CHAPTER 7

Sample Code for Application Program Interfaces

Revised: June 28, 2007, OL-13930-01

This section contains the README files for the following application program interfaces (APIs):

- Shell script for retrieving subscriber or provisioning information from an external source by using a script
- Java code for retrieving subscriber or provisioning information from an external source by using an HTTP application

The sections that describe scripts and code provide the following information:

- Location
- Purpose
- Requirements
- Description of components
- Directions for modifying the code
- Directions for troubleshooting the code
- Sample code that you can modify to meet your needs

This section contains the following topics:

Shell Script for Retrieving Subscriber or Provisioning Information

You can use a sample script written for various databases to retrieve subscriber or provisioning information from an external source. These scripts, that you can modify according to your needs, are in the README file in the following locations:

- **Linux and Solaris**
  - /opt/CSCOcbt/samples/db_script/oracle/
  - /opt/CSCOcbt/samples/db_script/sybase/
- **Windows**

- %CSCOCBT_ROOT%\samples\db_script\oracle\
- %CSCOCBT_ROOT%\samples\db_script\sybase\

This section provides a printed copy of that README file, formatted for this user guide.

Purpose of Shell Script

This section describes how to use scripts written against a Sybase database. The sybase directory contains sample scripts that you can modify to meet your needs. These scripts demonstrate how to:

- Receive requests from CBT
- Query data against a Sybase database
- Return data to CBT in the format specified in "Retrieving Data with a Script" section on page 37

Sample Script Requirements

The sample scripts have the following criteria for use:

- Sample scripts require Sybase running with CBT's database schema, which is created automatically during the CBT install.
- Sample scripts are written for the Solaris OS. To use the scripts on a Linux machine, see the "Troubleshooting the Script" section on page 7-3.

After trying the sample scripts, you can tailor them for your database or any other external data source.

Script and File Descriptions

The following list of sample scripts describes the purpose of each script or file:

- **get_subscriber**—Queries subscriber information from the database
- **get_provision**—Queries provisioning information from the database
- **insert_cmts**—Invokes insert_cmts.sql
- **insert_cmts.sql**—Inserts CMTSs and cable modems into the database and is invoked by insert_cmts
- **subscriberinfo.txt**—Contains subscriber information in ASCII text file format
- **add_col_name.pl**—Adds column names to the subscriber fields (for example, Customer Last Name) queried from get_subscriber

Modifying the Script

Perform the following steps to modify a script:

Step 1 Modify insert_cmts.sql to insert a list of CMTSs and cable modems into the database.

Step 2 Execute insert_cmts from the command line: `./insert_cmts`

Step 3 Modify subscriberinfo.txt to import subscriber information into CBT. For field descriptions, refer to online help for the Import Subscriber Data dialog box.

Step 4 Start CBT. Log in as an Admin user.

Step 5 Click **Import Subscriber Data**, enter the Subscriber Data File Location, and click **Start**.

Step 6 Click **Set External Interfaces**.

Step 7 Under Subscriber Information, select the **Script** radio button and enter the location of the get_subscriber filename (for example, /my_dir/get_subscriber) in the Script Location field.

Step 8 Under Provisioning Information, select the **Script** radio button and enter the location and the get_provision filename (for example, /my_dir/get_provision) in the Script Location field.

Troubleshooting the Script

- The get_subscriber and get_provision scripts can be run from the UNIX command line. The following examples show how to execute a script and the output the script produces:

  ./get_provision GET_PROVISION MAC 000216d5a0cf

  OUT_DATA=PROVISION^172.22.85.10^172.22.127.26^

  ./get_provision GET_MAC IP 172.22.127.26

  OUT_DATA=MAC^000216d5a0cf^

  ./get_subscriber GET_SUBSCRIBER MAC 000216d5a0cf

  OUT_DATA=SUBSCRIBER^AccId=ID000006^Name=Name000006^Phone=6172300006^Address=175 West Tasman^ClassOfService=Policy000006^FiberNode=User A000006^

  ./get_subscriber GET_MAC PHONE 5106663152

  OUT_DATA=MAC^000164ffeb95^000164ffc3c7^

  ./get_subscriber GET_ADDRESS MAC 000164ffc3c7^000164ffeb95^

  OUT_DATA=ADDRESS^000164ffc3c7=170 West Tasman Drive,95134^000164ffeb95=170 West Tasman Drive,95134^

- The get_subscriber script invokes add_col_name.pl. Modify the scriptDir to accurately reflect where add_col_name.pl is located.
- All scripts use /tmp as a temporary directory for creating temp files. Create the /tmp directory with read and write privileges for all users.
- If you are running the scripts on a Linux machine, make the following changes to the get_provision, get_subscriber, and insert_cmts scripts:
  - Modify commLib to /bin
  - Replace dbisql to dbisqlc
- If you get Invalid Data when using echo in Window's batch file:
  - Turn off echo:
    `@echo off`
  - If you are not using double quotes around the message, then the delimiter has to be "^^" instead of "^".

    For example:

    echo OUT_DATA=PROVISION^^172.22.85.10^^172.22.127.26^^

Sample Script Code

The following sample code is extracted from the get_subscriber script:

```
#!/bin/ksh
argc=$#
set -A argv $*
integer i=0
while let "i < $argc"; do
  case ${argv[$i]} in
    "GET_MAC") function=${argv[$i]};;
  esac
  let "i = i + 1"
done

#retrieving MAC based on Customer phone number
if [ "$function" = "GET_MAC" ] ; then

   str="SELECT MACAddress FROM SUBSCRIBERINFO where CusPhone='$in_param';";
   echo $str >> "$tmpDir/script.$filenameExt"
   str="OUTPUT TO $tmpDir/test.$filenameExt FORMAT ASCII;"
   echo $str >> "$tmpDir/script.$filenameExt"

   tmpi=`dbisqlc -q -c $dbAccess read $tmpDir/script.$filenameExt`
   rm $tmpDir/script.$filenameExt

   MAC=`$commLib/cat $tmpDir/test.$filenameExt | $commLib/sed s/\'//g`
   echo $MAC > $tmpDir/test.$filenameExt
   MAC=`$commLib/cat $tmpDir/test.$filenameExt | $commLib/sed 's/ /^/g'`

   tmp="OUT_DATA=MAC^$MAC^";
   if [ $tmp = "OUT_DATA=MAC^^" ] ; then
      echo "OUT_DATA=ERROR^No data found for $in_param^";
      exit;
   fi
   echo $tmp

   rm $tmpDir/test.$filenameExt
   exit;

fi
```

Java Code for Retrieving Subscriber or Provisioning Information

You can use an application that is running on an HTTP server to retrieve subscriber or provisioning information from an external source.

A sample application that you can modify according to your needs is in the README file in the following locations:

- **Linux and Solaris**
  - /opt/CSCOcbt/samples/db_script/oracle/
  - /opt/CSCOcbt/samples/db_script/sybase/
- **Windows**
  - %CSCOCBT_ROOT%\samples\db_script\oracle\

  - %CSCOCBT_ROOT%\samples\db_script\sybase\

This section provides a printed copy of that README file, formatted for this user guide.

Purpose of Java Code

This sample script file describes how to use Data Manager, a sample HTTP application. The HTTP directory contains this sample application that you can modify to meet your needs. Data Manager demonstrates how to:

- Receive requests from CBT
- Query data against an Oracle database
- Return the data to CBT in the format specified in the "Retrieving Data with an HTTP Application" section on page 39

Sample HTTP Application Requirements

The sample HTTP application:

- Requires the Oracle database running with the schema described in create_tbls.sql.
- Is written to use Oracle OCI, instead of the pure Java JDBC, to communicate with the database. To use the sample HTTP application on a Solaris machine, see the "Troubleshooting the Sample HTTP Application" section on page 7-8.

Module and File Descriptions

This section lists the modules and files used in Data Manager, the sample HTTP application. For a description, refer to a specific module or file. To modify Data Manager to suit your needs, refer to the next section, "Modifiable Components in the Data Manager Application."

- Java modules:
  - DataManager.java
  - Db.java
  - DbProps.java
  - DbQuery.java
  - GetResponse.java
  - HttpManager.java
  - HttpObj.java
  - HttpRequest.java
  - HttpResponse.java
  - HttpServer.java
  - LoadOracleJDBCDriver.java
  - Logger.java
  - PostResponse.java

  - ShutDown.java
- System and configuration files:
  - Makefile
  - Db-Vital.properties
  - Db-Sigma.properties
  - install
  - start_dm
  - stop_dm
- SQL files:
  - create_tbls.sql
  - insert_cmts.sql
  - insert_subscriber.sql

Modifiable Components in the Sample HTTP Application

- **create_tbls.sql, insert_cmts.sql, insert_subscriber.sql**—Scripts that create database tables and insert data into the database.
- **DbQuery.java**—Module that queries the data from the database. The following functions were written based on the schema described in create_tbls.sql:
  - getMacByIP
  - getMacByPhone
  - getProvInfo
  - getProvInfoExt
  - getFiberNode
  - getSubscriberInfo

  If create_tbls.sql is changed, the code for these functions must be changed accordingly.
- **Db-Sigma.properties**—Contains attributes of the subscriber database.
  Db-Vital.properties—Contains attributes of the provisioning database.

  If the provisioning and subscriber data tables reside in the same database, these two files must have identical information. In Data Manager, the provisioning and subscriber data is in two separate databases.

  For both files, the following settings must be changed accordingly:
  - db_sid
  - db_port
  - db_userid
  - db_password
- **Makefile**—Compiles the Java files.
- **start_dm**—Starts the Data Manager HTTP application.

  The current variable settings in start_dm are:

```
debug_flag=1
(Send debug messages to a log file.)
port=8012
(Change to the port of your choice.)
logDir= /opt/CSCOcbt/jakarta-tomcat-4.0.3/logs
(Specify the location where the log file should be created and changes should be
written.)
```

- **stop_dm**—Stops the Data Manager HTTP application.

  The following variable settings are in stop_dm:

```
ip=171.71.50.52
(IP address where the HTTP application is running.)
port=8012
(Port in which the HTTP application is running. This should be the same port specified
in the start_dm script.)
```

  For these three scripts (Makefile, start_dm, and stop_dm), the following properties must be updated according to the system environment:

  - ORACLE_HOME
  - JAVA_HOME
  - LIBRARY_PATH

Building and Running the Sample HTTP Application

1. Build class files and copy files to destination.

   From the command line, type:

   ./make

   ./install

2. Start the application.

   From the command line, type:

   /opt/CSCOcbt/bin/start_dm

3. Stop the application.

   From the command line, type:

   /opt/CSCOcbt/bin/stop_dm

4. Link Data Manager to CBT.
   a. Start CBT. Log in as an **Admin** user.
   b. Click **Set External Interfaces**.
   c. Under Subscriber Information, select the **HTTP** radio button and enter the URL for the application in the **URL Location** text box:

      http://*your_machine_name:port_num*/**DataManager**

5. Under Provision Information, select the **HTTP** radio button and enter the URL for the application in the **URL Location** text box:

   http://*your_machine_name:port_num*/**DataManager**

Note In the previous steps *port_num* is the port number that is configured in start_dm.

Troubleshooting the Sample HTTP Application

- Failed to create database connection.

  Make sure there is connectivity between the machine and the Oracle database. A quick way to test this is to invoke the Oracle SQL*Plus application. For example, assume that the following settings are in Db-Vital.properties:

  - *db_sid*=CBT
  - *db_userid*=user1
  - *db_password*=mypasswd

  From the command line, type:

  sqlplus user1/mypasswd@CBT

  If there is no response or an error occurs, check the tnsnames.ora setting.

- Makefile failed - javac not found.

  Make sure that JAVA_HOME in the Makefile points to the correct JAVA compiler.

- Makefile failed while linking or missing Oracle library error generated during run time.

  The sample HTTP application was written to use the Oracle OCI protocol for communicating with the database. Oracle OCI requires machine-dependent libraries, so make sure that the path to the libraries is set correctly in Makefile.

- Socket creation error.

  This error occurs when *port_num* is changed in start_dm and stop_dm. To correct it, restart the program /opt/CSCOcbt/bin/start_dm program.

- DbQuery returns error status.

  It is possible that some of the Oracle environment did not get set up correctly. To check this, review oracle.csh, update it as required, and source it. Then, run start_dm again.

Sample Java Code

1. Set up the HTTP Connection to receive a request from CBT (HttpManager.java).

```
HttpRequest = new HttpRequest()
```

2. Retrieve the message header (HttpRequest.java):

```
StringBuffer sb = new StringBuffer();
InputStream is = sock.getInputStream();
DataInputStream fromBrowser = new DataInputStream(is);
while (true) {
        msg = fromBrowser.readLine();
        if (msg.equals("")) break;
        sb.append(msg + nl);  // put back the '\n\r' for newline
}
```

3. Retrieve the message body (HttpRequest.java):

```
DataInputStream  fromBrowser = sock.getInputStream();
StringBuffer    sb = new StringBuffer();
```

```
    for (int x=0; x < len; x++) {
            // return int range 0-255, -1: end of stream
            i = fromBrowser.read();
            if (i == -1) break;
            // however, char is 16 bits for unicode 2 bytes
            // now use 2 bytes to hold just one byte data.
            sb.append((char)i);
            //System.out.println (sb);
    }
```

4. Parse the message (HttpObj.java):

```
    //parse and save message into Hashtable

        String req_command = decodeString(command);
        //System.out.println ("req_command=" + req_command);

        // Parse the data from the servlet
        StringTokenizer rdata = new StringTokenizer(req_command, "&");

        if (rdata == null)  return;

        //System.out.println ("data=" + data + ", rdata=" + rdata);
        while (rdata.hasMoreTokens()) {
            String cmd = rdata.nextToken();
            int index = cmd.indexOf("=");
            if (index == -1) return;
            String key = cmd.substring(0, index);
            String val = cmd.substring(index+1);
            hParams.put(key, val);
        }
```

5. Process the message (HttpManager.java & DataManager.java):

```
 //if POST message then process the message:
  if (req.getReqType().equals ("POST")) {
                req.fetchBody(sock_);
                hParams = req.getParams();
                resp = new PostResponse(req);
                String status = null;
                status = dm_.processCommand(req);
  ...
  }
//examine the request type and process the message accordingly:
Hashtable hParams = req.getParams();
String task = (String)hParams.get("REQUEST");
int status = 1;

        if (task.equals(GET_PROV_INFO)) {
            String sStr = "";
            String sType =  (String)hParams.get("SEARCH_TYPE");
            if (sType.equals("MAC"))
            {
               String sTmp =  (String)hParams.get("IN_PARAM");
               String sMac = sTmp.toLowerCase();
               Logger.debug("In Mac is " + sMac);
               sStr = DbQuery.getProvInfo(sMac);
               Logger.debug("Prov Info is " + sStr);
            }
.....
        }
```

```
  // retrieve information from the database (DbQuery.java):
    public static String getProvInfo(String Mac) {

        String query = "";
        String subscriberId =  "";
        String modemIp = "";
        String cmtsIp = "";
        Statement stmt = null;

           try {
               long start = (new java.util.Date()).getTime();

               start = (new java.util.Date()).getTime();
               stmt = vital_db_conn.createStatement();
              query = "select subscriberid, ipaddress,giaddr from device where deviceid =
'" + Mac + "'";
               Logger.debug(query);
               boolean bFound = false;
               ResultSet rs = stmt.executeQuery(query);
               while (rs.next()) {
                   bFound = true;
                   subscriberId = rs.getString(1);
                   modemIp = rs.getString(2);
                   cmtsIp = rs.getString(3);
                   rs.clearWarnings();
               }
               rs.close();
               rs = null;
               stmt.close();
               stmt = null;

            long end = (new java.util.Date()).getTime();
            Logger.debug("total msec to execute getProvInfo: " + (end - start));
            if (!bFound)
               return ("OUT_DATA=ERROR^" + Mac + " does not exist in the Provisioning
Database.^");
            String sStr = "OUT_DATA=PROVISION^" + cmtsIp + "^" + modemIp + "^";
            return (sStr);
           }
           catch (SQLException se) {
               String sError = "OUT_DATA=ERROR^" + se.getMessage() + "^";
               if (stmt != null)
               {
                  try {
                     stmt.close();
                  }
                  catch (Exception e)
                  {
                  }
                  stmt = null;
               }
               return sError;
           }
    }
```

6. Build a Post-HTTP Response to return data to CBT (HttpResponse.java):

```
        //send header to CBT
        OutputStream os = null;
        try {
                os = sock.getOutputStream();
        }
        catch (IOException e) {
                Logger.error("DataManager.HttpResponse:sendHeader: error: " + e);
                e.printStackTrace(DataManager.getLogPW());
                e.printStackTrace();
        }

        StringBuffer sb = new StringBuffer();
        sb.append ("HTTP/1.0 200 OK" + nl);
        sb.append ("Content-type: text/html" + nl);
        byte[] b = (sb.toString()).getBytes();
        try {
                os.write(b);
                os.flush();
        }
        catch (IOException e) {
                Logger.error ("DataManager.HttpResponse:sendHeader: error write: " + e);
                e.printStackTrace(DataManager.getLogPW());
                e.printStackTrace();
        }

Send result data off to CBT

        os.write(result_data);
        os.flush();
```