



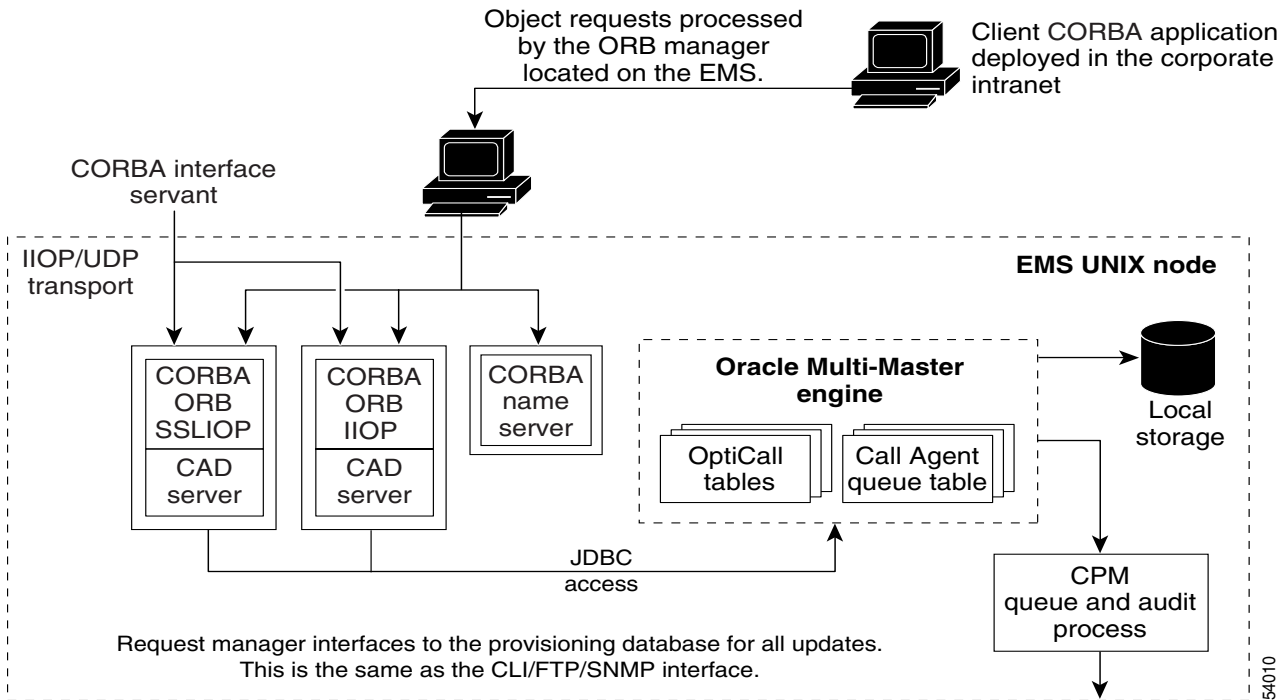
CORBA Architecture and Application Programming Interface

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

- [CORBA Adapter Architecture, page 1-1](#)
- [Cisco BTS 10200 Softswitch API, page 1-9](#)

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 process.

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

CAD utilizes the Java language implementation of the OpenORB 1.3.1 CORBA package. Likewise, the servant implementation leverages the Java POA interface to the ORB. The servant implementation supports a Java client-side implementation when using OpenORB. The interface is truly language neutral.

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"
```

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 using 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. 21.3.15) protocol.

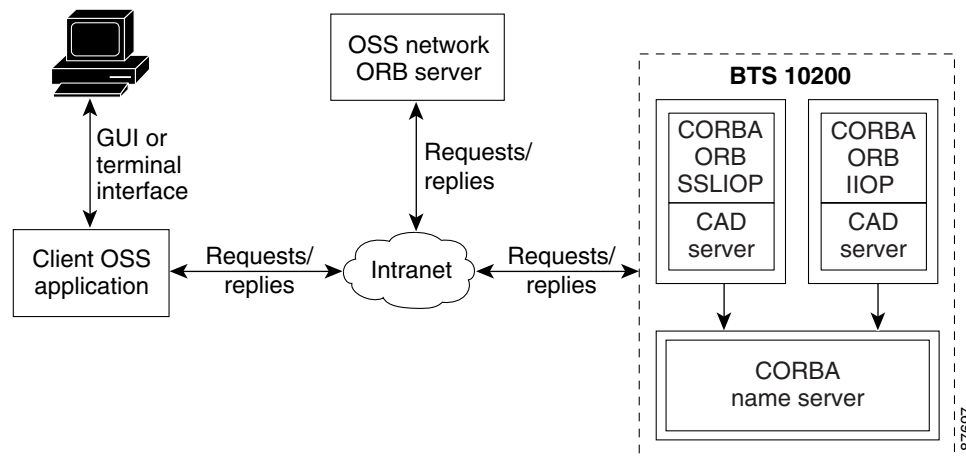
The following example details an object resolution using the NameService module. Note that at this time the basic POA is used as a root-level reference to the local Cisco BTS 10200 Softswitch.

```
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);
```

ORB Deployment—Client Build and Bind Semantics

Figure 1-2 shows the ORB deployment process.

Figure 1-2 ORB Deployment



All clients must bind to the POA name to access the objects under it. In addition, the objects themselves require a full package path specified for the implementation. 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.



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.

The code 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, 2003 by Cisco Systems, Inc.
#
# AUTHOR: A. J. Blanchard
#
# DESC: Invoke the IDL compiler for the OpenORB package.
#
#####
set -e
set -a
#set -x

if [ -z "$PROJECTDIR" ]; then
    PROJECTDIR=/opt/BTSxsdk
fi

#
# List required jar files
#
CLASSPATH=.:$HOME/mb/devel/em/lib:/opt/BTSorb/lib/logkit.jar:/opt/BTSorb/lib/openorb-1.3
.1.jar:/opt/BTSorb/lib/openorb_tools-1.3.1.jar:/opt/BTSorb/lib/xerces.jar:/opt/BTSorb/l
ib/avalon-framework.jar:/opt/BTSorb/lib/openorb_ots-1.3.1.jar:/opt/BTSorb/lib/openorb_ps
s-1.3.1.jar:/opt/BTSorb/lib/openorb_ins-1.3.1.jar:/opt/BTSorb/lib/openorb_tns-1.3.1.jar

export CLASSPATH

#java -classpath $CLASSPATH org.openorb.compiler.IdlCompiler $1 -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, 2003 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

if [ -z "$PROJECTDIR" ]; then
    PROJECTDIR=/opt/BTSxsdk
fi

CLASSPATH=.:$HOME/lib/xml-tool.jar:$HOME/mb/devel/em/lib/mysql_2_uncomp.jar:/opt/BTSorb/l
ib/logkit.jar:/opt/BTSorb/lib/openorb-1.3.1.jar:/opt/BTSorb/lib/openorb_tools-1.3.1.jar:
/opt/BTSorb/lib/xerces.jar:$HOME/mb/devel/em/lib/cad.jar:$HOME/mb/devel/em/lib/orajdbc12.
zip:$HOME/mb/devel/em/lib/ecs-1.4.1.jar:/opt/BTSorb/lib/openorb_ots-1.3.1.jar:/opt/BTSor
b/lib/openorb_pss-1.3.1.jar:/opt/BTSorb/lib/openorb_ins-1.3.1.jar:/opt/BTSorb/lib/openor
b_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.

The client implementation uses different semantics for policies and so forth. Also, the use of the `oo-run` command to launch an application is a matter of choice. It is provided as a simplification in the OpenORB package.

**Note**

If other tools are used, you must modify the IDL objects as well as the OpenORB files.

For example:

```
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, 2003 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("optiuser", "optiuser", 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 copmmand 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 copmmand 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.

IDL Overview

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 documents.

BTS 10200 Softswitch IDL

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.
- **Mgp**—The object used to communicate to certain media gateways. It uses simple strings to contain MGW-compatible text commands.

Cisco BTS 10200 Softswitch API

This section covers the actual API calls to the 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.

Cisco BTS 10200 Softswitch Security

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 30 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.

```
// MGW Profile Object ID-used to bind to the POA
byte[] btsSecuritymgwProfileId = "Bts10200_SecurityMgwProfile".getBytes();
```

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 to indicate the specific authorization of a particular user. The method signature is defined using the following code:

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

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

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

- **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.