/commons-logging-1.0.4.jar:$AXISLIB/jaxrpc:$AXISLIB/commons-discovery-0.2.jar:$AXISLIB/saaj:$AXISLIB/wsdl4j-1.5.1.jar

export CLASSPATH

javac -classpath $CLASSPATH -d ./ com/sswitch/oam/soap/client/*.java
```

Compile a package of Java files to generate the required class files. These class files must exist in the client classpath.



Note

This is a common example where all the java files in a single directory are built with a single command. This is one of the fastest ways to compile bulk java code.

The Cisco BTS 10200 Softswitch offers a Software Developers Kit (SDK) with a complete range of examples that utilize the SOAP interface. These include many topics such as:

- CLI
- Example proxy

- Batch file processing

**Note**

The example below illustrates the basic extraction of the BTS 10200 objects.

For example:

```
package com.sswitch.oam.ccc;

import com.sswitch.oam.soap.client.*;
import java.net.*;

/**
 * Copyright (c) 2002-2004, 2006 by Cisco Systems, Inc.
 */
public class SoapProvAdapter implements XMLAdapter {

    private String key="";

    private String url=null;

    private Bts10200Operations port=null;

    private String host="";

    public SoapProvAdapter(String args[]) {

        //
        // General SSL properties
        //
        java.util.Properties props = System.getProperties();
        props.put( "javax.net.ssl.keyStore", "bts10200_ks" );
        props.put( "javax.net.ssl.keyStorePassword", "Chillan" );
        props.put( "javax.net.ssl.trustStore", "bts10200_ts" );
        props.put( "javax.net.ssl.trustStorePassword", "Chillan" );

        // Plug our context finder class into the SSL extension
        for(int i=0; i<args.length; i++) {

            if (args[i].equals("-b")) {

                host=args[i+1];

                break;

            }

        }

    }

    /**
     * Connect to target server specified by destAddr, using default user and password
     * @param destAddr Hostname, ip address or URL of the destination
     */
    public void connect() throws XMLAdapterException {

        connect("btsadmin","btsadmin");

    }

    /**
     * Connect to target server specified by destAddr, with provided user and password
     * @param destAddr Hostname, ip address or URL of the destination
     */
}
```

```

public void connect(String user, String pass) throws XMLAdapterException {

    try {

        disconnect();

    } catch(Exception e) {}

    try {

        if (host.equals("")) {

            throw new XMLAdapterException(100, "not a valid host
name");

        }

        url="https://" + host + "/axis/services/bts10200";

        port=new Bts10200OperationsServiceLocator().getbts10200(new
URL(url));

        key=port.login(user,pass);

    } catch(BtsSoapException e) {

        throw new XMLAdapterException(100, e.getError_string());

    } catch(Exception e) {

        throw new XMLAdapterException(100, e.toString());

    }

}

/**
 * Make a request to execute a normal CLI command
 * @param request The XML request document
 * @param String The XML formatted answer
 */
public synchronized String request(String request) throws XMLAdapterException {

    try {

        testConnected();

        return port.request(request,key);

    } catch (XMLAdapterException e) {

        throw e;

    } catch(BtsSoapException e) {

        throw new XMLAdapterException(100, e.getError_string());

    } catch(Exception e) {

        throw new XMLAdapterException(100, e.toString());

    }

}

```

```

/**
 * Make a getCommandDoc request to
 * @param request The XML request document
 * @param String The XML formatted answer
 */
public synchronized String getCommandDoc(String verb, String noun) throws
XMLAdapterException {

    try {

        testConnected();

        return port.getCommandDoc(verb,noun,key);

    } catch (XMLAdapterException e) {

        throw e;

    } catch(BtsSoapException e) {

        throw new XMLAdapterException(e.getError_code(),
e.getError_string());

    } catch(Exception e) {

        throw new XMLAdapterException(1, e.toString());

    }

}

public synchronized String getExtCommandDoc(String verb, String noun) throws
XMLAdapterException {

    try {

        testConnected();

        return port.getExtCommandDoc(verb,noun,key);

    } catch (XMLAdapterException e) {

        throw e;

    } catch(BtsSoapException e) {

        throw new XMLAdapterException(e.getError_code(),
e.getError_string());

    } catch(Exception e) {

        throw new XMLAdapterException(1, e.toString());

    }

}

public synchronized String executeMacro(String command) throws XMLAdapterException {

    throw new XMLAdapterException(1, "not implemented yet!");

}

```

```

    }

    public void disconnect() throws XMLAdapterException {

        try {

            if (port != null && key!=null) {

                port.logout(key);

            }

        } catch(BtsSoapException e) {

            throw new XMLAdapterException(e.getError_code(),
e.getError_string());

        } catch(Exception e) {

            throw new XMLAdapterException(1, e.toString());

        } finally {

            port=null;
            key=null;

        }

    }

    private void testConnected() throws XMLAdapterException {

        if (port==null || key==null) {

            throw new XMLAdapterException(1, "Adapter is not connected to
target box !");

        }

    }

}

```

WSDL Overview

The WSDL is used to express the interface in the SOAP adapter. This section provides an overview of the WSDL for the Cisco BTS 10200 Softswitch. These WSDL operations define access to the XML descriptions and documents used to provision the Cisco BTS 10200 Softswitch. A full description of the XML document is covered in a later chapter. For the most part, SOAP acts as the transport for these documents.

BTS 10200 Softswitch WSDL

The bts10200.wsdl file contains the general system-wide data structures and type definitions. It also contains the error interfaces (exceptions). See the [“Cisco BTS 10200 Softswitch WSDL Code” section on page 2-6](#) for the full text of the bts10200.wsdl file. This file contains all objects that are defined for use in the Cisco BTS 10200 Softswitch. The breakdown of each object is listed below.

- **Bts10200_Security**—The primary security object. It is used to create login keys for use in another object. This object is required to access the Cisco BTS 10200 Softswitch.
- **Bts10200**—The basic object used to retrieve XML description documents as well as provisioning and control documents.
- **BtsSoapException**—The object used to report all errors in the Cisco BTS 10200 Softswitch SOAP interface.
- **Mgp**—The object used to communicate to certain media gateways. It uses simple strings to contain MGW-compatible text commands.

Cisco BTS 10200 Softswitch API

This section covers the actual API calls to the SOAP interface. The assumption is that the client application is developed in the Java language. This does not prohibit the use of C++. However, that is not within the scope of this document.

All parameters that are listed are required for each invocation of methods in the associated object.

Cisco BTS 10200 Softswitch Security

The Cisco BTS 10200 Softswitch security object (**Bts10200_Security**) provides a user several levels of security for the SOAP interface. It allows authorized users to obtain a security key and use this key for all future transactions. This object must be used prior to all other SOAP method invocations in the interface. This key is valid in the SOAP interface for the life of the user's session. The key is no longer valid once the logout method has been invoked. Likewise, the security key expires after 10 minutes if the system has not been accessed during that period of time, and the user is automatically logged out of the SOAP interface. The user name and password are the same values allowed in the CLI/MAC adapter interfaces, and the same authorization permissions apply.

Each method in this section is part of the **Bts10200_Security** interface. The parameters listed are required for each method and must contain data.

Login

The login method provides authentication of a SOAP interface user. It utilizes the same user security as the FTP or CLI adapters. This method returns a string value defined as a key. This key is required for all transactions against the SOAP interface. It is an authentication key to indicate the specific authorization of a particular user. The method signature is defined using the following code:

```
<wsdl:operation name="login">
    <wsdl:input message="impl:loginRequest" name="loginRequest"/>
    <wsdl:output message="impl:loginResponse" name="loginResponse"/>
    <wsdl:fault message="impl:BtsSoapException" name="BtsSoapException"/>
</wsdl:operation>
```

- **Return value**—Status indicating success or failure of the operation. A failure indication means the facility is unavailable. A successful return means the operation was completed.

- **Exception**—A `BtsSoapException` means there is an operational error in processing the request. This includes faults with the parameter types, ranges, and database access.

Logout

The `logout` method terminates a login session. This destroys the validity of the authentication key. Once this method is complete, the key can no longer be used for other method invocations. The method signature is defined via the following code:

```
<wsdl:operation name="logout">
    <wsdl:input message="impl:logoutRequest" name="logoutRequest" />
    <wsdl:output message="impl:logoutResponse" name="logoutResponse" />
    <wsdl:fault message="impl:BtsSoapException" name="BtsSoapException" />
</wsdl:operation>
```

- **Return value**—Status indicating success or failure of the operation. A failure indication means the facility is unavailable. A successful return indicates the operation was completed.
- **Exception**—A `BtsSoapException` means there is an operational error in processing the request. This includes faults within the parameter types, ranges, and in database access.

Bts10200 Provisioning API

The Cisco BTS 10200 Softswitch object (`Bts10200`) provides provisioning interface functions to the Cisco BTS 10200 Softswitch CLI engine for authorized users. Both input and output are in XML as described in [Chapter 2, “Extensible Markup Language Processing”](#). The CLI commands are parsed into an XML document before sending it through the SOAP interface. The SOAP adapter web service then executes the CLI provisioning commands and sends back the reply in an XML document. Each method in this section is part of the `Bts10200` interface. The parameters listed are required for each method and must contain data.

getCommandDoc

The `getCommandDoc` method provides command description retrieval. This method obtains the XML document describing the command syntax and options for a specific noun/verb combination. The method signature is defined using the following code:

```
<wsdl:operation name="getCommandDoc">
    <wsdl:input message="impl:getCommandDocRequest" name="getCommandDocRequest" />
    <wsdl:output message="impl:getCommandDocResponse" name="getCommandDocResponse" />
    <wsdl:fault message="impl:BtsSoapException" name="BtsSoapException" />
</wsdl:operation>
```

getExtCommandDoc

The getExtCommandDoc method provides command description retrieval. This method obtains the XML document describing the command syntax and options for a specific noun/verb combination. The method signature is defined using the following code:

```
<wsdl:operation name="getExtCommandDoc">
    <wsdl:input message="impl:getExtCommandDocRequest"
name="getExtCommandDocRequest" />
    <wsdl:output message="impl:getExtCommandDocResponse"
name="getExtCommandDocResponse" />
    <wsdl:fault message="impl:BtsSoapException" name="BtsSoapException" />
</wsdl:operation>
```

request

The request method processes an XML document based provisioning request through SOAP interface. The SOAP web service executes the provisioning commands and sends back the reply in an XML document. The method signature is defined using the following code:

```
<wsdl:operation name="request">
    <wsdl:input message="impl:requestRequest" name="requestRequest" />
    <wsdl:output message="impl:requestResponse" name="requestResponse" />
    <wsdl:fault message="impl:BtsSoapException" name="BtsSoapException" />
</wsdl:operation>
```

BtsSoapException

The following basic BtsSoapExceptions can be returned. The numbers given in the sample code refer to the text in the explanation. The text is returned if the sample code is used.

```
public static final int CLIENT_LOGIN_FAIL = 401;
public static final int CLIENT_LOGOUT_FAIL = 402;
public static final int CLIENT_ACCESS_DENIED = 404;
public static final int CLIENT_SESSION_INVALID = 406;
public static final int CLIENT_SESSION_IN_USE = 407;

public static final int CLIENT_BAD_REQUEST = 410;
public static final int CLIENT_INVALID_VALUE = 411;
public static final int CLIENT_SIDE_UNKNOWN = 499;

/** server side exception block */
public static final int ADAPTER_BLOCKED = 501;
public static final int ADAPTER_INITING = 502;
public static final int ADAPTER_STOPED = 503;
public static final int ADAPTER_FAULT = 504;
public static final int ADAPTER_UNKNOWN = 505;

public static final int BTS_STANDBY = 511;
```

```

public static final int BTS_FAULT = 512;
public static final int BTS_UNKNOWN = 513;

public static final int SESSION_OVERLOAD = 520;
public static final int RESOURCE_ALO_FAIL = 530;
public static final int CMD_NOT_SUPPORT = 540;

public static final int SERVER_SIDE_UNKNOWN = 599;

```

Understanding the SOAP Session Manageability Feature

The SSM feature enhances the manageability of user sessions. It impacts four areas:

1. SOAP Software Developer's Kit (SDK)
2. SOAP Interface Servant (SOAP)
3. Session and System Manager (SMG for EMS)
4. User Security Manager (USM for EMS)

SOAP session policy is a timer-controlled process to remove the policy violated sessions.

Software Developer's Kit

There are two login APIs in the SDK client code. The first API is *loginWithStatus* that returns password aging status. The other API is *loginResetPassword* that resets a new password when an old password is aged.

The SOAP SDK has the following components:

- *ResetPassword.java*—A driver program to utilize the login, reset password, and logout functions.
- *SoapProvAdapter.java*—Adds access to the password reset API for SOAP, and provides external indications and API for driver logic.

SOAP Interface Servant

This feature impacts the SOAP Interface Servant application. User security is controlled by *UserSessionManager* and *UserAuth* objects. User security is the location of password validation and tracking user attributes such as idle login and security keys.

The user security information must have its data externalized through an API. Through this API, queries are available to take snapshots of the present condition of sessions. This information is through database statistics tables in the MySQL database.

Additional message handler functions add an on-demand reporting of session information and acceptance of command requests to terminate the sessions. This interface utilizes the user security API.

The Session Control Policy is handled in a minute-based looping process to screen and remove sessions that match a record in the policy. Policy management is supplied in a new CLI command.

The *bts.properties* file contains the Maximum User Limit (item name: *max.users*) and Idle Timeout (item name: *idle.timeout*) that can be modified manually. The SOAP adapter relies on these numbers to decide if: (1) the maximum user limit is reached, or (2) the user session is idle timeout. The SOAP adapter dynamically reads the file upon user login and during session audit. If the maximum number of users is set to a value higher than 100, the hard limit of 100 maximum users applies.

**Note**

Modification of the `bts.properties` file should not be attempted without Cisco TAC support or supervision.

Session and System Manager

The SOAP Manageability feature has the following capabilities:

- **User Session Display**—Display current secure and nonsecure SOAP sessions using the new “show client-session” and “report client-session” commands. The returned data also includes any current CLI sessions.
- **Manual Session Removal**—Remove a SOAP session or a CLI session using the “stop client-session” command. The present “stop session” command applies to CLI users only. Additional information clearly indicates individual sessions.
- **Policy-driven Smart Session Management**—Includes the smart removal of idle sessions allowing new sessions to login, while allowing administrative access at all times. This does not effect idle time. The maximum duration of a session is set whether a session is idle or not. Default idle time is 10 minutes.
- **Password Aging Notification**—Aging notification of the password for a given user when *loginWithStatus* API is used.
- **SOAP Password Reset**—Users can login and reset aging password using the *loginResetPassword* API. If a password expires, access is denied until the password is reset.
- **Disable Password Aging**—Set passwords to never expire when adding a new user or using the “change user” command. Set the status token to PERSIST.
- **Alarms and Events for Critical Session Handling**—Issue warning and major alarms and events when the session threshold of usage is reached. Issue an alarm when the maximum login sessions is reached. Issue an event when a user session is terminated because of a policy violation. This behavior is managed through the Policy table.

User Security Manager

The User Security Manager (USM) has a new status. This status disables the password aging function. The status has the following attributes:

- **DISABLED**—The user account is locked out and the user cannot access the system.
- **ENABLED**—The user is active and current for all attributes including password aging.
- **PERSIST**—The user account has no password aging.

