Cisco BTS 10200 Softswitch CORBA Adapter Interface Specification Programmer Guide

Release 5
February 16, 2007

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Text Part Number: OL-12438-01

CONTENTS

Preface

The Cisco BTS 10200 Softswitch Release 4.5.1 CORBA Adapter Interface Specification Programmer Guide describes the CORBA adapter (CAD). CAD provides a machine-to-machine interface (MMI) over Common Object Request Broker Architecture (CORBA). This architecture is defined by the Object Management Group (OMG) organization.

The the CAD interface provides a provisioning method for the Cisco BTS 10200 Softswitch product that parallels the Command Line Interface (CLI) adapter in capabilities. CAD provides an abstraction of the Cisco BTS 10200 Softswitch in a consistent, object-oriented model. Discussion of the actual object model for this interface is not within the scope of this document.

The *Cisco BTS 10200 Softswitch Command Line Interface Reference Guide* is the definitive source for token descriptions (parameters) and their values, as used in the CORBA interface.

Note This document describes CORBA using OpenORB. If you are using Visibroker, use the *Cisco BTS 10200 Softswitch Release 3.2, 3.3, 3.5 Programmers Specification*.

Feature History

Release	Modification
5.0	No change.
4.5.1	Compliant with Release 2.4; added information to the Connections and Transactions section; added information to Batch Data Retrieval section; new information regarding architecture & design; added Proxy chapter and additional troubleshooting information. See Appendix C, "Sample CORBA Client Package (BTSxsdk) Implementation" for additional Software Development Kit (SDK) enhancements.
4.5	Added new Appendix C. Changed Openorb 1.3.0 to 1.3.1, including use in jar filenames.
4.4.0/1	Added CORBA Secure Socket Layer Support, Batch Data Retrieval (Paging), and XML Interface information. Updated the BTS 10200 IDL. Added new Chapters 4 and 5 on security and troubleshooting.
4.1	No change.
3.5.2	OpenORB changes were incorporated.
3.5	Macro command was introduced.
3.3	CORBA enhancements were introduced.

1.0	CORBA was introduced.
All	Added Invalid Character table.

Audience

This document is designed for intermediate and advanced CORBA programmers. It presumes an intermediate or higher level of CORBA programming familiarity.

Interface

The CORBA Adapter (CAD) uses the OpenORB 1.3.1 interface to develop and deploy distributed object-based applications, as defined in the CORBA specification 2.4.

XML Interface

The XML interface is abstracted from CORBA itself. The OpenORB package uses the Internet Inter-ORB Protocol (IIOP) using either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP) for connections. Narrowing on the NameService also produces the BTS10200 objects. Narrowing is covered in great detail in the coding examples in the Cisco BTS 10200 Softswitch SDK package. This is bundled with the Cisco BTS 10200 Softswitch application.

Creating separate application-level connections or objects requires some object pooling and numerous logins to obtain valid login keys for every instance of the BTS10200 object. Examples are available in the bundled SDK package.

Batch Data Retrieval (Paging)

Batch data retrieval (paging) is available using the CLI **show** command. The use of paging is required for viewing large data sets, but the initial request to set up paging impacts performance. Subsequent requests against the same paged data are faster because the data is cached for quick retrieval. This applies to a specific session (key) and that any other command executed against that session flushes any cached data. All requested paged data must be contiguous for optimal performance. For more information, see the *Cisco BTS 10200 Softswitch Operations, Maintenance and Troubleshooting Manual* and the *Cisco BTS 10200 Command Line Interface Reference Guide*.

Note For the limit token, in any single show, CORBA automatically limits paging to 500 rows.

Connections and Transactions

CORBA supports up to 50 simultaneous sessions. A session is any valid login to the CORBA interface. CORBA also supports 20 concurrent transactions. A transaction is any specific request; for example, show or change.

Each login session allocates its own specific set of resources much like an individual CLI. These sessions are audited for idle activity. If the session is not active (executes a command) within a 10 minute period, then the session is declared idle, removed from the interface, and all resources are closed.

Note The idle time has changed from previous releases. The older default was 30 minutes; it is now 10 minutes. Idle time is configurable through the bts.properties file of the CIS application. This properties file is located on the Cisco BTS 10200 Softswitch in the /opt/BTScis/etc directory.

Obtaining Documentation, Obtaining Support, and Security Guidelines

For information on obtaining documentation, obtaining support, providing documentation feedback, security guidelines, and also recommended aliases and general Cisco documents, see the monthly *What's New* in Cisco Product Documentation, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

CHAPTER 1

CORBA Architecture and Application Programming Interface

This chapter describes the Common Object Request Broker Architecture (CORBA) adapter architecture and application programming interface (API) for the Cisco BTS 10200 Softswitch.

CORBA Adapter Architecture

The CORBA adapter (CAD) 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. Figure 1-1 illustrates the overall architecture for the CAD and shows the CORBA architecture.

The architecture provides dual mode support for secure and nonsecure CORBA, which are active at the same time. This was an install option in previous releases. The non-secure mode and the secure mode are fully supported. As part of this dual mode support, there are two java processes on the Cisco BTS 10200 Softswitch EMS that manage the CIS application. Each process is marked with a unique name to indicate its unique function.

- Nonsecure is *-D_CIS_IIOP*
- Secure is *-D_CIS_SSLIOP*

There are side effects to this dual mode of support. When both modes are active, the new secure CORBA mode does not behave as it did in previous releases. This new behavior includes name space collisions, so a new secure POA context and name space is provided to avoid problems.

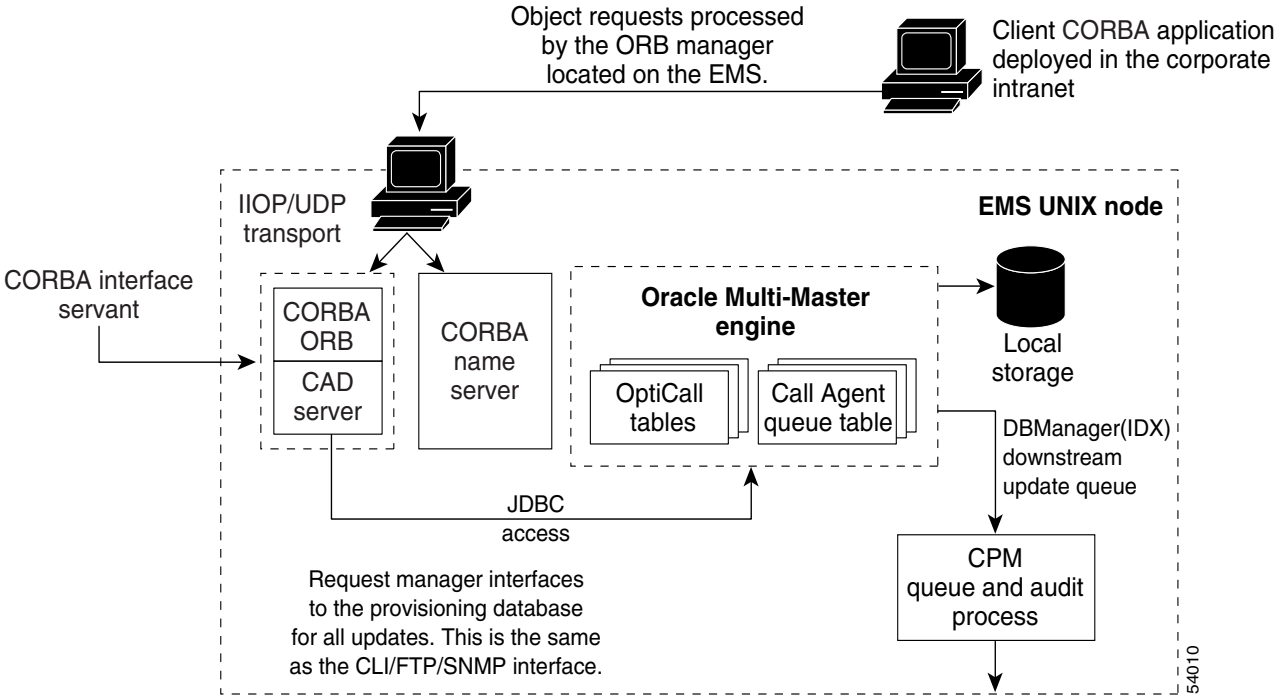
Both the non-secure mode and the secure mode only allow connections on the active EM01 EMS application and automatically drop connections if the active EM01 application fails or manually switches over to its redundant mate. It will, in addition, remove its objects from the local INS or NameService so that no new queries can successfully resolve to that particular EMS. This solves a legacy CORBA issue by preventing any provisioning from a standby EMS.

The CORBA Installation automatically uses VIP (Virtual IP Address) as the iiop.hostname, if the VIP is configured. If VIP is not configured, the first EMS Management IP address is used as the iiop.hostname. This allows the NameService to listen to all IP addresses on the active EMS. For the use of VIP, please refer to the *Cisco BTS 10200 Softswitch CLI Reference Guide, Operations, Administration and Maintenance*.

Installing CORBA also installs the CORBA SDK onto the EMS. The installer can run the sample test program to verify the CORBA installation. See Appendix C, "Sample CORBA Client Package (BTSxsdk) Implementation" for more information.

The bts.properties file in the CIS application affects client application development. 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.

Figure 1-1 CORBA Architecture



ORB Specifications

The Object Request Broker (ORB) used in the CAD interface is the OpenORB 1.3.1 compliant package. The ORB also supports other advanced features like the portable object adapter (POA). POA is the implementation model used in CAD.

Compiler Tools

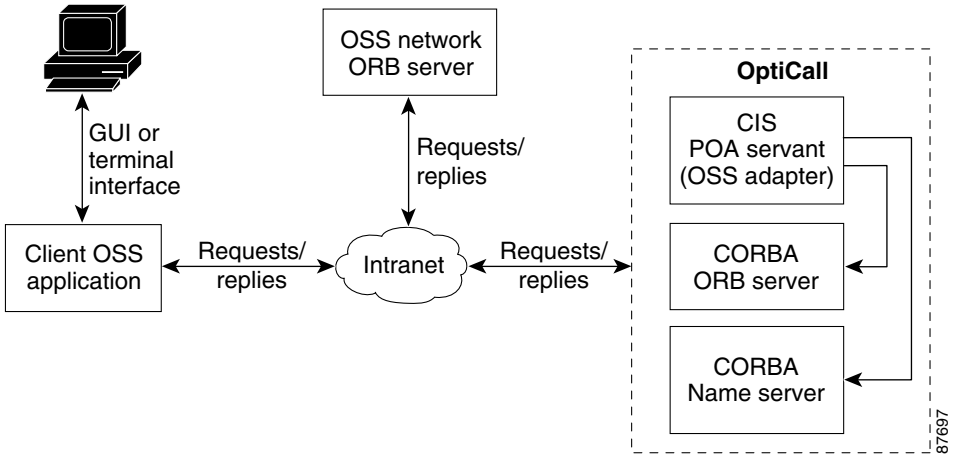
The minimum required compiler is the J2SE Development Kit (JDK)1.4.1. The Cisco BTS 10200 Softswitch uses JDK1.5.0 or JDK5 for compilation and for the Java Runtime Environment (JRE). Additional tools that may be required for the client side application are:

- Xerces parsers
- ECS Report Builder

ORB Deployment

Figure 1-2 shows the ORB deployment process.

Figure 1-2 ORB Deployment



When using OpenORB for the client side application, a few basic steps must be performed to ensure that the client environment is properly set up and ready for the application. The first is to configure the JVM on the client machine to use OpenORB as the primary ORB for Java. This can be done at runtime or it can be permanently set in the orb.properties of the JVM. To perform the latter requires privileges of the owner of the JVM install. In most cases this is root. The root user must execute the following command:

```
java -jar openorb_1.3.1.jar
```

This places the orb.properties settings in the correct location with the following values.

```
org.omg.CORBA.ORBClass=org.openorb.CORBA.ORB
org.omg.CORBA.ORBSingletonClass=org.openorb.CORBA.ORBSingleton
```

Otherwise, the settings must be supplied as environmental overrides to each invocation of the client application.

Additional values that are of use to the client programmer are the OpenORB DEBUG options. These control the volume of debug information produced by the client application. The debug information is sent to the standard output of the application. If a programmer wishes to capture this data into log files, use the basic shell redirect commands to redirect the data. Other environments such as web services can behave differently. The options for getting the highest degree of debug output are listed below. These produce IIOP/SSLIOP message dumps which are the most useful in debugging issues with communications with the Cisco BTS 10200 Softswitch.

```
-Dopenorb.debug.trace=DEBUG -Dopenorb.debug.level=HIGH
```

These options are supplied as an environment setting to the client application as part of the Java invocation. The following examples illustrate the interface definition language (IDL) compilation and Java code required for locating the POA and binding it to the compiled IDL objects.

This OpenORB generic process is a critical part of the development of any client application interfacing to the Cisco BTS 10200 Softswitch. The IDL supplied by the Cisco BTS 10200 Softswitch must be compiled into interface classes and then used in the client application.

Note IDL is a generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language. In distributed object technology, new objects must discover how to run in any platform environment to which they are sent. An ORB is a middleware program that brokers client/server relationships between objects.

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 IDL interface objects. See the SDK for a detailed breakdown of this script.

```
#!/bin/sh
###############################################################################
# Copyright (c) 2002, 2006 by Cisco Systems, Inc.
#
# AUTHOR: A. J. Blanchard
#
# DESC: Invoke the IDL compiler for the OpenORB package.
#
###############################################################################
set -e
set -a
#set -x

#
# List required jar files
#
CLASSPATH=./opt/BTSoorb/lib/logkit.jar:/opt/BTSoorb/lib/openorb-1.0.1.jar:/opt/BTSoorb/lib
/openorb_tools-1.3.1.jar:/opt/BTSoorb/lib/xerces.jar:/opt/BTSoorb/lib/avalon-framework.jar
:/opt/BTSoorb/lib/openorb_ots-1.3.1.jar:/opt/BTSoorb/lib/openorb_pss-1.3.1.jar:/opt/BTSoor
b/lib/openorb_ins-1.3.1.jar:/opt/BTSoorb/lib/openorb_tns-1.3.1.jar

export CLASSPATH

java -classpath $CLASSPATH org.openorb.compiler.IdlCompiler $1 -jdk1.4 -all -verbose -d ./
```

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.
#
# AUTHOR: A. J. Blanchard
#
# DESC: Compile Java ORB programs with the required components from OpenORB.
#
###############################################################################
set -e
set -a
#set -x

CLASSPATH=./opt/BTSoorb/lib/logkit.jar:/opt/BTSoorb/lib/openorb-1.3.1.jar:/opt/BTSoorb/lib
/openorb_tools-1.3.1.jar:/opt/BTSoorb/lib/xerces.jar:$HOME/mb/devel/em/lib/cad.jar:$HOME/m
b/devel/em/lib/ecs-1.4.1.jar:/opt/BTSoorb/lib/openorb_ots-1.3.1.jar:/opt/BTSoorb/lib/openo
rb_pss-1.3.1.jar:/opt/BTSoorb/lib/openorb_ins-1.3.1.jar:/opt/BTSoorb/lib/openorb_tns-1.3.1
.jar

export CLASSPATH

javac -classpath $CLASSPATH -d ./ $*
```

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 CORBA interface. These include many topics such as:

- CLI
- Batch file processing
- Multi-thread concurrency
- SSL

Note The example below illustrates the basic abstraction of the BTS 10200 objects. If other tools are used, you must modify the IDL objects as well as the OpenORB files.

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

import java.lang.*;
import java.io.*;
import java.util.*;
import java.text.*;
// CORBA stuff
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.Messaging.*;
// XML Stuff
import org.apache.ecs.xml.*;
import org.apache.ecs.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import org.apache.xml.serialize.*;
// BTS Code jar files...
import com.sswitch.oam.cad.*;
import com.sswitch.oam.xml.*;
import com.sswitch.oam.util.*;

/**
 * CorbaXmlIntf.java
 * Copyright (c) 2002, 2006 by Cisco Systems, Inc.
 * -- This is the client side driver stub. This allows the client application
 * to generate the Request object which is then digested in this class as a
 * XML document and sent as a request to the CORBA server. The results are
 * then returned to the user or the CORBA exception is thrown.
 *
 *   @author   A. J. Blanchard
 *   @version  4.0
 *   @since    BTS 10200 4.0
 *
 */

public class CorbaXmlIntf {

    /*
     * Class private data
     */
    private String []                               objArgs;
    private org.omg.CORBA.ORB                       objOrb;
    private org.omg.CosNaming.NamingContextExt      objContext;
```

```
    private com.sswitch.oam.cad.Bts10200            objBts;
    private com.sswitch.oam.cad.Bts10200_Security   objBtsSec;
    private org.omg.CORBA.StringHolder              objKey;

    /**
     * Generic Constructor for the test driver.
     */
    public        CorbaXmlIntf(String[] args)
    {
      // Initialize the ORB.
      objOrb = org.omg.CORBA.ORB.init(args, null);
      objArgs = args;
      return;
    }

    /**
     * This is the primary execution method for the object. It performs the
     * actual request and calls for the print of the reply.
     */
    public void       connect() throws CadExceptions
    {
      //
      // Log into the target machine with generic optiuser
      //
      try {
        bind();
        objKey = new org.omg.CORBA.stringholder();
        objBtsSec.login("btsadmin", "btsadmin", objKey);
        Log.info("BTS10200 Login successful: "+objKey.value);
        }
      catch (Exception e) {
        Log.error("Exception in CORBA Bind/Login = "+
                  Util.stackTraceToString(e));
        throw new CadExceptions(1, e.toString());
        }
    }

    /**
* This method generate the request to the CORBA server and returns
     * the reply or an exception if the interface throws an exception.
     * The argument "request" must be an XML formatted document.
     *
     * @param request    This XML request document.
     * @returns String   This is the XML formatted answer.
     */
    public String    request(String request)
        throws CadExceptions
    {
      String   answer=null;
      try {
        org.omg.CORBA.StringHolder reply = new org.omg.CORBA.StringHolder();

        // Issue request to BTS 10200
        objBts.request(request, objKey.value, reply);

        // Build an XMLReply from the document

        answer= reply.value;
        }
      catch (Exception e) {
        Log.warning("Request Command Exception:\n " +
                           Util.stackTraceToString(e));
        throw new CadExceptions(1, e.toString());
        }
      return answer;
    } // end request()

    /**
* This method generate the request for a command document to the
     * CORBA server and returns the reply or an exception if the interface
```

```
     * throws an exception.
     *
     * @param noun       This noun for the request.
     * @param verb       This verb for the request.
     * @returns String   This is the XML formatted answer.
     */
    public String    getCommandDoc(String verb, String noun)
        throws CadExceptions
    {
      String   answer=null;
      try {
        org.omg.CORBA.StringHolder reply = new org.omg.CORBA.StringHolder();

        // Issue request to BTS 10200
        objBts.getCommandDoc(noun, verb, objKey.value, reply);

        // Build an XMLReply from the document

        answer= reply.value;
        }
      catch (Exception e) {
        Log.warning("Request Command Exception:\n " +
                          Util.stackTraceToString(e));
        throw new CadExceptions(1, e.toString());
        }
      return answer;
    } // end getCommandDoc()

    /**
* This method generate the request for a command document to the
     * CORBA server and returns the reply or an exception if the interface
     * throws an exception.
     *
     * @param noun       This noun for the request.
     * @param verb       This verb for the request.
     * @returns String   This is the XML formatted answer.
     */
    public String    getExtCommandDoc(String verb, String noun)
        throws CadExceptions
    {
      String   answer=null;
      try {
        org.omg.CORBA.StringHolder reply = new org.omg.CORBA.StringHolder();

        // Issue request to BTS 10200
        objBts.getExtCommandDoc(noun, verb, objKey.value, reply);

        // Build an XMLReply from the document
        answer= reply.value;
        }
      catch (Exception e) {
        Log.warning("Request Command Exception:\n " +
                          Util.stackTraceToString(e));
        throw new CadExceptions(1, e.toString());
        }
      return answer;
    } // end getExtCommandDoc()

/**
     * This module disconnects the user from the BTS 10200 CORBA interface.
     */
    public void   disconnect()  throws CadExceptions
    {
      objBtsSec.logout(objKey.value);
      return;
    }

/*====================================================================
     * Internal processing methods...
     *====================================================================*/
```

```
    /**
     * This method binds to the target CORBA objects for us to operate
     */
    protected void        bind()
        throws org.omg.CORBA.ORBPackage.InvalidName,
               org.omg.CosNaming.NamingContextPackage.InvalidName,
               org.omg.CosNaming.NamingContextPackage.NotFound,
               org.omg.CosNaming.NamingContextPackage.CannotProceed
    {

      org.omg.CosNaming.NameComponent[] nameComponent = null;
      org.omg.CORBA.Object result = null;

      insLocate();

      result = objContext.resolve(objContext.to_name("Bts10200_Security_poa"));
      objBtsSec = Bts10200_SecurityHelper.narrow(result);

      result = objContext.resolve(objContext.to_name("Bts10200_poa"));
      objBts = Bts10200Helper.narrow(result);
      Log.info("Basic POA(s) have been located and bound.");
      return;
    }

    /**
     * Load the name service and find the context for the CORBA objects.
     * Remember, the INS must be the one located on the BTS. This has the
     * object references. Use a 'corbaloc:' for now but later a migration
     * to URL for name service location would be good.
     */
    protected void       insLocate()
      throws org.omg.CORBA.ORBPackage.InvalidName
    {
      //System.out.println("Locate NameService in system.");
      org.omg.CORBA.Object initial_context_obj =
          objOrb.resolve_initial_references("NameService");
      objContext =
          org.omg.CosNaming.NamingContextExtHelper.narrow(initial_context_obj);
      Log.info("NameService found in initial context.");
      return;
    }

} // end CorbaXmlIntf
```

- Actual implementations can make the POA selection dynamic and based on some form of navigation to a site (for example, to a softswitch home location or perhaps part of the softswitch ID). Once a POA is selected, all object implementations are the same. No site-specific behaviors are exhibited in any object. However, site-specific attributes are present, and are derived based on the local database contents.

NameService

The OpenORB NameService module provides an Object Management Group (OMG) compliant implementation of the NameService Specification Version 1.2 (September 2002). This module is required for CORBA operations on the Cisco BTS 10200 Softswitch. Clients attach to the NameService to obtain the references to the Cisco BTS 10200 Softswitch CORBA objects through the corbaloc process. When using OpenORB on the client side of the application, apply the following syntax to connect to the NameService:

```
"corbaloc::1.2@<Host Name>:14001/NameService"
```

The *corbaloc* string can be supplied in the OpenORB.xml configuration file located at /opt/BTSoorb/config directory as an initial reference or it can be dynamically built in the client application as required. Note that Cisco recommends that the hostname be an IP address.

Each Cisco BTS 10200 Softswitch EMS comes with management interfaces, and the INS or NameService must utilize a hostname that resolves across both of these management interfaces. This is required. But client side access does not care which interface is utilized. The client must be aware that a given management interface can be down for various reasons and that this can impact access to the name service. A recommendation to utilize both the IP address or retries on the hostname may be required to deal with switch or router troubles that may naturally occur over time on any given subnet.

Each Cisco BTS 10200 Softswitch comes with its own pair of duplex INS. Each INS represents the objects from a single side of the Cisco BTS 10200 Softswitch. Use this resource location string to derive a reference to the NameService. Each Cisco BTS 10200 Softswitch comes with its own instance of a name service, and a name service can be utilized separately for each EMS. The Cisco BTS 10200 Softswitch default UDP port for this module is 14001. In the OpenORB model, this value is passed in the configuration file OpenORB.xml. The Software Development Kit (SDK) contains examples of this configuration, as well as example code for building working examples using the OpenORB client implementation.

Multiple NameService modules can be used by applying a request interceptor. A proxy object allows a request to be forwarded using the ForwardRequest (CORBA 3.0 spec. 1.3.1) protocol. See Chapter 4, "Proxy" for more information.

The following example shows an object resolution using the NameService module. Note that at this time the basic POA is used as a root-level reference to the local Cisco BTS 10200 Softswitch.

```
org.omg.CORBA.Object initial_context_obj =
          objOrb.resolve_initial_references("NameService");
      objContext =
          org.omg.CosNaming.NamingContextExtHelper.narrow(initial_context_obj);
      result = objContext.resolve(objContext.to_name("Bts10200_poa"));
      objBts = Bts10200Helper.narrow(result);
```

BTS 10200 Softswitch IDL

The IDL is used to express the object-level interface in the CAD interface. This object interface includes the attributes and behaviors of the objects. This section provides an overview of the IDL for the Cisco BTS 10200 Softswitch. These IDL objects 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, CORBA acts as the transport for these XML documents.

The bts10200.idl file contains the general system-wide data structures and type definitions. It also contains the error interfaces (exceptions). See the "Cisco BTS 10200 Softswitch IDL Code" section on page 2-6 for the full text of the bts10200.idl 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.
- **CadException**—The object used to report all errors in the Cisco BTS 10200 Softswitch CAD interface.
- **Macro**—This object defines and executes custom *show* or *display* commands on the Cisco BTS 10200 Softswitch. This allows a user to create simplified display commands from complex relationships and permanently store them for recall later as "macro" commands.

Bts10200 API

This section covers the actual API calls to the CAD 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.

Bts10200 Security API

The Cisco BTS 10200 Softswitch security object (Bts10200_Security) provides a user several levels of security for the CAD interface in the Cisco BTS 10200 Softswitch. 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 CORBA method invocations in the interface. This key is valid in the CAD 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 CAD 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 CORBA 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 CAD interface. It is an authentication key indicating the specific authorization of a particular user. The method signature is defined by the following code:

```
int login (java.lang.String user, java.lang.String passwd, java.lang.StringHolder key)
throws CadExceptions
```

- **Return value**—Status indicating success or failure of the operation. Failure means the facility is unavailable. Success means the operation was completed.
- **Exception**—A CadException 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 in the following code:

```
int logout (java.lang.String) throws CadExceptions
```

- **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 CadException 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 the commands through the CORBA interface. The CORBA CIS server 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 by the following code:

```
void getCommandDoc (java.lang.String noun, java.lang.String verb, java.lang.String key,
org.omg.CORBA.StringHolder reply) throws CadExceptions
```

where:

- *parameter noun* is the command noun
- *parameter verb* is the command verb
- *parameter key* is the authorization key obtained in login
- *parameter reply* is the XML reply of the command syntax and options
- *exception* (cadexception) means there is an operational error in processing the request. This includes faults with the parameter types, ranges, and database access.

request

The request method processes an XML document based provisioning request through CORBA interface. The CORBA CIS server executes the provisioning commands and sends back the reply in an XML document. The method signature is defined by the following code:

```
void request (java.lang.String command, java.lang.String key, org.omg.CORBA.StringHolder
reply) throws CadExceptions
```

where:

- *parameter command* is the provisioning command request in the XML document
- *parameter key* is the authorization key obtained in login
- *parameter reply* is the XML reply of the command execution in the CLI engine
- *exception* (cadexception) means there is an operational error in processing the request. This includes faults with the parameter types, ranges, and database access.

Macro Command

The Cisco BTS 10200 Softswitch requires the ability to provide a high-level view of complex data in its database. Normally, determining relationships between database items requires several commands and multiple requests to the database. This slow and costly process impedes the progress of Operations Support System (OSS) management systems. Therefore, an optimized approach is required where several operations can be collapsed into a single request to the database, which returns the correct related data based on the set of defined rules.

The ability to view complex data relationships in the Cisco BTS 10200 Softswitch is referred to as a macro. A macro is a single command that builds complex queries across multiple commands by using relationship rules defined by the user.

The Macro command is specific to the Cisco BTS 10200 Softswitch and works with both simplex and duplex configurations. The primary focus of this command in this document is its utilization in the CORBA interface definition language (IDL) interface of the CORBA Adapter (CAD) feature.

Note The Macro command was available in Release 3.5. It was not available in Releases 3.2 and 3.3.

Behaviors and Attributes

The Macro command interface allows users to select and define multiple tables against the Cisco BTS 10200 Softswitch database. Normal commands only operate on single devices and/or tables.

The Macro command interface does not allow users to write to the tables in the database. There are many rules and constraints that apply to the database tables that prevent this activity.

Macro Command Management

Macro command management is composed of the user commands that create, change, and delete Macro command definitions. These user commands allow the definition and manipulation of the actual Macro commands and execute as standard provisioning commands. The other primary component is the macro execution. This is provided through the CORBA interface and uses the macro command management rules with additional user-specified data to return the instance values of the macro parameters.

Macro Definition

The values that are used to build a Macro command are validated internally. Technically, each macro is a superset **show** command of multiple nouns and their associated parameters. All values used in the creation of the Macro command are derived from the parameters of nouns and not from the actual table and column names. This helps to preserve the abstraction over the Cisco BTS 10200 Softswitch schema. The following examples of Macro command management show typical creation, alteration, and deletion of a Macro command. The sections following the examples define the actual values used in the macro definition and the constraints imposed on them.

```
add macro id=CTXG_NUMBERS; \
  parameters=office_code.NDC, \
             office_code.EC,\
             dn2subscriber.DN,\
             subscriber.CTXG_ID; \
  rules="office_code.OFFICE_CODE_INDEX=\
       dn2subscriber.OFFICE_CODE_INDEX,\
       dn2subscriber.SUB_ID=subscriber.ID";
```

When editing an existing macro, you must enter the parameter to be modified. In this case, the *parameters* and *rules* (and, or) are a list. The entire list must be reentered. The list that is stored in the macro database entry is then replaced. The following example demonstrates this.

```
change macro id=CTXG_NUMBERS; \
   parameters= office_code.NDC, \
               office_code.EC,\
               dn2subscriber.DN,\
                           subscriber.ID;
```

When a macro is no longer required, it can be removed. This is achieved through the **delete macro** command. The only required parameter is the macro ID. All associated definitions for the macro are then removed from the database. The **delete macro** command takes following format:

```
delete macro id=CTXG_NUMBERS;
```

To display a macro, only the macro ID is required. All static components of the command are returned. The following example shows the format of this command.

```
show macro id=CTXG_NUMBERS;
```

Macro ID

The macro ID is used as the handle for all references to a macro definition. If the macro is to be invoked, the ID becomes the noun by which the macro can be invoked. The ID is also the primary key or parameter used in the **change macro, delete macro, or show macro** commands for editing a particular definition. The ID is a character field that must be unique to all other macros. It can be up to 79 characters long. The macro ID provides a unique macro definition that can also be verbose enough to describe the desired operation.

Parameter List

The parameter list is used to list the nouns and parameters that are displayed in the Macro command. They constitute the selected items to place in the reply. The value of the parameter list field is designed to be a comma-separated list of nouns and parameters from related Cisco BTS 10200 commands. This list has the following form as input:

```
parameters=<noun.parameter>,<noun.parameter>,…,<noun.parameter>;
```

Each item in the list takes the form of the noun with a period followed by the parameter from that noun. These nouns and parameters are validated through the Element Management System (EMS). Each item in the list must always be addressed in full with the noun and parameter. This is due to the reuse of common parameter names such as ID.

No implied order of importance is given to the parameters. They are supplied to the command processing in the order they are defined in the macro. No additional data items from other parameters of a given noun are included.

Rules

The most critical parts of the Macro command are the rules. These rules manage the relationship of the data to be selected from the Cisco BTS 10200 Softswitch database. These rules amount to the “where” clause of a database selection statement. They help focus the data subset required for operations. There are two basic sets of rules that can be applied to a macro: the **and** rules and the **or** rules as well as **equivalence** and **not**. The generic parameter rules are applied as **and** rules. The definitions of these rules are:

- **And rules**—One or more sets of data qualifiers indicating a required conditional for the selection of data in the macro. A single **and** rule specifies that a particular noun and parameter must equal some other specific noun and parameter.
- **Or rules**—One or more sets of data qualifiers indicating an optional conditional for the selection of data in the macro. A single **or** rule specifies that a particular noun and parameter can equal some other specific noun and parameter.

- **Equivalence and Not**—The rules section of the Macro command can contain two variations. They can describe a noun/parameter pair as equivalent or not equivalent. The common syntax for this expression is "=" for equal, or "!=" for not equal.

User Input

The user input component is not required for the rules list. This is supplied at the time the macro is invoked. This additional input is treated as an **and** rule input. This data is intended to be the qualifying data that defines what subset of data to select. This rule data can use the equivalence syntax to express variations in the data selection. For example, a user may want to find all subscribers that do not have a specific feature.

CORBA/XML Interface

The CORBA interface servant (CIS) must support a new IDL interface for the Macro command interface. This involves the use of the following new components. The CIS subsystem supports the definition and execution of the Macro commands.

- **Macro command definition**—Macros that can be defined through the "Bts10200.request(…);" interface as normal provisioning requests are managed.
- **IDL interface definition**—A new IDL method is added to access a Macro command that is separate from the standard feature-provisioning interface. This is for the execution of the macro only.
- **CIS implementation of the IDL interface**—An implementation that follows the standard behavior of other CIS interface objects by accepting strings for the authorization key and input arguments. It also returns a string to indicate the response to the macro execution. The arguments and format are described as follows:
  - **Request**—The XML Request document that contains the additional user-supplied rules for the request as well as the paging facility parameters. The XML Request document must also have the noun key set to the desired macro name to execute the request. At this time, the verb key is not used. As a default, *show* is the best option. This avoids future conflicts if writes are allowed.
  - **Key**—A simple string object that contains the actual authentication key provided through the Bts10200_Security interface.
  - **Reply**—The XML Reply document that contains the returned data from the macro execution. This follows the same format as the Reply XML document from the Bts10200 interface.

External Interfaces

This section details the extensions provided in the Cisco BTS 10200 software that are reflected in the external interface of the CORBA adapter (CAD) for the Cisco BTS 10200 Softswitch. The paging parameters are available for Macro command execution. These include the limit and start-row parameters.

CAD Interface

This section describes the new IDL method signature in the CAD interface.

```
interface Macro {

      //-----------------------------------------------------------------
      // Issue a Macro Command XML document
      //-----------------------------------------------------------------
      void          execute(in  string                  request,
                            in  string                  key,
                            out string                  reply)
                    raises(CadExceptions);

      };  // end Macro
```

Request Format

The following example shows a macro XML Request document, which requests execution of the CTXG_NUMBERS example that was defined in the "Macro Definition" section on page 1-13. The additional paging parameters are added. These limit the volume of data that is returned during any single reply. Large XML documents fail in the interface. The current default limit is defined as 500 records per request.

```
<Request Noun="CTXG_NUMBERS" Verb="show">
    <Entry Key=" subscriber.ctxg_id" Value="rcdn_grp"/>
    <Entry Key="limit" Value="1"/>
</Request>
```

Reply Format

The following is an example of a XML Reply document. This reply is based on the example in the Request Format section.

```
<Reply id="Reply">
   <Status>true</Status>
   <Reason>Success: Entry 1 of 5002 returned.</Reason>
   <Size>1</Size>
   <AbsoluteSize>5002</AbsoluteSize>
   <StartRow>1</StartRow>
   <DataTable>
       <Row id="0">
           <Column id="NDC">601</Column>
           <Column id="EC">227</Column>
           <Column id="DN">1013</Column>
           <Column id="CTXG_ID">rcdn_grp</Column>
       </Row>
   </DataTable>
</Reply>
```

Operations

This section describes changes to the operations user interface as a result of the Cisco BTS 10200 Macro command. The standard paging parameters are available for the Macro command execution. These apply to large data sets.

You can use the operator interface for additional commands to manage the Macro command in the Cisco BTS 10200 Softswitch. These commands are available from the CLI interface. In addition, these same commands are also available from the CORBA and bulk-provisioning interface.

Table 1-1 shows the user commands that can be generated. [R] in the table indicates *required*. These are user-defined commands that have a variety of noun (id) and parameter combinations.

Table 1-1 User-Defined Macro Commands

Noun	Verb	Options	Description
macro	add	id [R]	Identifier for the macro. The identifier can be from 1 to 79 characters. It must be unique to all other IDs.
macro	add	parameters [R]	Comma-separated list of the actual data to return as a result of the macro execution.
macro	add	rules [R]	Criteria for the display of data in the macro.
macro	change	id [R]	Identifier for the macro.
macro	change	parameters	Comma-separated list of the actual data to return as a result of the macro execution.
macro	change	rules	Criteria for the display of data in the macro.
macro	show	id [R]	Defines the macro to be displayed.
macro	delete	id [R]	Identifier for the macro.

Cadexceptions

The following basic cadexceptions 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. See the Cadexceptions section in Chapter 5, "Troubleshooting" for recommended actions if necessary.

Error Message `No Error`

Explanation This is a placeholder since zero is not an error.

Sample Code **public static final int EM_NONE=0;**

Error Message `CIS Error`

Explanation This error is used for internal processing errors that may relate to ORB interaction or other runtime exceptions.

Sample Code **public static final int EM_ERROR=1;**

Error Message `CIS No Data`

Explanation This error indicates that there was no data to return from a **show** command. This may not be a "real" error; however, it is cleaner to throw an exception than a NULL object.

Sample Code **public static final int EM_NODATA=2;**

Error Message `User Security Error`

Explanation A fault was found in the user security. This could result from an invalid login through a password or username. This could also result from an internal error in the security system that failed to validate the user identity.

Sample Code **public static final int EM_USERSEC=5;**

Error Message `Permission Error`

Explanation A command was attempted that failed the authorization tests for that command. The user does *not* have permission to execute this command.

Sample Code **public static final int EM_PERMISSION=6;**

Error Message `Error Message: Block Error`

Explanation All provisioning on the switch has been blocked. The command may have been perfectly well formed and the connection is still valid. This just indicates the BTS 10200 is in a maintenance mode.

Sample Code Sample Code: public static final int EM_BLOCK=7;

Error Message `Linkage Error`

Explanation The linkage failed.

Sample Code public static final int EM_LINKAGE=10;

Error Message `Exception In Initializer Error`

Explanation The initialization provoked by this method fails.

Sample Code public static final int EM_INIT=11;

Error Message `Class Not Found Exception`

Explanation The class cannot be located.

Sample Code public static final int EM_NOTFOUND=12;

Error Message `Illegal Access Exception`

Explanation The class or initializer is not accessible.

Sample Code public static final int EM_ACCESS=13;

Error Message `Instantiation Exception`

Explanation The class represents an abstract class, an interface, an array class, a primitive type, or void; or if the instantiation fails for some other reason.

Sample Code public static final int EM_INSTANCE=14;

Error Message `Security Exception`

Explanation There is no permission to create a new instance.

Sample Code public static final int EM_SECURITY=15;

Error Message `Invalid Request Exception`

Explanation The request is not valid or cannot be initialized.

Sample Code **public static final int EM_REQUEST=16;**

Error Message `Invalid Noun Exception`

Explanation The command noun is not found or is invalid.

Sample Code **public static final int EM_NOUN=17;**

Error Message `Invalid Verb Exception`

Explanation The command verb is not found or is invalid.

Sample Code **public static final int EM_VERB=18;**

Error Message `SQL Exception`

Explanation The database cannot be accessed, if the constraints are violated, if there is another table conflict, if there is a resource issue, or any other Oracle-related cause.

Sample Code **public static final int EM_DATABASE=19;**

Error Message `Invalid Value Exception`

Explanation A parameter value exceeds the valid range or some other restrictions like text length, pick-list, and so forth.

Sample Code **public static final int EM_VALUE=20;**

Error Message `Invalid Key Exception`

Explanation An invalid key or token was used to describe some data value.

Sample Code **public static final int EM_KEY=21;**

Error Message `Missing Parameter Exception`

Explanation One or more required parameters were not included in the command parameter data.

Sample Code **public static final int EM_PARAM=22;**

CHAPTER 2

Extensible Markup Language Processing

This chapter describes the Extensible Markup Language (XML) process in the Common Object Request Broker Architecture (CORBA) adapter.

XML and Components

Along with XML, the primary component the CORBA adapter is the CORBA interface servant (CIS) Java program. In addition, there are dependencies on related components in the managed object (MO) Java package. The required packages are as follows:

- Apache XML
- Xerces (parser)
- ECS (XML Document Building Tool Kit)

XML and the CIS Java packages are central, functional components in the Cisco BTS 10200 Softswitch. There is no larger object model applied to the Softswitch. A larger object model is deferred until a Cisco Systems standard model is created. This model covers applications for packet telephony for a variety of different applications.

XML in the CORBA Interface Servant

This section describes how to use XML in the CIS. Terms used in this section follow those used in XML specifications. This is to avoid confusion in the use of terms such as element, subelement, and attribute.

CIS Functions

Schemas are provided for client-side verification of the XML document structure. These schemas cover the following items:

- *ManagedObject*
- *Request*
- *Reply*

The schema for a *ManagedObject* follows the format listed below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

        <xs:element name="ManagedObject">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="MOAttribute" maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attribute name="Verb" type="xs:string" use="required"/>
                        <xs:attribute name="id" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>

        <xs:element name="MOAttribute">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="Required"/>
                                <xs:element ref="Type"/>
                                <xs:element ref="Default"/>
                                <xs:element ref="Width"/>
                                <xs:element ref="HelpText"/>
                                <xs:element ref="Label"/>
                                <xs:element ref="Parser" minOccurs="0"/>
                                <xs:element ref="Permitted" minOccurs="0"/>
                                <xs:element ref="Fk" minOccurs="0"/>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>

        <xs:element name="Required" type="xs:boolean"/>

        <xs:element name="Type">
                <xs:simpleType>
                        <xs:restriction base="xs:string">
                                <xs:enumeration value="single"/>
                                <xs:enumeration value="text"/>
                                <xs:enumeration value="multi"/>
                        </xs:restriction>
                </xs:simpleType>
        </xs:element>

        <xs:element name="Default" type="xs:string"/>

        <xs:element name="HelpText" type="xs:string"/>

        <xs:element name="Label" type="xs:string"/>

        <xs:element name="Noun" type="xs:string"/>

        <xs:element name="Param" type="xs:string"/>

        <xs:element name="Parser">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="JavaScript"/>
                                <xs:element ref="RegExp"/>
                        </xs:sequence>
                        <xs:attribute name="id" use="required" type="xs:string"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="JavaScript" type="xs:string"/>
```

```
        <xs:element name="RegExp" type="xs:string"/>

        <xs:element name="Permitted" type="xs:string"/>

        <xs:element name="Width" type="xs:int" />

        <xs:element name="Fk">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="Noun"/>
                                <xs:element ref="Param"/>
                                <xs:element ref="Fk" minOccurs="0"/>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

The schema for a *Request* follows the format listed below.

```
<?xml version="1.0" encoding="UTF-8"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
        <xs:element name="Request">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="Entry" maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attribute name="Verb" type="xs:string" use="required"/>
                        <xs:attribute name="Noun" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="Entry">
                <xs:complexType>
                        <xs:attribute name="Key" type="xs:string" use="required"/>
                        <xs:attribute name="Value" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

The schema for a *Reply* follows the format listed below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

        <xs:element name="Reply">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="Status"/>
                                <xs:element ref="Reason"/>
                                <xs:element ref="Size"/>
                                <xs:element ref="AbsoluteSize"/>
                                <xs:element ref="StartRow"/>
                                <xs:element ref="DataTable"/>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>

        <xs:element name="Status" type="xs:boolean"/>

        <xs:element name="Reason" type="xs:string"/>

        <xs:element name="Size" type="xs:integer"/>
```

```
        <xs:element name="AbsoluteSize" type="xs:integer"/>

        <xs:element name="StartRow" type="xs:integer"/>

        <xs:element name="DataTable">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="Row" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

        <xs:element name="Row">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="Column" maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attribute name="id" use="required" type="xs:integer"/>
                </xs:complexType>
        </xs:element>

        <xs:element name="Column" >
                <xs:complexType>
                        <xs:simpleContent>
                                <xs:extension base="xs:string">
                                        <xs:attribute name="id" use="required"/>
                                </xs:extension>
                        </xs:simpleContent>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

The interface definition language (IDL) allows access to XML description documents for each noun and verb combination. For example, the **add subscriber** command generates a matching XML document that defines the element and attributes of this command. The IDL allows command processing based on a well-formed but unverified XML document.

The IDL allows command access to supported media gateway (MGW) devices. The command strings do not follow the XML access format defined in the schema. The CIS supports MGW command strings that are native to the MGW internal command structure.

All XML documents that originate in the Cisco BTS 10200 Softswitch are dynamically generated. This includes all command description documents.

ManagedObject

A *ManagedObject* has one element, the *MOAttribute*. A *ManagedObject* also has two attributes: the *id* of the ManagedObject and the *verb*. The *id* represents the object on which some action is to be taken. The *verb* indicates the action to be taken. For example, subscriber, or termination is a valid id. This is a required attribute.

The following list describes the various parts of the schema and their values:

- Id—This attribute represents the object on which to take some action.
- **Verb**—This attribute defines the action to take on a given *ManagedObject*. This is a required attribute and is composed of character data.
- **MOAttribute**—The *ManagedObject* can contain none, one, or more of these elements. It has one attribute named *id*. This character data acts as a label for the element. The order of these elements does not imply any specific behavior. They are arbitrarily listed.
- **Required**—This subelement has two values defined as true or false.
- **Type**—This subelement defines whether the *MOAttribute* has a single value, multiple values, or is a text. The multiple or single options infer that a list of choices is offered in the permitted element.
- **Default**—This subelement is informational. It indicates the default value for the *MOAttribute*.
- **Width**—This subelement indicates the total field width of the data. For example, if the *MOAttribute* is a description, this indicates the length of the description.
- **HelpText**—This subelement offers a brief text to indicate the nature of the *MOAttribute*.
- **Permitted**—This subelement specifies the possible values or ranges for the *MOAttribut*e.
- **Parser**—This subelement indicates what type of validation is required. There is a single attribute to this subelement. The attribute is an *id* field constructed of character data. The subelements are listed below:
  - **JavaScript**—This subelement indicates a possible JavaScript to perform validation or regular expression matching.
  - **RegExp**—This subelement defines the regular expression in character data format.

Request

The *Request* schema consists of one element, containing none, one or more *Entry* elements, and two attributes and their values.

The following list describes the various parts of the schema and its values:

- **Noun**—This attribute defines the item on which some operation is requested. This is expressed as character data.
- **Verb**—This attribute defines the action to perform on the "Noun" attribute. This is expressed as character data.

The *Entry* element is allowed to be empty. It can also contain two attributes. These attributes are defined as follows:

- **Key**—This is the *id* value derived from the *MOAttribute* in the *ManagedObject*. It is expressed as character data.

- **Value**—This is the client-derived value to assign to the key specified above. The *Value* is expressed as character data. It should also conform to the subelements in *MOAttribute* from which this key/value pair was derived.

Reply

The *Reply* schema defines the structure of returned data generated in response to a *Request*. The *Reply* contains three elements and no attributes. These elements are defined as follows:

- **Status**—The *Reply* contains one *Status* element. It has two possible values. Either true or false is applied to this element.
- **Reason**—The *Reason* element contains character data. This element explains the cause for an error in processing a command or returns a success indication.
- **DataTable**—This element has one attribute and one subelement, which are defined below. This element is used as the container for data that results from a execution of a request. Each *Reply* can contain a *DataTable* element.
  - **Row**—This subelement defines a single complete item of data. A *DataTable* can contain one or more *Row* subelements. A *Row* has one attribute. This character data defines the row ID. The ID is always a sequential value based on the number of returned rows. The *id* attribute is required.
  - **Col**—Each *Row* contains a subelement known as a *Col*. This subelement has one attribute. The value of the element is expressed as character data. The attribute for *Col* is *id*. It is expressed as a character value. This is the same id value used in the *MOAttribute*. This is a required attribute.

CORBA Interface Servant Adapter Implementation

The CIS is an adapter implementation that specifies an external interface. This section provides more detail about the structure of the document interchange between the CIS program and a client-side program. One global issue for the external interface is that all documents covered here are defined as *well-formed* but not *verified*. This means that the schema is not an embedded part of the XML document. By embedding the schema, parser packages can be used to validate the structure of the document. However, this impedes the transition to XML schemas, should schemas be desired by other customers. The client side can still use the schema, included in this document, to perform validation.

Cisco BTS 10200 Softswitch IDL Code

This section describes the system IDL file for the CORBA adapter (CAD) interface in the Cisco BTS 10200 Softswitch. This IDL applies to Cisco BTS 10200 Softswitch, Release 4.x.

```
// Copyright (c) 2002, 2006 by Cisco Systems, Inc.
//============================================================================
//
// Name:         bts10200.idl
// Author:       A. J. Blanchard
// Description:
//  This is the IDL for the entire provisioning infrastructure of the
//  BTS 10200. The text strings are all XML well-formed documents. The
//  current procedure is to maintain separate schema(s). This allows later
//  migration to schemas and away from schema for document validation.
//
```

```
//  All commands are expressed as XML documents. The template document for
//  each NOUN/VERB pair is accessible from the a separate method. This XML
//  interface is table oriented and follows the same nomenclature and syntax
//  as the other BTS 10200 adapter interfaces.
//
//
//============================================================================

#ifndef bts10200_idl
#define bts10200_idl

//----------------------------------------------------------------------------
// Set up modules to match java package tree for the OAM&P
//----------------------------------------------------------------------------
  module cad {

   typePrefix cad "oam.sswitch.com";

    //-----------------------------------------------------------------
    // Exceptions
    //-----------------------------------------------------------------
    exception CadExceptions {
          long   error_code;
          string error_string;
          };

    interface Bts10200_Security {

      //-----------------------------------------------------------------
      // Create a session key.
      //-----------------------------------------------------------------
      void      login(in  string                name,
                      in  string                password,
                      out string                key)
                  raises(CadExceptions);

      //-----------------------------------------------------------------
      // Destroy a session key.
      //-----------------------------------------------------------------
      void      logout(in  string                 key)
                  raises(CadExceptions);

      };  // end Bts10200_Security

    interface Bts10200 {

      //-----------------------------------------------------------------
      // Fetch a command (noun/verb) XML document
      //-----------------------------------------------------------------
      void      getCommandDoc(in  string                noun,
                              in  string                verb,
                              in  string                key,
                              out string                xml_doc)
                  raises(CadExceptions);

      //-----------------------------------------------------------------
      // Fetch an Extended command (noun/verb) XML document - with foreign keys
      //-----------------------------------------------------------------
      void      getExtCommandDoc(in  string             noun,
                              in  string                verb,
                              in  string                key,
                              out string                xml_doc)
                  raises(CadExceptions);
```

```
      //-----------------------------------------------------------------
      // Issue a command XML document (add, change, delete, show)
      //-----------------------------------------------------------------
      void      request(in  string                  xml_request,
                        in  string                  key,
                        out string                  xml_reply)
                    raises(CadExceptions);

      };  // end Bts10200

interface Macro {

      //-----------------------------------------------------------------
      // Process a Macro Command XML document (add, change, delete, show)
      //-----------------------------------------------------------------
      void          execute(in  string                  request,
                            in  string                  key,
                            out string                  reply)
                    raises(CadExceptions);

      };  // end Macro

    }; // end cad
#endif // end bts10200_idl
```

CHAPTER 3

CORBA Secure Socket Layer Support

This chapter describes the Common Object Request Broker Architecture (CORBA) Secure Socket Layer (SSL) support.

The system provides a secure CORBA transport using an SSL module in the CORBA adapter CORBA interface servant (CIS). The Object Management Group (OMG) organization defines the Common Secure Interoperability Specification, Version 2 (CSIv2) defines the Security Attribute Service (SAS) protocol. The SAS enables interoperable authentication, delegation, and privileges.

The SAS protocol exchanges its protocol elements in the service context of the General Inter-ORB Protocol (GIOP) request and reply messages that are communicated over a connection-based transport. The protocol is intended to be used in environments where transport layer security, such as that available using SSL/transport layer security (TLS) or Internet Inter-ORB Protocol (IIOP) over SSL (SSLIOP), provides message protection (that is, integrity and or confidentiality) and server-to-client authentication. The protocol provides client authentication, delegation, and privilege functionality that may be applied to overcome corresponding deficiencies in an underlying transport. The SAS protocol facilitates interoperability by serving as the higher-level protocol under which secure transports may be unified. The CIS implementation of SAS provides the following:

- Secure interoperability predicated on the use of a common transport-layer security mechanism, such as that provided by SSL/TLS.
- Message protection as necessary that is provided by the transport layer to protect GIOP input and output request arguments.
- Target-to-client authentication as necessary that is provided by the transport layer to identify the target for the purpose of ensuring that the target is the intended target.
- Transport-layer security ensures that the client does not have to issue a preliminary request to establish a confidential association with the intended target.
- Support for clients that cannot authenticate by using transport-layer security mechanisms; the SAS protocol provides client authentication above the transport layer.
- To support the formation of security contexts using GIOP service context, the SAS protocol requires at most one message in each direction to establish a security context.
- Support for security contexts that exist only for the duration of a single request/reply pair.
- Support for security contexts that can be reused for multiple request/reply pairs.

This implementation is provided through a module extension supplied in the OpenORB CORBA distribution.

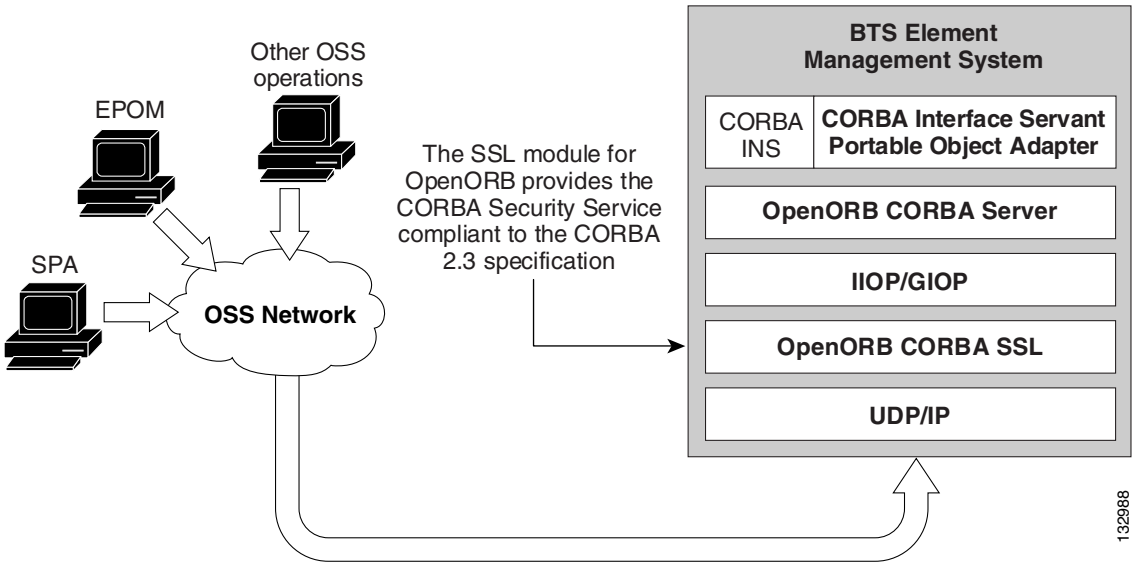
Caution Use SSL as the default transport protocol for the CORBA adapter to reduce unexpected security issues.

System Context for System Security Extensions

This section describes the system context for System Security Extensions. The System Security Extensions consists of the SSL module for CORBA combined with the exchange of security certificates.

Figure 3-1 provides an overview of the involved system components affected by the System Security Extensions. This overview relates to the affected system components in the Cisco BTS 10200 Softswitch. The exception is the base Solaris OS platform.

Figure 3-1 CORBA Implementation of Secure Sockets Layer



Dependencies

This section defines the dependencies for the application components in the Security Extension feature. The dependencies are largely based on external components.

Reduced Solaris Image

There is no specific dependency for this application component. However, the dependency is based on the security needs of the Cisco BTS 10200 Softswitch.

Java Implementation of SSL

The CIS application program depends on the OpenORB implementation of the SSL component module for the CORBA Security Support. This OpenORB module in turn requires the Java implementation of the JSSE or Java Secure Sockets from Sun Microsystems.

Certificate and Key Password

This section describes the certificate, and the key password for SSL CORBA. The primary feature of SSL CORBA is encrypted transport. This implementation uses a public-keyed, self-signed certificate only and all users have the same key password.

This section also describes the directory structure where the key store, trust store, and certificate reside for SSL CORBA and the naming conventions that must be adhered to for these files. In the Cisco BTS 10200 Softswitch, the certificate and key stores are delivered in the Cisco BTS 10200 CIS package. They are located at:

```
/opt/BTScis/cert
```

The certificate and key stores are also available in the CORBA SDK package BTSxsdk located at:

```
/opt/BTSxsdk/cert
```

The trust store and key store must be named as follows:

```
bts10200_ks bts10200_ts
```

The mandatory key password that must be used is **`Chillan`** (this is case sensitive).

SSLIOP enabled is the default during the CORBA installation using cis-install.sh. The client system cannot get in without using the aforementioned key password, key store and certificate. The key store and trust store must be on both Element Management Systems (EMSs), as there is no automatic redundancy.

Note The following commands build the key and trust stores on the Cisco BTS 10200 Softswitch and on the client side. Although they are built and deployed by default in the Cisco BTS 10200 Softswitch and in the BTSxsdk, they can be rebuilt or replaced by hand. This is not a recommended procedure at this time.

Password: Chillan

Step 1 Generate keys (validity 8 years):

```
keytool -genkey -alias bts10200 -keyalg RSA -validity 2840 -keystore bts10200_ks
```

Step 2 Export a certificate:

```
keytool -export -alias bts10200 -keystore bts10200_ks -rfc -file bts10200.cer
```

Step 3 Import the certificate into the truststore:

```
keytool -import -alias bts10200 -file bts10200.cer -keystore bts10200_ts
```

CHAPTER 4

Proxy

The Proxy software is an extension in the CORBA Software Developer's kit (SDK) package that allows several new features to easily integrate into the client application. The Proxy software is also designed to work exclusively with the new and revised SSL or secure CORBA interface to the Cisco BTS 10200 Softswitch.

The list of feature capabilities are:

- Abstraction of redundant management interfaces on a single Cisco BTS 10200 Softswitch Element Management System (EMS) node. Each EMS has two physical network interface cards (NICs) available for provisioning and the CORBA/SSL package utilizes both of these interfaces by binding object instances separately to each interface. The Proxy probes each of these interfaces and determines which interface is functioning or if both interfaces are functioning. The first interface to return the requested Cisco BTS 10200 Softswitch object is declared the primary interface and is used exclusively for transactions, The CORBA protocol IIOP/SSLIOP requires a consistent conversation over a single interface so "load sharing" on these interfaces in a given connection is not possible.
- Abstract the redundancy of the EMS from the client application. Many client applications do not fully account for the duplex nature of the Cisco BTS 10200 Softswitch and as a result must exercise some complex reinitialization if the EMS requires a manual switchover or failover. This is time intensive and prone to error. The Proxy allows the abstraction such that the objects returned from the interface fail and a new dip to the Proxy returns the corrected (newly ACTIVE) EMS objects. CORBA in the Cisco BTS 10200 Softswitch Release 4.5.1 unbinds its objects form the NameService and terminates current login sessions in the event of a failover. As a result, the Proxy continues to probe the NameService for changes on both the ACTIVE and STANDBY EMS nodes to sense a failover. Application clients detect this using a PERMISSION_DENIED error from CORBA or related CORBA COMM error. This release of the BTS 10200 does not contain a Notification Service, so there is no asynchronization notification.
- Ability to abstract multiple Cisco BTS 10200 Softswitch instances or "complexes" as well. Each complex must have a unique identifier such as a CLLI code, sensor-id, or the original CORBA site-id to determine which Cisco BTS 10200 Softswitch is being referenced. This ID is used in the Proxy logic as a unique key or locator to find the object and connection information for a particular Cisco BTS 10200 Softswitch. This allows a single instance of the Proxy to manage access to multiple Cisco BTS 10200 Softswitches (currently to several hundred). This does not preclude the use of other instances of the Proxy being used in the same network and even other instances of the Proxy talking to the same set of Cisco BTS 10200 Softswitch complexes. There is no requirement in the Proxy for mutual exclusion.
- Fully utilize both management interfaces on the EMS for redundancy. If no Virtual IP is deployed, the NameService and the CORBA application utilize both management interfaces on each EMS.

CHAPTER 5

Troubleshooting

This chapter describes basic cadexceptions, debug, and network configuration procedures.

Cadexceptions

The following basic cadexceptions 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.

Error Message `No Error`

Explanation This is a placeholder since zero is not an error.

Sample Code **public static final int EM_NONE=0;**

Recommended Action No action. Not an error.

Error Message `CIS Error`

Explanation This error is used for internal processing errors that can relate to ORB interaction or other runtime exceptions.

Sample Code **public static final int EM_ERROR=1;**

Recommended Action Retry the command. If the problem persists, restart the Name Service and CORBA Adapter as described in the section Modify the CORBA Network Configuration.

Error Message `CIS No Data`

Explanation There was no data to return from a **show** command. This may not be a "real" error; however, it is cleaner to throw an exception than a NULL object.

Sample Code **public static final int EM_NODATA=2;**

Recommended Action Verify that there is data for that particular Noun (id) by executing the same command using the CLI. Refer to the *Cisco BTS 10200 Softswitch Command Line Interface Reference Guide* for appropriate syntax.

Error Message `User Security Error`

Explanation A fault was found in the user security. This can result from an invalid login by a password or username. This can also result from some internal error to the security system that failed in validating the user identity.

Sample Code **public static final int EM_USERSEC=5;**

Recommended Action Verify that the username and password are correct and valid for access to the Cisco BTS 10200 Softswitch. Cross-check by logging into the CLI with the same username and password.

Error Message `Permission Error`

Explanation A command was attempted that failed the authorization tests for the command. The user does *not* have permission to execute this command.

Sample Code **public static final int EM_PERMISSION=6;**

Recommended Action Verify that the user has the appropriate command privilege to execute the request. Cross-check by executing the same command using the CLI as the same user. Refer to the *Cisco BTS 10200 Softswitch Command Line Interface Reference Guide* for command privilege information.

Error Message `Error Message: Block Error`

Explanation All provisioning on the switch is blocked because the Cisco BTS 10200 Softswitch is in a maintenance mode. The command may be perfectly well formed and the connection is still valid.

Sample Code **public static final int EM_BLOCK=7;**

Recommended Action Try again later.

Error Message `Linkage Error`

Explanation The linkage failed.

Sample Code **public static final int EM_LINKAGE=10;**

Recommended Action This is an installation error. Call Cisco TAC.

Error Message `Exception In Initializer Error`

Explanation The initialization method failed.

Sample Code **public static final int EM_INIT=11;**

Recommended Action Retry the command again. If the problem persists, restart the Name Service and CORBA Adapter as described in the section Modify the CORBA Network Configuration.

Error Message `Class Not Found Exception`

Explanation Class cannot be located.

Sample Code **public static final int EM_NOTFOUND=12;**

Recommended Action Verify that the CLASSPATH is set up and all jar files exist where specified in the CLASSPATH. The CLASSPATH is set up in /opt/BTScis/bin/cis3 and /opt/ems/bin/Runtime.sh on the EMS.

Error Message `Illegal Access Exception`

Explanation Class or initializer is not accessible.

Sample Code **public static final int EM_ACCESS=13;**

Error Message `Instantiation Exception`

Explanation Thrown if this class represents an abstract class, an interface, an array class, a primitive type, or void; or if the instantiation fails for some other reason.

Sample Code **public static final int EM_INSTANCE=14;**

Recommended Action Verify that the CLASSPATH is set up and all jar files exist where specified in the CLASSPATH. The CLASSPATH is set up in /opt/BTScis/bin/cis3 and /opt/ems/bin/Runtime.sh on the EMS.

Error Message `Security Exception`

Explanation There is no permission to create a new instance.

Sample Code **public static final int EM_SECURITY=15;**

Recommended Action Verify that the CORBA adapter is running as root. Verify by performing the following command on the EMS: **`ps -ef | grep cis3 and not the user`**.

Error Message `Invalid Request Exception`

Explanation Request is not valid or cannot be initialized.

Sample Code **public static final int EM_REQUEST=16;**

Recommended Action Verify that the XML request is syntactically correct according to the Cisco BTS 10200 Softswitch standard.

Error Message `Invalid Noun Exception`

Explanation Command noun is not found or is invalid.

Sample Code **public static final int EM_NOUN=17;**

Recommended Action Verify that the Noun (id) in the request is valid and correct for the current software release of the Cisco BTS 10200 Softswitch. Cross-check by executing the same command using CLI on the EMS. Refer to the *Cisco BTS 10200 Softswitch Command Line Interface Reference Guide* for syntax information.

Error Message `Invalid Verb Exception`

Explanation Command verb is not found or is invalid.

Sample Code **public static final int EM_VERB=18;**

Recommended Action Verify that the Verb in the request is valid and correct for the current software release of the Cisco BTS 10200 Softswitch. Cross-check by executing the same command using the CLI on the EMS. Refer to the *Cisco BTS 10200 Softswitch Command Line Interface Reference Guide* for syntax information.

Error Message `SQL Exception`

Explanation Thrown if the database cannot be accessed, if the constraints are violated, if there is another table conflict, if there is a resource issue, or any other Oracle-related cause.

Sample Code **public static final int EM_DATABASE=19;**

Recommended Action Verify that the relational databases are currently running on the EMSs. Verify by using the following commands:

1) **`ps -ef | grep mysqld.`** Verify there's a process running.
2) as root, perform a **`nodestat`** and verify that the Oracle database is running.
If either are not running, refer to the *Cisco BTS 10200 Operations and Maintenance Manual* to restart these processes.

Error Message `Invalid Value Exception`

Explanation A parameter value exceeds the range or some other restrictions like text length, pick-list, and so forth.

Sample Code **public static final int EM_VALUE=20;**

Recommended Action Verify that all the values within the XML request conforms to the restrictions for those values. Cross-check by executing the same command with the same values using the CLI.

Error Message `Invalid Key Exception`

Explanation An invalid key or token was used to describe some data value.

Sample Code **public static final int EM_KEY=21;**

Recommended Action Verify that all keys and values are valid within the XML request. Cross-check by executing the same command with the same values using the CLI.

Error Message `Missing Parameter Exception`

Explanation One or more required parameters were not included in the command parameter data.

Sample Code **public static final int EM_PARAM=22;**

Recommended Action Verify that all mandatory keys and values are valid. Cross-check by executing the same command with the same values using the CLI.

Modify the CORBA Network Configuration

In previous releases, the operator was asked for either an IP address or hostname. In this 4.5.1 release, the CORBA Installation automatically selects VIP (Virtual IP) if the VIP is configured. Otherwise, the first Management IP address is selected. This requires adding the hostname on the client machine. The following procedure can also be used to modify the name usage in the Cisco BTS 10200 Softswitch. In most cases, it is better to change to an IP address to prevent name resolution issues in the client-side network.

Note The hostname/IP address is used as part of the interoperable object reference (IOR) that is sent to the client side. That is why the name/IP address must resolve and route to the client.

Determine the hostname or IP address that works best for the client side of the network and perform the following steps to modify the CORBA network configuration:

Step 1 Edit the /etc/inittab file and modify the names on the last two entries to match the desired configuration. For example:

```
ns:3:respawn:/sbin/ins3 <VIP or EMS Management IP or hostname>
cs:3:respawn:/sbin/cis3 <VIP or EMS Management IP or hostname> <EMS Management IP #1>
< EMS Management IP #2>
```

Step 2 In the directory /opt/BTSoorb/config/. (a period (.) fully qualifies the domain directory), execute the setConfig.sh script. The XML configuration file is a bundled resource file in the OpenORB architecture. This command updates the JAR files with the modified OpenORB.xml file.

Step 3 Kill and restart the NameService (ins3) and CORBA Servant programs in the listed order.

```
pkill ins3
pkill cis3
```

The NameService (ins3) and CORBA Servant (cis3) programs restart automatically.

Step 4 Kill the Java children using the **ptree** and **kill** commands. Orphaned Java programs can cause unpredictable behavior.

CORBA Cannot Connect to the Cisco BTS 10200 Softswitch

If CORBA cannot connect to the Cisco BTS 10200 Softswitch, client-side hostname resolution may be required to match what was configured in the Element Management System (EMS).

Perform the following steps to troubleshoot:

Step 1 Determine the name used in the /etc/inittab file when invoking the entries:

```
ns:3:respawn:/sbin/ins3 <VIP or EMS Management IP or hostname>
cs:3:respawn:/sbin/cis3 <VIP or EMS Management IP or hostname> <EMS Management IP #1>
< EMS Management IP #2>
```

Step 2 Add the following arguments to the turn on debug-trace command to get the debug information on the client-side. These are the Java arguments:

```
-Dopenorb.debug.trace=DEBUG -Dopenorb.debug.level=HIGH
```

Step 3 If the hostname of the EMS server does not resolve, then an exception is thrown. If a default name was used such as “priems01” or “priems_nms1”, these names do not resolve in the client network. Uninstall and then reinstall the CORBA package to modify the EMS to accept IP addresses.

CORBA and EPOM Troubleshooting Steps

This section describes various procedures for troubleshooting CORBA with the Extensible Provisioning and Operations Manager (EPOM). This section requires a knowledge of Unix and Java commands.

Caution EPOM cannot be co-resident with the EMS.

Perform the following steps when a problem is encountered:

Step 1 Verify that the CORBA application is running.

```
ps -ef | grep cis
```

Step 2 Check the cis.log for errors or clues.

```
more /opt/ems/log/CIS.log
```

Step 3 Check the EPOM logs for errors or clues.

```
more var/opt/CSCOepom/logs
```

If the problem is with the way the Cisco BTS 10200 Softswitch CORBA services are configured, perform the following steps:

Step 1 Ensure that the Cisco BTS 10200 Softswitch hostname is provided in the CORBA configuration.

a. If the hostname is provided, change it to the IP address of the Cisco BTS 10200 Softswitch and restart the CORBA services on the Cisco BTS 10200 Softswitch.

b. If the hostname is not provided, stop and reinstall the CORBA application before doing anything else.

Step 2 Check if ports 683 and 14001 are listening on the Cisco BTS 10200 Softswitch server by using the following commands:

```
netstat -a |grep 683
netstat -a |grep 14001.
```

Step 3 Do a tail -f on the CIS.log on the Cisco BTS 10200 Softswitch server.

```
tail -f /opt/ems/log/CIS.log
```

Note EPOM log files can be found at: /var/opt/CSCOepom/logs. Check for trace.log and localhost files on the EPOM server.

Step 4 After clicking on the **Config** button on the EPOM GUI, (you may need to click more than once):

```
netstat -a |grep 14001 and netstat -a |grep 683
```

Step 5 Make sure that the username and password are the same as given when adding the Cisco BTS 10200 Softswitch inventory to EPOM. The username and password are optiuser/optiuser by default. Verify this by logging into the system using the CLI.

CORBA/EPOM Special CharacterTroubleshooting: Subscriber Commands

Table 5-1 shows the CORBA and EPOM responses to special characters in Subscriber commands.

Table 5-1 CORBA and EPOM Responses to Special Characters

Character	Usage Example	CORBA/EPOM Response
‘ (single quote)	`btsadmin>change subscriber id=x1-6-00-00-ca-ac-ef-98_02;name=Joe'`	**Error Message** `BtsException: IDL:oam.sswitch.com/cad/CadExceptions:1.0, Invalid parameter value. name=Joe'; contains one of the following invalid characters: ("')` **Explanation** The single quote is a reserved character for delimiting strings. It is used this way in caller-ID messages to the MTA. A single quote cannot be used in a name because it causes an error in the MTA that is parsing the caller-ID message.

Table 5-1 CORBA and EPOM Responses to Special Characters (continued)

Character	Usage Example	CORBA/EPOM Response
“ (double quote)	**btsadmin>change subscriber id=x1-6-00-00-ca-ac-ef-98_02;name=Joe"**	**Error Message** `org.omg.CORBA.UNKNOWN: Server Exception: Unregistered vendor exception #0 vmcid: 0x0 minor code: 0 completed` **Explanation** Invalid parameter value. name=Joe"; contains one of the following invalid characters: (") The double quote is a reserved character for delimiting strings. It is used this way in caller-ID messages to the MTA. A single quote cannot be used in a name because it causes an error in the MTA that is parsing the caller-ID message.

Table 5-1 CORBA and EPOM Responses to Special Characters (continued)

Character	Usage Example	CORBA/EPOM Response

; (semi-colon)	Example 1:
	btsadmin>change subscriber id=x1-6-00-00-ca-ac-ef-98_02;name=Joe;;
	
	Reply : Success: Transaction 914661109242987301 was processed.
	
	btsadmin>show subscriber id=x1-6-00-00-ca-ac-ef-98_02
	ID=x1-6-00-00-ca-ac-ef-98_02
	CATEGORY=INDIVIDUAL
	NAME=Joe
	STATUS=ACTIVE
	COUNTRY=USA
	PRIVACY=NONE
	RING_TYPE_DN1=1
	TERM_ID=aaln/S1/1
	MGW_ID=iad-11
	PIC1=NONE
	PIC2=NONE
	PIC3=NONE
	GRP=N
	USAGE_SENS=Y
	SUB_PROFILE_ID=subpf1
	TERM_TYPE=TERM
	IMMEDIATE_RELEASE=N
	TERMINATING_IMMEDIATE_REL=N
	SEND_BILLING_DN=N
	SEND_BDN_AS_CPN=N
	SEND_BDN_FOR_EMG=N
	
	Example 2:
	btsadmin>change subscriber id=x1-6-00-00-ca-ac-ef-98_02;name=Joe;
	Reply : Success: Transaction 914661178134430501 was processed.
	btsadmin>show subscriber id=x1-6-00-00-ca-ac-ef-98_02
	
	ID=x1-6-00-00-ca-ac-ef-98_02
	CATEGORY=INDIVIDUAL
	NAME=Joe
	STATUS=ACTIVE
	COUNTRY=USA
	PRIVACY=NONE
	RING_TYPE_DN1=1
	TERM_ID=aaln/S1/1
	MGW_ID=iad-11
	PIC1=NONE
	PIC2=NONE
	PIC3=NONE
	GRP=N
	USAGE_SENS=Y
	SUB_PROFILE_ID=subpf1
	TERM_TYPE=TERM
	IMMEDIATE_RELEASE=N
	TERMINATING_IMMEDIATE_REL=N
	SEND_BILLING_DN=N
	SEND_BDN_AS_CPN=N
	SEND_BDN_FOR_EMG=N
	Reply : Success: Entry 1 of 1 returned.
		Error Message `ID=x1-6-00-00-ca-ac-ef-98_02`
		`CATEGORY=INDIVIDUAL`
		`NAME=Joe;`
		`STATUS=ACTIVE`
		`COUNTRY=USA`
		`PRIVACY=NONE`
		`RING_TYPE_DN1=1`
		`TERM_ID=aaln/S1/1`
		`MGW_ID=iad-11`
		`PIC1=NONE`
		`PIC2=NONE`
		`PIC3=NONE`
		`GRP=N`
		`USAGE_SENS=Y`
		`SUB_PROFILE_ID=subpf1`
		`TERM_TYPE=TERM`
		`IMMEDIATE_RELEASE=N`
		`TERMINATING_IMMEDIATE_REL=N`
		`SEND_BILLING_DN=N`
		`SEND_BDN_AS_CPN=N`
		`SEND_BDN_FOR_EMG=N`
		
		`Reply : Success: Entry 1 of 1 returned.`
		
		Explanation In the first example, the two semicolons delimit only two null parameters. In the second example, the last parameter is a null parameter terminated by the <enter> key. The <enter> key always terminates the last parameter in the CLI.
		
		The fact that the semicolons terminate the command, without parameters between them, makes this possible. Placing the semicolons anywhere else causes the queue to hang.

Table 5-1 CORBA and EPOM Responses to Special Characters (continued)

Character	Usage Example	CORBA/EPOM Response
% (percent sign)	**btsadmin>change subscriber id=x1-6-00-00-ca-ac-ef-98_02;name=Joe%**	**Error Message** `BtsException: IDL:oam.sswitch.com/cad/CadExceptions:1.0, Invalid parameter value. name=Joe%; Enter at least 1 character, but not more than 32 characters.` **Explanation** The percent sign is a wildcard: for example, show subscriber id=x1-6-00% shows all subscribers whose id begins with x1-6-00. Thus the percent sign is a valid character for the subscriber noun, but is not valid when used with the add or change verbs.
- (hyphen)	**btsadmin>change subscriber id=x1-6-00-00-ca-ac-ef-98_02;name=Joe-**	**Error Message** `Success: Transaction 914661231396356901 was processed.` **Explanation** This is a valid character but impacts the way the caller id is displayed. Using this character is not recommended.
_ (under-score)	**btsadmin>change subscriber id=x1-6-00-00-ca-ac-ef-98_02;name=Joe_**	**Error Message** `Failure: NAME cannot contain an _ character` **Explanation** This is a valid character but impacts the way the caller id is displayed. Using this character is not recommended.
&	**btsadmin>change subscriber id=x1-6-00-00-ca-ac-ef-98_02;name=Joe&** `Reply : Success: Transaction 914661254499435301 was processed.`	**Error Message** `org.omg.CORBA.UNKNOWN: Server Exception: Unregistered vendor exception #0 vmcid: 0x0 minor code: 0 completed` **Explanation** The ampersand is a reserved character in XML that causes this transaction to fail when sent using XML over a CORBA interface.
@#$^&*()}{|\/<>,.:[]~! (With ampersand)	**btsadmin>change subscriber id=x1-6-00-00-ca-ac-ef-98_02;name=Joe!@#$^&*()}{|\/<>,.:[]~** `Reply : Success: Transaction 914661290390225701 was processed.`	**Error Message** `org.omg.CORBA.UNKNOWN: Server Exception: Unregistered vendor exception #0 vmcid: 0x0 minor code: 0 completed.` **Explanation** The ampersand is a reserved character in XML that causes this transaction to fail when sent using the XML over a CORBA interface.
@#$^*()}{|\/>,.:[]~! (No ampersand)	**btsadmin>change subscriber id=x1-6-00-00-ca-ac-ef-98_02;name=Joe!@#$^*()}{|\/<>,.:[]~**	**Error Message** `Successful` **Explanation** These characters are valid.

APPENDIX A

XML Description Documents

This appendix provides examples of the XML description documents.

Subscriber Noun and Add Verb

The following sample XML description document provides an example of the subscriber noun and add verb.

```
================================================
<ManagedObject Verb="add" id="subscriber"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ManagedObject.xsd">
  <MOAttribute id="dn1">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>14</Width>
    <HelpText>Enter at least 1, but not more than 14 characters from the following set:
{0123456789-}.</HelpText>
    <Label>Dn1</Label>
    <Parser id="GenericDNParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="tg">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1 character, but not more than 32 characters.</HelpText>
    <Label>Tg</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="policy_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>POLICY_ID</Label>
    <Parser id="TextParser">
```

```
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="category">
    <Required>false</Required>
    <Type>single</Type>
    <Default>[INDIVIDUAL]</Default>
    <Width>15</Width>
    <HelpText>Enter one of the following values: [INDIVIDUAL, MLHG, MLHG_INDIVIDUAL,
MLHG_PREF_INDIV, CTXG, CTXG_INDIVIDUAL, PBX, CTXG_TG, CTXG_MLHG, RACF, IVR]</HelpText>
    <Label>Category</Label>
   <Permitted>[INDIVIDUAL, MLHG, MLHG_INDIVIDUAL, MLHG_PREF_INDIV, CTXG, CTXG_INDIVIDUAL,
PBX, CTXG_TG, CTXG_MLHG, RACF, IVR]</Permitted>
  </MOAttribute>
  <MOAttribute id="ss_number">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>11</Width>
   <HelpText>Enter a Social Security Number in the form ###-##-#### where # is digit from
0-9.</HelpText>
    <Label>Ss Number</Label>
    <Parser id="SocSecParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="ctxg_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>Ctxg Id</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="name">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1 character, but not more than 32 characters.</HelpText>
    <Label>Name</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="mlhg_pref_list_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>Mlhg Pref List Id</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
```

```
  <MOAttribute id="address2">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1 character, but not more than 32 characters.</HelpText>
    <Label>Address2</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="address1">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1 character, but not more than 32 characters.</HelpText>
    <Label>Address1</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="city">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>City</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="terminating_immediate_rel">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[N]</Default>
    <Width>3</Width>
    <HelpText>Enter a boolean value of Y for yes or N for no.</HelpText>
    <Label>Terminating Immediate Release</Label>
    <Parser id="BooleanParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="billing_dn">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1, but not more than 32 characters from the following set:
{0123456789-}.</HelpText>
    <Label>Billing Dn</Label>
    <Parser id="GenericDNParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="language">
    <Required>false</Required>
    <Type>text</Type>
```

```
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>Language</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="email">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>64</Width>
    <HelpText>Enter an email address in the form text@text where text is a set of
characters with no spaces.</HelpText>
    <Label>Email</Label>
    <Parser id="EmailParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="mlhg_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>MLHG ID</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="tgn_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>8</Width>
    <HelpText>Enter a number from 0 to 99999999.</HelpText>
    <Label>Trunk Group Number ID</Label>
    <Parser id="DecimalParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="mgw_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 0 characters, but not more than 32 characters.</HelpText>
    <Label>Media Gateway ID</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="status">
    <Required>false</Required>
    <Type>single</Type>
    <Default>[ACTIVE]</Default>
    <Width>17</Width>
```

```
    <HelpText>Enter one of the following values: [ACTIVE, TEMP_OOS, TEMP_DISCONNECTED,
TEMP_UNAVAILABLE]</HelpText>
    <Label>Status</Label>
    <Permitted>[ACTIVE, TEMP_OOS, TEMP_DISCONNECTED, TEMP_UNAVAILABLE]</Permitted>
  </MOAttribute>
  <MOAttribute id="term_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1 character, but not more than 32 characters.</HelpText>
    <Label>Termination ID</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="usage_sens">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[Y]</Default>
    <Width>3</Width>
    <HelpText>Enter a boolean value of Y for yes or N for no.</HelpText>
    <Label>Usage Sens</Label>
    <Parser id="BooleanParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="id">
    <Required>true</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>30</Width>
    <HelpText>Enter at least 1 character, but not more than 30 characters.</HelpText>
    <Label>ID</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="grp">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[N]</Default>
    <Width>3</Width>
    <HelpText>Enter a boolean value of Y for yes or N for no.</HelpText>
    <Label>Grp</Label>
    <Parser id="BooleanParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="sub_profile_id">
    <Required>true</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>Sub Profile Id</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
```

```
</MOAttribute>
<MOAttribute id="country">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[USA]</Default>
  <Width>16</Width>
  <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
  <Label>Country</Label>
  <Parser id="TextParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="cos_restrict_id">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>16</Width>
  <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
  <Label>COS Restrict ID</Label>
  <Parser id="TextParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="qos_id">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>16</Width>
  <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
  <Label>QOS ID</Label>
  <Parser id="TextParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="term_type">
  <Required>false</Required>
  <Type>single</Type>
  <Default>[TERM]</Default>
  <Width>5</Width>
  <HelpText>Enter one of the following values: [TERM, TG, ROUTE, RG]</HelpText>
  <Label>TERM TYPE</Label>
  <Permitted>[TERM, TG, ROUTE, RG]</Permitted>
</MOAttribute>
<MOAttribute id="ring_type_dn1">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[1]</Default>
  <Width>1</Width>
  <HelpText>Enter a number from 1 to 3.</HelpText>
  <Label>Ring Type Dn1</Label>
  <Parser id="DecimalParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="immediate_release">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[N]</Default>
  <Width>3</Width>
  <HelpText>Enter a boolean value of Y for yes or N for no.</HelpText>
```

```
  <Label>Immediate Release</Label>
  <Parser id="BooleanParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="sip_url">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>32</Width>
  <HelpText>Enter at least 1 character, but not more than 32 characters.</HelpText>
  <Label>Sip Url</Label>
  <Parser id="TextParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="zipcode">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>10</Width>
  <HelpText>Enter at least 1 character, but not more than 10 characters.</HelpText>
  <Label>Zipcode</Label>
  <Parser id="TextParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="pic3">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>4</Width>
  <HelpText>Enter a PIC value as four numeric characters, NPIC, or NONE.</HelpText>
  <Label>Pic3</Label>
  <Parser id="PicParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="pic2">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>4</Width>
  <HelpText>Enter a PIC value as four numeric characters, NPIC, or NONE.</HelpText>
  <Label>Pic2</Label>
  <Parser id="PicParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="privacy">
  <Required>false</Required>
  <Type>single</Type>
  <Default>[NONE]</Default>
  <Width>4</Width>
  <HelpText>Enter one of the following values: [FULL, NAME, NONE]</HelpText>
  <Label>Privacy</Label>
  <Permitted>[FULL, NAME, NONE]</Permitted>
</MOAttribute>
<MOAttribute id="pic1">
```

```
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>4</Width>
    <HelpText>Enter a PIC value as four numeric characters, NPIC, or NONE.</HelpText>
    <Label>Pic1</Label>
    <Parser id="PicParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="state">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>State</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
</ManagedObject>
```

Foreign Key Relationships

The following sample XML extended format description document contains the foreign key relationships for a given command.

```
================================================
<ManagedObject Verb="add" id="subscriber"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ManagedObject.xsd">
  <MOAttribute id="dn1">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>14</Width>
    <HelpText>Enter at least 1, but not more than 14 characters from the following set:
{0123456789-}.</HelpText>
    <Label>Dn1</Label>
    <Parser id="GenericDNParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="tg">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1 character, but not more than 32 characters.</HelpText>
    <Label>Tg</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="policy_id">
    <Required>false</Required>
```

```
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>POLICY_ID</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="category">
    <Required>false</Required>
    <Type>single</Type>
    <Default>[INDIVIDUAL]</Default>
    <Width>15</Width>
    <HelpText>Enter one of the following values: [INDIVIDUAL, MLHG, MLHG_INDIVIDUAL,
MLHG_PREF_INDIV, CTXG, CTXG_INDIVIDUAL, PBX, CTXG_TG, CTXG_MLHG, RACF, IVR]</HelpText>
    <Label>Category</Label>
    <Permitted>[INDIVIDUAL, MLHG, MLHG_INDIVIDUAL, MLHG_PREF_INDIV, CTXG, CTXG_INDIVIDUAL,
PBX, CTXG_TG, CTXG_MLHG, RACF, IVR]</Permitted>
  </MOAttribute>
  <MOAttribute id="ss_number">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>11</Width>
    <HelpText>Enter a Social Security Number in the form ###-##-#### where # is digit from
0-9.</HelpText>
    <Label>Ss Number</Label>
    <Parser id="SocSecParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="ctxg_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>Ctxg Id</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
    <Fk id="CENTREX_GRP_PK">
      <Noun>centrex_grp</Noun>
      <Param>id</Param>
    </Fk>
  </MOAttribute>
  <MOAttribute id="name">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1 character, but not more than 32 characters.</HelpText>
    <Label>Name</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="mlhg_pref_list_id">
    <Required>false</Required>
```

```
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>Mlhg Pref List Id</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
    <Fk id="MLH_PREF_LIST_PK">
      <Noun>mlhg_pref_list</Noun>
      <Param>id</Param>
    </Fk>
  </MOAttribute>
  <MOAttribute id="address2">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1 character, but not more than 32 characters.</HelpText>
    <Label>Address2</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="address1">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1 character, but not more than 32 characters.</HelpText>
    <Label>Address1</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="city">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>City</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="terminating_immediate_rel">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[N]</Default>
    <Width>3</Width>
    <HelpText>Enter a boolean value of Y for yes or N for no.</HelpText>
    <Label>Terminating Immediate Release</Label>
    <Parser id="BooleanParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="billing_dn">
    <Required>false</Required>
```

```
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1, but not more than 32 characters from the following set:
{0123456789-}.</HelpText>
    <Label>Billing Dn</Label>
    <Parser id="GenericDNParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="language">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>Language</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="email">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>64</Width>
    <HelpText>Enter an email address in the form text@text where text is a set of
characters with no spaces.</HelpText>
    <Label>Email</Label>
    <Parser id="EmailParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="mlhg_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>MLHG ID</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
    <Fk id="MLH_PREF_LIST_PK">
      <Noun>mlhg_pref_list</Noun>
      <Param>mlhg_id</Param>
      <Fk id="MLHG_PK">
        <Noun>mlhg</Noun>
        <Param>id</Param>
      </Fk>
    </Fk>
  </MOAttribute>
  <MOAttribute id="tgn_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>8</Width>
    <HelpText>Enter a number from 0 to 99999999.</HelpText>
    <Label>Trunk Group Number ID</Label>
    <Parser id="DecimalParser">
```

```
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
    <Fk id="TRUNK_GRP_PK">
      <Noun>trunk_grp</Noun>
      <Param>id</Param>
    </Fk>
  </MOAttribute>
  <MOAttribute id="mgw_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 0 characters, but not more than 32 characters.</HelpText>
    <Label>Media Gateway ID</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
    <Fk id="TERMINATION_PK">
      <Noun>termination</Noun>
      <Param>mgw_id</Param>
      <Fk id="MGW_PK">
        <Noun>mgw</Noun>
        <Param>id</Param>
      </Fk>
    </Fk>
  </MOAttribute>
  <MOAttribute id="status">
    <Required>false</Required>
    <Type>single</Type>
    <Default>[ACTIVE]</Default>
    <Width>17</Width>
    <HelpText>Enter one of the following values: [ACTIVE, TEMP_OOS, TEMP_DISCONNECTED,
TEMP_UNAVAILABLE]</HelpText>
    <Label>Status</Label>
    <Permitted>[ACTIVE, TEMP_OOS, TEMP_DISCONNECTED, TEMP_UNAVAILABLE]</Permitted>
  </MOAttribute>
  <MOAttribute id="term_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1 character, but not more than 32 characters.</HelpText>
    <Label>Termination ID</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
    <Fk id="TERMINATION_PK">
      <Noun>termination</Noun>
      <Param>id</Param>
    </Fk>
  </MOAttribute>
  <MOAttribute id="usage_sens">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[Y]</Default>
    <Width>3</Width>
    <HelpText>Enter a boolean value of Y for yes or N for no.</HelpText>
    <Label>Usage Sens</Label>
    <Parser id="BooleanParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
```

```
  </Parser>
</MOAttribute>
<MOAttribute id="id">
  <Required>true</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>30</Width>
  <HelpText>Enter at least 1 character, but not more than 30 characters.</HelpText>
  <Label>ID</Label>
  <Parser id="TextParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="grp">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[N]</Default>
  <Width>3</Width>
  <HelpText>Enter a boolean value of Y for yes or N for no.</HelpText>
  <Label>Grp</Label>
  <Parser id="BooleanParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="sub_profile_id">
  <Required>true</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>16</Width>
  <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
  <Label>Sub Profile Id</Label>
  <Parser id="TextParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
  <Fk id="SUBSCRIBER_PROFILE_PK">
    <Noun>subscriber_profile</Noun>
    <Param>id</Param>
  </Fk>
</MOAttribute>
<MOAttribute id="country">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[USA]</Default>
  <Width>16</Width>
  <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
  <Label>Country</Label>
  <Parser id="TextParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="cos_restrict_id">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>16</Width>
  <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
  <Label>COS Restrict ID</Label>
  <Parser id="TextParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
```

```
    </Parser>
    <Fk id="COST_RESTRICT_PK">
      <Noun>cos_restrict</Noun>
      <Param>id</Param>
    </Fk>
  </MOAttribute>
  <MOAttribute id="qos_id">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>16</Width>
    <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
    <Label>QOS ID</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
    <Fk id="QOS_PK">
      <Noun>qos</Noun>
      <Param>id</Param>
    </Fk>
  </MOAttribute>
  <MOAttribute id="term_type">
    <Required>false</Required>
    <Type>single</Type>
    <Default>[TERM]</Default>
    <Width>5</Width>
    <HelpText>Enter one of the following values: [TERM, TG, ROUTE, RG]</HelpText>
    <Label>TERM TYPE</Label>
    <Permitted>[TERM, TG, ROUTE, RG]</Permitted>
  </MOAttribute>
  <MOAttribute id="ring_type_dn1">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[1]</Default>
    <Width>1</Width>
    <HelpText>Enter a number from 1 to 3.</HelpText>
    <Label>Ring Type Dn1</Label>
    <Parser id="DecimalParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="immediate_release">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[N]</Default>
    <Width>3</Width>
    <HelpText>Enter a boolean value of Y for yes or N for no.</HelpText>
    <Label>Immediate Release</Label>
    <Parser id="BooleanParser">
      <JavaScript>TBD</JavaScript>
      <RegExp>TBD</RegExp>
    </Parser>
  </MOAttribute>
  <MOAttribute id="sip_url">
    <Required>false</Required>
    <Type>text</Type>
    <Default>[null]</Default>
    <Width>32</Width>
    <HelpText>Enter at least 1 character, but not more than 32 characters.</HelpText>
    <Label>Sip Url</Label>
    <Parser id="TextParser">
      <JavaScript>TBD</JavaScript>
```

```
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="zipcode">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>10</Width>
  <HelpText>Enter at least 1 character, but not more than 10 characters.</HelpText>
  <Label>Zipcode</Label>
  <Parser id="TextParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="pic3">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>4</Width>
  <HelpText>Enter a PIC value as four numeric characters, NPIC, or NONE.</HelpText>
  <Label>Pic3</Label>
  <Parser id="PicParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="pic2">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>4</Width>
  <HelpText>Enter a PIC value as four numeric characters, NPIC, or NONE.</HelpText>
  <Label>Pic2</Label>
  <Parser id="PicParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="privacy">
  <Required>false</Required>
  <Type>single</Type>
  <Default>[NONE]</Default>
  <Width>4</Width>
  <HelpText>Enter one of the following values: [FULL, NAME, NONE]</HelpText>
  <Label>Privacy</Label>
  <Permitted>[FULL, NAME, NONE]</Permitted>
</MOAttribute>
<MOAttribute id="pic1">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[null]</Default>
  <Width>4</Width>
  <HelpText>Enter a PIC value as four numeric characters, NPIC, or NONE.</HelpText>
  <Label>Pic1</Label>
  <Parser id="PicParser">
    <JavaScript>TBD</JavaScript>
    <RegExp>TBD</RegExp>
  </Parser>
</MOAttribute>
<MOAttribute id="state">
  <Required>false</Required>
  <Type>text</Type>
  <Default>[null]</Default>
```

```
      <Width>16</Width>
      <HelpText>Enter at least 1 character, but not more than 16 characters.</HelpText>
      <Label>State</Label>
      <Parser id="TextParser">
        <JavaScript>TBD</JavaScript>
        <RegExp>TBD</RegExp>
      </Parser>
    </MOAttribute>
  </ManagedObject>
```

APPENDIX B

XML Test Drivers

This appendix describes the XML test drivers.

XML Request Batch File

The following example XML/CORBA interface test driver executes an XML request batch file built as a CLI script. This allows compatibility with older CLI scripts used to provision the system.

```
package com.sswitch.oam.drv;

import java.lang.*;
import java.io.*;
import java.util.*;
import java.text.*;
// XML Stuff
import org.apache.ecs.xml.*;
import org.apache.ecs.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import org.apache.xml.serialize.*;
// BTS Interface Code objects...
import com.sswitch.oam.cad.*;
import com.sswitch.oam.xml.*;
import com.sswitch.oam.util.*;
import com.sswitch.oam.ccc.*;

/**
 * XmlBatch.java
 * Copyright (c) 2002, 2003 by Cisco Systems, Inc.
 * --Test driver for the XML/CORBA interface...
 *    This test driver executes a batch file built as a CLI script as XML
 *    requests. This allows compatibility with the older CLI scripts used to
 *    provision the system. Note that this example can be built with the
 *    provided tool "oo-cc" this simple script creates the correct CLASSPATH
 *    and invokes the compiler with the correct options. Also, the "oo-idl"
 *    tool can be used to generate the correct IDL output.
 *
 *   @author A. J. Blanchard
 *   @version  3.0
*/

public class XmlBatch {

    /*
     * Class private data
```

```
     */
    private String []                                   objArgs;
    private CorbaXmlIntf                                objBts;

    private File                                        objFile;
    private RandomAccessFile                            objFileHandle;

    /**
     * Generic Constructor for the test driver.
     */
    protected        XmlBatch(String[] args)
    {
      // Initialize the ORB.
      objArgs = args;
      objBts = new CorbaXmlIntf(args);
      return;
    }


    /**
     * This is the main method for the application.
     */
    public static void main(String[] args)
    {

      //
      // Verify that the argument match what is expected...
      //  oo-run XmlRequest <CLI Request File> \
      //       -ORBopenorb.config="./OpenORB.xml"
      //
      if(args.length < 2)
        {
        System.out.println("\nStart program as: oo-run XmlRequest <XML Request File>
-ORBopenorb.config=\"./OpenORB.xml\"\n");
        System.exit(0);
        }
      XmlBatch me = new XmlBatch(args);
      me.go();
      return;
    }



    /**
     * This is the primary execution method for the object. It performs the
     * actual request and calls for the print of the reply.
     */
    protected void       go()
    {
      //
      // Log into the target machine with generic optiuser
      //
      try {
        objBts.connect();
        System.out.println("BTS10200 Login successful...");
        }
      catch (Exception e) {
        System.out.println("Exception in login = " + e);
        System.exit(1);
        }
      //
      // Read in the file and send request...
      //
      try {
```

```
    String reply = "";

    openCLI();

    while(true)
      {
      CommandParser parser = new CommandParser();
      String cmd = readCLI(); // Fetch the request file....
      if(cmd==null)
        break;

      if(cmd.startsWith("#") || cmd.length()==0)
         continue;

      else
        {
        // Issue request to BTS 10200
        reply = objBts.request(parser.toXML(cmd));
        System.out.println("RETURN VALUE: ");
        parser.prettyPrint(reply);
        }
      } // end while(1)

    closeCLI();
    // Clean up and logout
    objBts.disconnect();
    }
  catch (CadExceptions ce) {
    System.out.println("CIS Command Exception: CODE="+ce.error_code +
                       "\n"+ce.error_string);
    ce.printStackTrace();
    }
  catch (Exception e) {
    System.out.println("Batch Command Exception = " + e);
    e.printStackTrace();
    }
  return;
} // end go()


/**
 * Open the input file for reading. This allows the read method to suck
 * a line at a time of the CLI style input.
 */
protected void        openCLI()
{
  try {
    objFile=new File(objArgs[1]);
    objFileHandle=new RandomAccessFile(objFile,"r");
    }
  catch(Exception e) {
    // In the event of an error. just bail out of the program
    System.out.println("Error in processing file:\n"+e.toString());
    System.exit(1);
    }
}

/**
 * This is the method that closes and clean up after a file has been
 * processed.
 */
protected void        closeCLI()  throws java.io.IOException
{
  objFileHandle.close();
```

```
    }

    /**
     * Read in the file provided as the request. Just exit on errors. Don't
     * worry about throwing an error exception.
     */
    protected String      readCLI()
    {
      String data=null;

      try {
        if((data = objFileHandle.readLine())!=null)
           data.trim();
        }
      catch (Exception e) {
        System.out.println("Unable to read "+objFile.toString()+
                           " with error:\n"+e.toString());
        System.exit(1);
        }
      // May return a valid string or a NULL object...
      return data;
    }


    //=========================================================================
    // Tools and utilities...
    //=========================================================================


} // end XmlRequest
```

CLI to CORBA XML Transaction

The following example test driver executes a normal CLI command but processes it as a CORBA XML transaction.

```
package com.sswitch.oam.drv;

import java.lang.*;
import java.io.*;
import java.util.*;
import java.text.*;
// XML Stuff
import org.apache.ecs.xml.*;
import org.apache.ecs.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import org.apache.xml.serialize.*;
// BTS Utility Code objects...
import com.sswitch.oam.cad.*;
import com.sswitch.oam.xml.*;
import com.sswitch.oam.util.*;
import com.sswitch.oam.ccc.*;

/**
 * XmlCli.java
 * Copyright (c) 2002, 2003 by Cisco Systems, Inc.
 * --Test driver for the XML/CORBA interface...
 *    This test driver executes a normal CLI command and processes the request
```

```
 *    as a CORBA XML transaction. The reply is then displayed. I am no as
 *    concerned with complex data show(s) as with the ability to issue
 *    provisioning commands. Note that this exmaple can be built with the
 *    provided tool "oo-cc" this simple script creates the correct CLASSPATH
 *    and invokes the compiler with the correct options. Also, the "oo-idl"
 *    tool can be used to generate the correct IDL output.
 *
 *   @author   A. J. Blanchard
 *   @version  3.0
 *
 */

public class XmlCli {

    /*
     * Class private data
     */
    private String []                                objArgs;
    private CorbaXmlIntf                             objBts;

    /**
     * Generic Constructor for the test driver.
     */
    protected        XmlCli(String[] args)
    {
      // Initialize the BTS ORB interface object.
      objArgs = args;
      objBts = new CorbaXmlIntf(args);
      return;
    }


    /**
     * This is the main method for the application.
     */
    public static void main(String[] args)
    {

      //
      // Verify that the argument match what is expected...
      //  oo-run XmlCli -ORBopenorb.config="./OpenORB.xml"
      //
      if(args.length < 1)
        {
        System.out.println("\nStart program as: oo-run XmlCli
-ORBopenorb.config=\"./OpenORB.xml\" \n");
        System.exit(0);
        }
      XmlCli me = new XmlCli(args);
      me.go();
      return;
    }



    /**
     * This is the primary execution method for the object. It performs the
     * actual request and calls for the print of the reply.
     */
    protected void       go()
    {
      //
      // Log into the target machine with generic optiuser
      //
```

```
  try {
    objBts.connect();
    System.out.println("BTS10200 Login successful...");
    }
  catch (Exception e) {
    System.out.println("Exception in login = " + e);
    System.exit(1);
    }
  //
  // Read in the file and send request...
  //
  try {

    while(true)
      {
      openCLI();                    // Put out the prompt...
      CommandParser parser = new CommandParser();
      String cmd = readCLI().trim(); // Fetch the request file...
      String reply = "";
      if((cmd.equals("")))
        continue;
      if(cmd.equals("exit"))
        break;
      else
        {
        // Issue request to BTS 10200
        reply = objBts.request(parser.toXML(cmd));
        System.out.println("RETURN VALUE: ");
        parser.prettyPrint(reply);
        }

      } // end while(1)

    closeCLI();
    // Clean up and logout
    objBts.disconnect();
    }
  catch (CadExceptions ce) {
    System.out.println("CIS Command Exception: CODE="+ce.error_code +
                       "\n"+ce.error_string);
    ce.printStackTrace();
    }
  catch (Exception e) {
    System.out.println("CIS Command Exception: \n"+e.toString());
    //e.printStackTrace();
    }
  return;
} // end go()



/**
 * Open the input file for reading. This allows the read method to suck
 * a line at a time of the CLI style input.
 */
protected void        openCLI()
{
  System.out.print("CORBA-CLI> ");
  return;
}

/**
 * This is the method that closes and clean up after a file has been
 * processed.
```

```
     */
    protected void        closeCLI()  throws java.io.IOException
    {
      System.out.println("\n Bye...");
      return;
    }

    /**
     * Read in the file provided as the request. Just exit on errors. Don't
     * worry about throwing an error exception.
     */
    protected String      readCLI()  throws java.io.IOException
    {
      int          temp=0;
      int          idx=0;
      byte []      buf= new byte[256];

      while(true)
           {
           temp = System.in.read();
           if(temp==10)           // <ENTER Key>
             break;
           buf[idx++]=(byte) temp;
           }
      return new String(buf);
    }

    //=========================================================================
    // Tools and utilities...
    //=========================================================================


} // end XmlCli
```

APPENDIX C

Sample CORBA Client Package (BTSxsdk) Implementation

This appendix details a sample implementation of a CORBA Client Package (BTSxsdk). BTSxsdk is a Software Development Kit (SDK) that implements a sample CORBA client for connecting the Cisco BTS 10200 Softswitch through the CORBA adapter. BTSxsdk package is automatically installed onto the Element Management System (EMS) when the CORBA adapter (BTScis package) installs. Use this SDK as a starting point to develop custom client programs. Successfully running this SDK also verifies the correct installation of the CORBA adapter on an EMS.

BTSxsdk Java Capabilities

This section describes some of capabilities of the BTSxsdk.

SDK

The Software Developers Kit (SDK) includes the following capabilities:

- Provide both secured and non-secured CORBA interface access using port 684 for the Cognitronics Privacy Manager application.
- The Cisco BTS 10200 Softswitch CORBA interface supports both nonsecured and secured access to the Cisco BTS 10200 Softswitch CORBA interfaces.
- A single Cisco BTS 10200 Softswitch CORBA interface to customer OSS applications can redirect requests from the OSS to 2 redundant Cisco BTS 10200 Softswitch systems.
- Provide CORBA access to the active EMS within a Cisco BTS 10200 Softswitch system. The Cisco BTS 10200 Softswitch CORBA interface exclusively binds to the active EMS.
- No predetermined NIC interface is required during Cisco BTS 10200 Softswitch installation. Previous Cisco BTS 10200 Softswitch installation required operators to specify a NIC address in the dual NIC interface configuration. The Cisco BTS 10200 Softswitch CORBA interface now dynamically binds to the VIP or a NIC address in a dual NIC configuration.
- Provide a single management interface to all Cisco BTS 10200 Softswitch systems.

Java

Java classes provide the following capabilities:

- Support access to multiple BTS sites using CLLI code
- Conceal redundant EMS nodes (for ACTIVE access only)
- Conceal redundant NIC(s) on each EMS node
- Track the state of the CIS application on each EMS
- Supply a single set of IOR references to the Cisco BTS 10200 Softswitch client applications

Most of this capability is supplied in a series of classes to abstract access to the Cisco BTS 10200 Softswitch. This simplifies the management of objects and connections to the Cisco BTS 10200 Softswitch.

CORBA Interface Servant

Extensions

The following extensions were added for the CORBA Interface Servant (CIS) application. These extensions are applied in running CIS application engine.

- Bind individual IP addresses to Name Service(s) on each EMS
- Unbind if EM01 is STANDBY, shutdowns or fails over
- Stop processing requests when not in ACTIVE EM01

Dual Mode Operation

Dual mode operations apply. Both non-secure and secure provisioning work only on the active EMS. A switchover from either a failure or a manual request causes the objects to unbind and current connections to break. As a result, queries to the standby side produce Java exceptions.

Prerequisites

The BTSxsdk package is automatically installed onto the EMS when the CORBA adapter installs. If the BTSxsdk is to be installed onto another machine, the following prerequisites apply when implementing the BTSxsdk package:

- BTSxsdk package must be installed on a UNIX machine that is in the same intranet as the EMS where the CORBA adapter is installed.
- Java Development Kit (JDK) 1.4.1_01 or later is installed. This document assumes that the JDK is installed under /usr/java.

Users must be familiar with the Cisco BTS 10200 Softswitch Command Line Interface (CLI) adapter commands.

OpenORB Settings

The OpenORB package requires a modification to the JDK. It updates properties in the JDK to enable it to point to the OpenORB implementation for the CORBA interface objects. There are two ways to modify JDK for OpenORB:

1. the Cisco BTS 10200 Softswitch package BTSoorb automatically updates the JDK during installation, or
2. type the following commands in the order given:
   - cd /opt/BTSxsdk/oorb/lib
   - java -jar openorb-1.3.1.jar

Build the BTSxsdk

The BTSxsdk package is pre-built in Java. A build is not necessary if sample files are not modified. If necessary, perform the following steps to build a sample client application.

Step 1 Go to the BTSxsdk directory.

```
cd /opt/BTSxsdk
```

Step 2 Perform export.

```
export PROJECTDIR=`pwd`
```

Step 3 Perform export.

```
export JAVA_HOME=/usr/java
```

Step 4 Perform make all.

```
./bin/make all
```

Run BTSxsdk

To run the sample CLI client application, perform the following steps:

Step 1 Go to the bin directory.

```
cd /opt/BTSxsdk/bin
```

Step 2 Issue the following command to run the BTSxdsk.

a. To connect to the non-secure CORBA server, enter the command:

```
./bts-cli <Active EMS IP> -n <username> -p <password>
```

The "Active EMS IP" can be one of the following:

- One of the active EMS management IP addresses
- VIP
- Active EMS DNS name

b. To connect to the secure CORBA server, enter the command:

```
./bts-cli-secure <Active EMS Mgmt IP> -n <username> -p <password>
```

The "active EMS Mgmt IP" must be one of the active EMS management IP addresses.

A CLI-like interface appears. Issue a test CLI command such as **show sub; limit=1**—the XML response displays on the screen.