Cisco and Third-Party APIs

The following sections describe new features and changes that are pertinent to this release of the Cisco Unified Communications Manager APIs and the Cisco extensions to third-party APIs.:

Cisco Unified JTAPI

In Cisco Unified Communications Manager Releases 6.x and 5.x, if an application tries to conference two or more addresses on the same terminal, based on the order of participants in the request, an application may receive CiscoJtapiException.CONFERENCE_INVALID_PARTICIPANT for the conference request. The conference might then start successfully with some of the participants. In such cases, no guarantees exist which application joins the conference. The system creates the conference with only one of the addresses on a terminal. It ignores the other addresses.

This situation occurs in the following scenarios.

Scenario 1

Assume that B1 and B2 represent different addresses on the same terminal, TB.

A -> B1 – GC1

A -> B2 – GC2

A -> C – GC3

Assume that an application issues a conference request GC1.conference(GC2,GC3).

In Cisco Unified Communications Manager Releases 6.x and 5.x, the application receives CiscoJtapiException.CONFERENCE_INVALID_PARTICIPANT; however, A, B1, and C join into conference, and the following normal set of events in a conference scenario occur:

```
GC1 CiscoConferenceStartEv

GC2 TermConnDroppedEv TB
GC2 CallCtlTermConnDroppedEv TB
GC2 ConnDisconnectedEv B1
GC2 CallCtlConnDisconnectedEv B1
GC1 CallCtlTermConnTalkingEv TB
GC2 CiscoCallChangedEv

GC1 ConnCreatedEv C
GC1 ConnConnectedEv C
GC1 CallCtlConnEstablishedEv C
GC1 TermConnCreatedEv TC
```

```
GC1 TermConnActiveEv TC
GC1 CallCtlTermConnTalkingEv TC

GC2 TermConnDroppedEv TC
GC2 CallCtlTermConnDroppedEv TC
GC2 ConnDisconnectedEv C
GC2 CallCtlConnDisconnectedEv C
GC2 CallInvalidEv
GC1 CiscoConferenceEndEv
```

Scenario 2

Assume that B1 and B2 represent different addresses on the same terminal, TB.

A -> B1 – GC1

A -> B2 – GC2

A -> C – GC3

Assume that an application issues a conference request GC3.conference(GC1,GC2).

In Cisco Unified Communications Manager Releases 5.x, the application does not receive an exception and the request processes successfully. A, C, and B1 join the conference with the regular set of conference events.

In Cisco Unified Communications Manager releases 5.x 6.x, the application receives CiscoJtapiException.CONFERENCE_INVALID_PARTICIPANT; however, A, C, B1 join conference, and the normal set of conference events occurs.

Skinny Client Control Protocol (SCCP)

The following sections describe updates to the SCCP interface:

BLF Enhancements

This release supports the following variations of Busy Lamp Field (BLF):

- Notification BLF Alerting—When a call arrives on a monitored line, an alerting indication notifies the monitoring stations. The alerting indication consists of a BLF icon, a flashing LED, an alerting icon, and an audible alert.
- BLF Pickup—This enhancement allows a user to pick up a call from the monitoring station during the BLF Alerting state by pressing the corresponding BLF line button.

I-Hold

This feature provides visual distinction that shows which line placed a call on hold in a shared line configuration. With this enhancement, when a call is put on hold by the local line, the current LED behavior (flashing green) continues, and the remote line changes to a flashing red LED status.

Updated SCCP Messages

This release adds, modifies or deletes the following SCCP messages:

New messages

- StationFeatureStatV2Message—Sent from Cisco Unified Communications Manager
- StationFeatureStatReqMessage—Sent from Cisco Unified Communications Manager
- StationSetLampMessage—Sent from Cisco Unified Communications Manager
- StationStimulusMessage—Sent to Cisco Unified Communications Manager
- StationStartToneMessage—Sent from Cisco Unified Communications Manager

Modified message

- StationCallStateMessage—Modified StationDCallState ENUM

Deleted messages

- StationStartSessionTransmissionMessage
- StationStopSessionTransmissionMessage

Administrative XML Interface

The Administrative XML (AXL) interface uses the Tomcat web service that is running on the Cisco Unified Communication Manager Publisher to return client data. Several other Publisher features share this service, including Cisco Unified Communication Administration, Cisco IP Manager Assistant, Cisco Web Dialer, Cisco Extension Mobility, and the Bulk Administration Tool. The Cisco Unified Communications Manager Publisher includes enhancements that ensure that no single application can consume all available Tomcat resources.

The AXL interface now includes a throttling mechanism that limits the amount of data that client can return. The limit for data that gets returned in any single request specifies 8 MB. The limit for concurrent data requests specifies 16 MB, which can be split over any number of concurrent requests (for example, 8 concurrent requests with each requiring 2 MB of data to be returned; 4 concurrent requests with each requiring 4 MB of data to be returned; or any other combination not exceeding 8 MB per request or 16 MB concurrently).

This feature:

- Prevents AXL request processing from making the Tomcat service unresponsive.
- Allows critical applications such as the Cisco Unified Communications Manager Administration interface and logging in to and out of Cisco Extension Mobility to function even when heavy AXL queries are processed.
- Allows client applications to obtain the information that is requested.
- Maximizes interface throughput.
- Minimizes required changes to existing applications.

AXL Data Throttling

The following AXL methods now include data throttling:

- executeSQLQuery
- listDeviceByNameAndClass

- listDeviceByServiceName
- listPhoneByDescription
- listPhoneByName
- listUserByName

With data throttling, the AXL interface now returns the following message when the 8 MB limit is reached: “Query request too large. Total rows matched: <Matched Rows>. Suggested row fetch: less than <Number of Rows>.”

In addition, new <skip> and <first> tags for the List X methods allow developers to fetch the data requested.

Developers who use ExecuteSQLQuery should use standard SQL Skip and First tags in the request to retrieve the desired data set.

Interaction

This section describes how the interface responds in a variety of situations.

Example 1, Option 1: Client requests data that exceeds 8 MB

Server response: “Query request too large. Total rows matched: <Matched Rows>. Suggested row fetch: less than <Number of Rows>”

<Number of Rows> specifies the suggested number of rows that a single request can return to keep the data exchange under the 8 MB limit. Clients should use <Matched Rows> to determine the number of iterations that are required to retrieve the complete data set that a query tries to fetch.

Client response, Option 1:

1. Client logic analyzes the server response and obtains the value of <Row Fetch>.
2. Client stores the <Row Fetch> value in a constant that depicts Row Fetch Step Size. For this example, rowFetchStepSize represents the constant name.
3. Client logic generates the request in parts based on the <Row Fetch> value.
4. Client keeps track of the number of rows that were received in the previous request. For this example, this information exists in a variable that is named prevRows.
5. Client keeps track of the total number of rows that are fetched. For this example, this information exists in a variable that is named totalRowFetch.
6. Before sending the next request, client checks whether prevRows == <Row Fetch>.
7. If the check returns true, continue the request generation loop. Otherwise, break from the loop.

Example 1, Option 2: Client requests data that exceeds 8 MB

Server response: “Query request too large. Total rows matched: <Matched Rows>. Suggested row fetch: less than <Number of Rows>”

<Number of Rows> specifies the suggested number of rows that a single request can return to keep the data exchange under the 8-MB limit. Clients should use <Matched Rows> to determine the number of iterations that are required to retrieve the complete data set that a query tries to fetch.

Client response, Option 2:

1. Client logic analyzes the server response and obtains the value of <Row Fetch> and <totalRows>.
2. Client stores the <Row Fetch> and <totalRows> values in variables. For this example, the variable names comprise rowFetchStepSize and countOfRows.

Note Alternatively, the client can execute a query for count(*) to obtain the number of rows that can be fetched from the database.

3. Client logic calculates the number of iterations based on the values that are stored in rowFetchStepSize and countOfRows.
4. Client logic declares two variables: “skip” (with the initial value of 0) and “first” (with initial value set to rowFetchStepSize).
5. Client starts the iteration.
6. Client logic generates the request based on “skip” and “first” at every iteration.
7. Client modifies the value of “skip” and “first” at every iteration.
8. Client checks the response at every iteration.
9. If the response is a MemoryConstraintException, the client waits until the requests in progress completes, then continues the iteration.

   If the response is not a MemoryConstraintException, the client continues the iteration.
10. Iterations continue until the <number Of Iterations> value is reached.

Example 2: Client sends single 8-MB request

Server responds with requested data.

Example 3: Client sends two 8-MB requests simultaneously

Server responds with requested data.

Example 4: Client sends more than two 8-MB requests simultaneously

The server processes the first two requests. Other concurrent requests that may be received generate this exception: “Maximum AXL Memory Allocation Consumed. Please retry once requests in progress have completed.”

Client response:

1. Client waits for the requests that are being processed to complete and then sends the request again.

   Cisco recommends that applications wait 2 to 3 minutes before resubmitting the request.
2. Client logic must track the requests that fail and send them again.

Example 5: Concurrent data requests reach 16-MB limit

This example applies to all AXL methods.

This situation returns MemoryConstraintException AXL error code 5009: “Maximum AXL Memory Allocation Consumed. Please retry once requests in progress have completed.”

Cisco recommends that applications wait 2 to 3 minutes after receiving this message before resubmitting the request.

Using <skip> and <first> Tags in List APIs

The new <skip> and <first> tags provide additional functionality when retrieving data by using the List methods. If a List Response exceeds 8 MB, the client can fetch data in sets of rows by using a combination of these tags. These tags provide navigation functionality. Be aware that they are not mandatory and have no default values.

In addition,

- Negative values for the <skip> or <first> tags cause a SQL syntax error.
- If the <skip> tag is not mentioned or is empty and the <first> tag is not mentioned or is empty, the default query or full query that pertains to the List API executes.
- If the <skip> tag is not mentioned or is empty and the <first> tag has some positive value (for example, “m”), a query that skips the “zero” row and fetches the first “m” rows executes.
- If the <skip> tag and <first> tag value are both positive (for example, “n” and “m,” respectively), a query that skips “n” rows and fetches “m” rows executes.
- The <skip> tag can only be used when the <first> tag is speficied and the <searchLimit> tag value is zero.
- The <first> tag can be used when the <searchLimit> tag value is zero.

Suggested Use of <skip> and <first> Tags in List APIs

If a client request exceeds 8 MB of data, the server responds: “Total rows matched: <Matched Rows>. Suggested row fetch: less than <Number of Rows>.”

<Number of Rows> specifies the suggested number of rows that a single request can return to keep the data exchange under the 8 MB limit. Clients should use <Matched Rows> to determine the number of iterations that are required to retrieve the complete data set that a query tries to fetch.

Client response:

1. Client logic analyzes the server response and obtains the value of <Row Fetch>.
2. Client stores the <Row Fetch> value in a constant depicting Row Fetch Step Size. For this example, the constant name is rowFetchStepSize.
3. Client executes a query for count(*) to obtain the number of rows that can be fetched from database. (Consider this step as required only if you are not using the exception to exit the loop.)
   a. Client modifies the tags <first> and <skip> in every iteration of the row fetch.
   b. The first iteration starts with <skip>0<skip> and <first> rowFetchStepSize</first>.
   c. Subsequent row fetch iteration have <skip>”first” tag value from previous iteration</skip> and <first>previous iteration “first” tag value + rowFetchStepSize</first>.
4. Before each iteration, client checks for the condition <skip> tag value < count(*) value. If <skip> tag value >= count(*) value, the iteration is trying to fetch more rows than the existing number of rows in the database. In this case, break from the loop. Otherwise, the SQL Error (No Current Row) occurs, which you can use to break from loop.

Sample Code for Use of <skip> and <first> Tags in List APIs

Example 1

If the recommended RowFetch is X = 100 for ListPhoneByName, the client should use this logic:

```
StepSize = RowFetch (X = 9999)
//Two variables to store the skip and first values.
Int skip = 0;
Int first = StepSize;
//start the iteration
While (1)
{
 Try {
```

```
//A function to Modify the values of <first> and <skip> in the request with variables
values defined above
modifyRequest(FilePath);
//Sending the modified request
Response = SendExecuteSQLQuery (select SKIP <SkipCount> First <StepSize> * from endusers;)
//Modify the variables
Skip+ = StepSize;
First+= StepSize;
Check for fault_message from reply
//An SQL Exception from server while trying to fetch rows greater than that present in
database
If(fault_message .contains("No Current Row Found")){
Break;
}
Else{
Continue the loop;
}
} /* end of while*/
```

Example 2

```
#Declare variable for first and skip
long skip = 0;
long first = suggestiveRowFetch;
#Calculate number of iterations required
float precision = totalRowFetch/suggestiveRowFetch;
calculate the decimal point of variable precision
if decimal point == 0;
no. Of Iterations = totalRowFetch/suggestiveRowFetch;
if decimal point <5
no. Of Iterations = Math.round(totalRowFetch/suggestiveRowFetch)+1;
if decimal point >5
no. Of Iterations = Math.round(totalRowFetch/suggestiveRowFetch);
****Here (no. Of Iterations = 15)********
#iterator
int iLoop = 1;

while( iLoop <=no. Of Iterations){

#modify the values of first and skip in the query
String mdSQLQuery = modifySQLQuery(skip,first); /* Its like select skip 0 first 2000 *
from device for first loop*/
#send the request and get response
response = sendRequest(mdSQLQuery);
#update the skip and first variables
skip+=suggestiveRowFetch;
first+=suggestiveRowFetch;
#check for response
if(response contains MemoryConstraintException){
#undo the variable modification to get  attributes (i.e skip and first) of the query ,
that failed with an exception.
skip - =suggestiveRowFetch;
first - =suggestiveRowFetch;
wait till the requests in Progress Gets Processed;
continue the loop;

}
else{
iLoop++;
}
}
```

Sample Code for ExecuteSQLQuery

Example 1

To obtain the first X rows and then next X rows, a client should send queries as described in this section. For example, the client should use the following logic if the recommended RowFetch is X = 9999 for the query “select * from endusers;”:

```
StepSize = RowFetch (X = 9999)
SkipCount = 0
 While (1)
{
 Try {
Response = SendExecuteSQLQuery (select SKIP <SkipCount> First <StepSize> * from endusers;)
RowsReturned = Rowcount (Response);
If RowsReturned < StepSize
    Break;  /* All the rows have been fetched*/
Else
    SkipCount = SkipCount + StepSize /*Increase the SkipCount to get the next set of
rows*/
}
Catch for any exception
{
    Take appropriate action based on Exception
    Break;
}
} /* end of while*/
```

Example 2

```
#sql query to be executed
sqlQuery = "select * from device";
#send the query to server
response = sendRequest(sqlQuery);
#what is in response?
processResponse();
**** Response contains an Exception: "Query request too large.Suggestive row fetch 2000
rows.Total row fetch 30000" ****
#Declare two variables
long suggestiveRowFetch = 0; /*contains Suggestive RowFetch Count*/
long totalRowFetch = 0;      /*contains Total RowFetch*/
#fetch the two values and store it into variables.
parseExceptionMessage();
```

Testing Suggestions

This section provides tests that you can run to check various operations.

Testing ListPhoneByDescription and ListPhoneByName Methods

To test the ListPhoneByDescription and ListPhoneByName AXL methods, follow these steps:

Step 1 Populate database fields (<name>, <tkproduct>, <tkmodel>) for devices/phones to the maximum field size, which is 15 characters each.

Step 2 Populate the database with more than 60,000 devices.

Step 3 Execute the ListPhoneByDescription and ListPhoneByName AXL methods. The resulting data set has a response that is greater than 8 MB.

Expected Results: The interface returns "<API name> API request exceeds Threshold Limit. Total rows matched: <Matched Rows>. Suggested row fetch: less than <Number of Rows>." <API name> specifies the name of the API method.

Testing listDeviceByNameAndClass and listDeviceByServiceName Methods

To test the listDeviceByNameAndClass and listDeviceByServiceName AXL methods, follow these steps:

Step 1 Populate database fields (<name>, <tkproduct>, <tkmodel>) for devices/phones to the maximum field size, which is 15 characters each.

Step 2 Populate the database with more than 75,000 devices.

Step 3 Execute the listDeviceByNameAndClass and listDeviceByServiceName AXL methods. The resulting data set has a response that is greater than 8 MB.

Expected Results: The interface returns "<API name> API request exceeds Threshold Limit. Total rows matched: <Matched Rows>. Suggested row fetch: less than <Number of Rows>." <API name> specifies the name of the API method.

Testing the ExecuteSQLQuery AXL Method

To test the ExecuteSQLQuery AXL method, follow these steps:

Step 1 Populate database fields (<name>, <tkproduct>, <tkmodel>) for devices/phones to the maximum field size, which is 15 characters each.

Step 2 Create a SQL Select statement that retrieves this data.

Step 3 When the exception occurs, pick up the recommended Row Fetch Count and send the modified executeSQLQuery (in loop) by using the recommended row fetch count.

Expected Results: Demonstrate that Row Fetch Count logic works and, at the end loop, client can retrieve the entire set of required data from Cisco Unified Communications Manager.

Verifying that Tomcat Resources Are Protected

To verify that Tomcat resources are protected (Publisher server continues to operate under heavy AXL load):

Step 1 Write a script that generates a load on the AXL interface. Run executeSQLQuery to generate a response slightly less than 8 MB.

Step 2 Client runs this executeSQLQuery in a loop as soon as the transaction completes (both passed and failed transactions) for 1 hour. The client also notes the time that the request was sent, time that the response was received, and whether the response passed or failed.

Step 3 Create four instances of this script and make them run simultaneously against the same Cisco Unified Communications Manager Publisher.

Step 4 During the 1 hour load test, monitor the Tomcat JVM-related RTMT counters. In addition using the Cisco Unified Communications Manager Administration interface, check whether you can list the devices on the system.

Step 5 At the end of the test, document the request and response times of the four clients on a timescale and note whether they succeeded or failed.

This analysis indicates whether the total responses that are processed by the AXL interface are within 16 MB. Every third concurrent request should have been rejected. By looking at the JVM on the Tomcat server that was available during the test, you can determine whether there was enough JVM to allow other applications to function properly.

Cisco Web Dialer API

The following sections describe updates to the Cisco Web Dialer API:

User Interface Enhancements

The following enhancements apply to the Make Call window:

- WebDialer Preferences moved to the Make Call page and show consolidated information.
- The MAC address displays only when a user is associated with multiple devices of the same type.
- Partition information does not display unless duplicate DNs are configured on the same device.
- The Calling Line option displays only when the device has multiple lines.
- The Extension Mobility option displays only when this feature is enabled for the user.

The following enhancements apply to the Hang Up window:

- The Destination User Name displays if it is available.
- Partition information does not display.

Hidden Proxy User Credentials

Credentials no longer display in clear text in a URL. For example, a URL that displayed as follows in previous versions

https://wdserver.com/webdialer/Webdialer?cmd=doSetProfile&destination=&loc=en-us&red=null&uid=wd&pwd=xyz

now displays as

https://wdserver.com/webdialer/Webdialer

As a result, developers who use the browser interface should use the HTTP POST method to pass the parameters. For example, this approach reduces the delay when Web Dialer converts GET parameters to POST:

```
FORM action="https://10.77.31.238/webdialer/Webdialer" method="post">
   <P>
    <INPUT type="hidden" name="destination" value="666">
    </P>
</FORM>
```

Redirector Throttling

Redirector throttling supports up to 8 sessions/requests per second. The system throws the following error if it exceeds the throttle:

HTTP Status 503 - Service temporarily unavailable, please try again later

Configuration Changes

The following configuration changes occurred:

- List of Web Dialers setting displays on the Application server window in Cisco Unified Communications Manager Administration.
- Existing settings migrate automatically during an upgrade.
- No restriction exists on the number of Cisco Web Dialer servers that you can add.
- You can easily associate Cisco Web Dialers servers with Redirectors. To do so, take the following actions in the Application Server Configuration window in Cisco Unified Communications Manager Administration:
  1. In the Hostname or IP Address field, enter the host name or the IP address of the Cisco Web Dialer Server. Include the port number if applicable.
  2. Assign the Cisco Web Dialer server to a particular Redirector Node. Choose Use None for a cluster-wide setting.

Update to Endcall SOAP Method

The SOAP method Endcall update means that it no longer drops all calls on a device. Endcall now ends an active call, but not a call on hold. EndCall can be used only when at least one call is active on the device.

Internet Explorer 7 Support

When you exit Cisco Web Dialer when you run it with Internet Explorer 7, the pop-up window that prompts for confirmation no longer displays. This change does not affect the programming interface.