API Usage and Conventions

The Cisco Packaged Contact Center Enterprise Developers Guide uses REST-based API functions accessed over HTTP. Five API functions are supported; each is mapped to an HTTP operation. Not all API functions are used for all objects.

The components are:

- create (http POST)—creates an object in the database and returns a response that contains the URL reference to the newly created object. This code sample shows the URL reference returned for a newly-created Bucket Interval: *HTTP/1.1 201 Created Location: https://192.168.0.1/unifiedconfig/config/bucketinterval/100162*. The id for the Bucket Interval is *100162*. This URL reference can be used to retrieve the object with an HTTP GET.
- delete (http DELETE)—deletes the object.
- get (http GET)—returns data for an object. For objects for which there are multiple records, GET takes an <id>.
- update (http PUT)—modifies an object. For some objects, PUT must include a changeStamp, on page 5, but all other parameters are optional for PUT. *
- list (http GET)—for objects for which there can be multiple records, returns a list.

The POST and PUT operations take a payload for which the input format is XML. GET and DELETE calls do not take a payload. All output is provided as XML when there is a response other than HTTP headers.

XML is case sensitive. When XML data is sent to the server, the tag names must match. <Name> and <name> are two different XML elements. If a payload contains duplicate fields, only one is transmitted. The duplicates are ignored.

Note: There are two types of URLs used:

- Absolute URLs
- Relative URLs, with <refURL> tags

Whereas absolute URLs are used as the target of the POST/PUT/GET/DELETE operations, <refURL> tags are sent or returned as part of the XML body.

So, the location header that is returned during a create has the absolute URL format, while the relative URL format is seen in all the XML that is passed in or returned from the REST calls within the <refURL> </refURL> tag.

The URL format for each type looks like this:

- Absolute URLs: https://<server_address>/unifiedconfig/config/...
- Relative URLs, with <refURL> tags: /unifiedconfig/config/...

Note **The size of the Request Payload for all APIs must be less than or equal to 5 MB**.

Change Log

This section notes the new and changed APIs in this release.

API	See	Notes
Congestion Control	Congestion Control API	Re-added into API documentation.
Network VRU Script	Network VRU Script APIs	Added routingType field.

General Usage

Access

Administrators who are in the Active Directory Config Security Group or Setup Security Group have full access to the Cisco Packaged Contact Center Enterprise APIs, unless that access has been limited by the Feature Control Set List Tool and the User List Tool. These tools are Unified CCE Configuration Manager tools, used together to establish and limit access to the Cisco Packaged Contact Center Enterprise administration tools—both the user interface and APIs—and to Unified CCE Configuration Manager. Note that the Administrator user name should be in the form of a Fully Qualified Domain Name (FQDN).

The Feature Control Set List Tool is used to create a Feature Control Set, while the User List Tool associates that Feature Control Set with users. The Feature Control Set List Tool establishes access by marking check boxes for the application names on the Feature Control Set and denies access by leaving the boxes unchecked. The User List Tool can associate a Feature Control Set with a user and/or limit access to read-only.

Full access to a Cisco Packaged Contact Center Enterprise API assumes that permissions are not limited by a Feature Control Set or by a read-only setting on the User List Tool.

For example, if Agent Explorer is unchecked in a Feature Control Set List and the user is associated with that list, the user is restricted from the Agent API. Likewise, if Attributes is unchecked in a Feature Control Set List and the user is associated with that list, the user is restricted from the Attribute API.

Note: A user that is restricted from an API cannot make changes to that API, but the user can still read it.

Note: Most application names on the Feature Control Set List *do not* correspond to Cisco Packaged Contact Center Enterprise APIs. The following table calls out the ones that apply.

API:	**Application Name in the Feature Control Set List:**
Agent	Agent Explorer
Attribute	Attribute
Agent Desk Settings	Agent Desk Settings List
Precision Queue	Precision Queue
Skill Groups	Skill Group Explorer
Agent Team	List Agent Team
Reason Code	Reason Code List
Bucket Intervals	Bucket Intervals List
Call Type	Call Type List
Dialed Number	Dialed Number/Script Selector List
Expanded Call Variable	Expanded Call Variable List
Network VRU Script	Network VRU Script List
Bulk Job	Dialed Number Bulk Edit AND Agent Bulk Edit
Congestion Control, Deployment Type Info, and Agent State Trace	System Information

It is important to note also that users who have Feature Control Set limitations and read-only cannot see the tools that the Feature Control Set excludes. They can see the other tool, but cannot make changes in those tools. For example, if Agent Explorer is checked, but read-only is checked for that user in the User List, the user can run Gets and Lists only.

Supervisor Access

In the Webconfig system you can log in either as an Administrator using the Fully Qualified Domain Name (FQDN), or as a Supervisor using the Agent username field.

Supervisors have limited access to and restricted usage of the APIs.

The following table outlines the APIs that Supervisors can access and the associated restrictions. Note that using any methods not listed in the Method column for each API will return a 405 (Method not supported) error. Trying to access any APIs not listed in the table below will return a 404 (Not found) error.

Table 1: Supervisor Access Restrictions

API	Access Level	Method	Additional Restrictions
Agent Team	Read Only	List or get	
Agent	Read and Update	List, get, update	Supervisors that call the Agent list API only see agents on their team(s). Supervisors that try to use the Agent get or update APIs for an agent not on their team(s) get a 404 (Not found) error. When updating an Agent API, Supervisors can change only the following fields: • Skill Groups • Default Skill Group • Attributes • Password
Skill Group	Read and Update	List, get, update	Supervisors are only allowed to update the list of agents that are members of the skill group. The supervisor can only add or remove agents from the skill group that are on their team(s).
Attribute	Read Only	List or get	
Precision Queue	Read Only	List or get	

Object ID

Object ID <id>: Using http POST to create all objects generates and returns an *id* for the object.

The DELETE, GET, and PUT operations for these objects are performed using the object id in the REST URL. For example:

- Use this URL to view results for a specified Bucket Interval:
- ◦ *https://<ServerIP>/unifiedconfig/config/bucketinterval/<id>/results*
- Use this URL to delete a Bucket Interval:
- ◦ *https://<ServerIP>/unifiedconfig/config/bucketinterval/<id>*.

Use the List(GET) function to identify the object ids.

```
<results>
<pageInfo>
....
</pageInfo>
<bucketIntervals>
<bucketInterval>...</bucketInterval>
<bucketInterval>...</bucketInterval>
</bucketIntervals>
</results>
```

changeStamp

A changeStamp is a required parameter for the body of a PUT (update) operation for objects.

If you do not provide a changeStamp, the update fails. This mechanism is in place so that two clients cannot edit the record at the same time.

If the update is successful, the database increments the changeStamp by 1.

Passwords

For security, the APIs do not return passwords in cleartext. Password elements are masked (******).

HTTP Responses

All errors are returned as HTTP 1.1 Status Codes. The common codes used by the APIs are:

- **200 OK**: Success
- **201 Created**: The requested item was created.
- **202 Accepted**: The request was accepted. Generally, a URL is provided to obtain additional details, for example, for polling the OAuth status.
- **400 Bad Request**: The request is invalid. Information returned in the ApiErrors message – example below – shows more details.
- **401 Unauthorized**: The authentication credentials were not supplied or were incorrect.
- **404 Not Found**: The URI requested does not exist on the server.
- **405 Method Not Allowed**: The method specified in the Request-Line is not allowed for the resource identified by the Request-URI.

- **500 Internal Server Error**: There is a problem on the server. Submit a post to the Forum explaining what you did and the response sent from the server.

Field specific and database errors are provided in an XML error message with the format:

```
<apiErrors>
    <apiError>
        <errorType>Type of Error</errorType>
        <errorData>Field Error Occurred</errorData>
        <errorMessage>A Description of the Error</errorMessage>
        <errorDetail>Extra information about the Error</errorDetail>
    </apiError>
<apiErrors>
```

ErrorDetail Examples

An example of an **errorDetail** field for an error type such as invalidInput.outOfRange is as follows:

```
<errorDetail>
    <min>0</min>
    <max>5</max>
</errorDetail>
```

An example of an **errorDetail** field for error types such as invalidInput.fieldLengthExceeded, limitExceeded.expandedCallVariableSize, and limitExceeded.totalExpandedCallVariableSize is:

```
<errorDetail>
    <max>5</max>
</errorDetail>
```

Note the preceding list of error types is not a comprehensive list and is given only as an example.

ErrorDetail References

Errors of type **referenceViolation** include the following kinds of error detail fields, for example: totalCount, totalShown, referenceType, name, refURL, id, and deleted.

For example, if you try to delete a Bucket Interval that is referenced by a Call Type, the error details look like this:

```
<errorDetail>
    <totalCount>1000</totalCount>
    <totalShown>5</totalShown>
    <referenceType>callType</referenceType>
    <references>
        <reference>
            <name>callType1</name>
            <refURL>/unifiedconfig/config/calltype/5000</refURL>
        </reference>
       <reference>
            <name>callType2</name>
            <refURL>/unifiedconfig/config/calltype/5001</refURL>
        </reference>
       <reference>
            <name>callType3</name>
            <refURL>/unifiedconfig/config/calltype/5002</refURL>
        </reference>
       <reference>
            <name>callType4</name>
            <refURL>/unifiedconfig/config/calltype/5003</refURL>
        </reference>
       <reference>
            <name>callType5</name>
            <refURL>/unifiedconfig/config/calltype/5004</refURL>
        </reference>
    </references>
</errorDetail>
```

ErrorDetail Script References

For items that are referenced by the script editor, the following detail fields are included for each Master_Script entry: name, id, and versions.

API Behavior

For any field except **list** elements, you can specify the same attribute more than once. However, the API takes the last attribute that you specify for that field.

For example, if you create an Agent using the following XML:

```
<agent>
<agentId>00370</agentId>
<description>bling</description>
<person>
   <firstName>fred</firstName>
   <firstName>bill</firstName>
   <lastName>smithx</lastName>
   <userName>fsmithax</userName>
   <password>freddieboy</password>
   <loginEnabled>true</loginEnabled>
   <changeStamp>0</changeStamp>
</person>
</agent>
```

Notice that <firstName> is specified twice:

```
   <firstName>fred</firstName>
   <firstName>bill</firstName>
```

The API takes the second <firstName> attribute and sets the Agent's first name to:

```
bill
```

Be aware that this type of behavior is common to all the APIs.

Internationalization

In some of the fields in the APIs, if you enter characters that are not supported by the database, such as native characters, for example, an error is returned that states: **The system does not support these characters**. The fields include <description> in any of the APIs that have a <description> field, and the following fields in Agent API: <first.Name> and <lastName>.

For information on how to configure your system to support native character sets, see the latest version of *Installing and Configuring Cisco Packaged Contact Center Enterprise*. Got to: http://www.cisco.com/en/US/products/ps12586/prod_technical_reference_list.html

Pagination

The pagination of the API provides information about how many objects are in the database, as well as pointers to the first, last, previous, and next page of items, if available.

This section outlines the pagination parameters, shows a sample response, and describes the fields that are returned in the response. It also provides important notes about pagination.

Parameters

The following table shows the pagination parameters that you can set.

Parameter Name	Explanation	Notes
startIndex	Specifies the index of the element, at which to start.	Zero-based: 0 is the first element. DEFAULT = 0.
resultsPerPage	Specifies the number of elements to retrieve.	MIN=1. DEFAULT=25 MAX=100.

Note The following is an example of how to use the pagination parameters when listing a specific element type:

- **https://<server>/unifiedconfig/config/bucketinterval?startIndex=0&resultsPerPage=5**

Response

The following shows an example XML response:

Example XML Response:

```
<pageInfo>
      <resultsPerPage>2</resultsPerPage>
      <startindex>0</startIndex>
      <totalResults>10</totalResults>
       <firstPage>
http://<server>/bucketIntervals/?resultsPerPage=2</firstPage>
       <lastPage>
http://<server>/bucketIntervals/?startIndex=8&resultsPerPage=2</lastPage>

      <prevPage/>
      <nextPage>
http://<server>/bucketIntervals/?startIndex=2&resultsPerPage=2</nextPage>

 </pageInfo>

<bucketIntervals>
   <bucketInterval/>
   <bucketInterval/>
</bucketIntervals>
```

Response Fields

The following table shows the fields that are returned in the response.

Field	Description	Example
totalResults	Total number of elements in the database.	<totalResults></totalResults>

Field	Description	Example
resultsPerPage	Number of items requested per page.	<resultsPerPage></resultsPerPage>
startIndex	The index of the first element returned.	<startIndex></startIndex>
nextPage	refURL to next page.	<nextPage></nextPage>
prevPage	refURL to previous page.	<prevPage></prevPage>
firstPage	refURL to first page.	<firstPage></firstPage>
lastPage	refURL to last page.	<lastPage></lastPage>
searchTerm	String value.	<searchTerm></searchTerm>
sortTerm	String value.	<sortTerm></sortTerm>

Important Notes

The following is a list of caveats and important notes about pagination.

- If you request a **startIndex** that is greater than total items, a full last page is returned.
- The **lastPage** should always return a full last page.
- The **firstPage** should always return a full first page (starting at 0).
- The **nextPage** is null if there is no **nextPage** (on last page).
- The **prevPage** is null if there is no **prevPage** (on first page).

Permissions Information

To facilitate making gadgets read-only for Supervisors, the APIs include permissions information, which indicates the operations that the user is allowed to perform.

The <permissionInfo> data is returned in the XML response when you perform a GET using the get/list operation. Here is an example:

```
<results>
    <pageInfo>...</pageInfo>
    <globalInfo>...</globalInfo>
    <permissionInfo>
        <canCreate>false</canCreate>
        <canUpdate>true</canUpdate>
        <canDelete>false</canDelete>
    </permissionInfo>
</results>

<congestionControl>
```

```
    <permissionInfo>
        <canUpdate>true</canUpdate>
        <canChangeDeploymentType>true</canChangeDeploymentType>
    </permissionInfo>
</congestionControl>
```

The <canCreate>, <canUpdate> and <canDelete> tags correspond to create, update and delete operations. You can only perform one of these operations if the corresponding tag is set to true.

If an API does not support a given operation, the corresponding permission is omitted. For instance, the Bulk Job API does not support updates, so it does not have a <canUpdate> tag.

If an API does not support any write operations, the <permissionInfo> tag is omitted entirely. This applies to the Active Directory Domain API, for example, because it cannot be modified in any way.

Search API

This section provides an overview of Search API, defines the search parameter, shows a search example, and outlines the default search values for existing configuration objects.

Overview

Search API has two parameters:

- **q**, where **q=<search_string>**
- **ignoreSearchErrors**, where **ignoreSearchErrors=[true|false]**

The **q** parameter is an optional search parameter taken by the various list API commands. It limits returned results to the configuration objects that match the search string.

You can perform a search on a predefined set of default fields for each configuration object. Typically, this is the **name** and **description** field, or the object's equivalent.

If **ignoreSearchErrors** is set to true, an invalid search string results in an empty list being returned rather than an API error. If this parameter is missing, it defaults to false. The setting **ignoreSearchErrors=true** should be specified if a client depends on list operations always succeeding for correct operation and/or has client-side search validation.

Search is subject to the following restrictions:

- Case-insensitive.
- String-only searches.
- Sql wildcards are not supported.
- "Contains" - the <search_string> will match any part of the default fields.
- The <search_string> is treated as a single string.
- An "or" search with a match on any of the default fields returning that record.

The search criteria are applied before the pagination parameters, so that pagination's **totalResults** value lists the total number of elements in the database that meet the search criteria.

Example

For example, a search for all the Call Types whose name or description contains "Supervisor" would be as follows:

https://<server>/unifiedconfig/config/calltype?q=supervisor

XML Returned

The following XML content is returned when the Search API is called.

```
<results>
   <pageInfo>
     <sortTerm>name</sortTerm>
     <searchTerm>supervisor</searchTerm>
     <firstPage>

[https://1.1.1.1/unifiedconfig/config/calltype?q=supervisor&sort=name%20asc&resultsPerPage=25]

     </firstPage>
     <lastPage>

[https://1.1.1.1/unifiedconfig/config/calltype?q=supervisor&sort=name%20asc&startIndex=0&
       resultsPerPage=25]
     </lastPage>
     <resultsPerPage>25</resultsPerPage>
     <startIndex>0</startIndex>
     <totalResults>1</totalResults>
   </pageInfo>
  <callTypes>
    <callType>
      <changeStamp>2</changeStamp>
      <refURL>[/unifiedconfig/config/calltype/5001]</refURL>
      <description>Used for Supervisor and Emergency Assist</description>
      <name>Assist</name>
    </callType>
  </callTypes>
</results>
```

Note if you include Sort, a <sortTerm> tagged value is returned.

Default Search Values

The following table shows the default search values for existing configuration objects.

Configuration Object	Default Search Value
agent	• agentId • description • person.firstName • person.lastName • person.userName

agentDeskSettings	• name • description
agentTeam	• name • description
attribute	• name • description
bucketIntervals	• name
bulkJob	• description
callType	• name • description • id
dialedNumber	• dialedNumberString • description
expandedCallVariable	• name • description
networkVruScript	• name • description
precisionQueue	• name • description • id
reasonCode	• text • description

skillGroup	• name • description

Sort API

This section provides an overview of Sort API, defines the sort parameter, shows a sort example and the allowable sort attributes, and discusses error results.

Overview

You can sort **list** API results for each configuration object, in either an ascending or descending manner.

The parameter is: **sort=<attributeName> [asc|desc]**, where:

- **attributeName** is the name of the field as returned in the XML and is case-sensitive.
- **asc|desc** are optional and case-insensitive.
- **asc** stands for *ascending* sort, which is the default.
- **desc** stands for *descending* sort.

Note Only the first sort argument on a GET query string is used to perform the sort operation. The others are ignored.

The sort option is applied after the search parameter and before the pagination parameters. The sort option can be specified by itself. The default sort field is the *name* field or equivalent.

Strings are returned in linguistic sort order. For example, *Alpha*, *abel*, *Beta*, *bagel* is sorted as:

- abel
- Alpha
- bagel
- Beta

Integer fields are sorted in integer order, not linguistic order. For example: 6, 12, 100 is sorted as 6, 12, 100.

Example

For example, to find all the CallTypes whose name or description contains *supervisor*, and to sort ascending by *name*:

```
https://<server>/unifiedconfig/config/calltype?q=supervisor&sort=name
```

And, to find all the Dialed Numbers and sort descending by *description*:

```
https://<server>/unifiedconfig/config/dialednumber?sort=description desc
```

XML Returned

The following XML content is returned when the Search API is called:

```
<results>
   <pageInfo>
     <sortTerm>name</sortTerm>
     <searchTerm>supervisor</searchTerm>
     <firstPage>

[https://1.1.1.1/unifiedconfig/config/calltype?q=supervisor&sort=name%20asc&resultsPerPage=25]

     </firstPage>
     <lastPage>

[https://1.1.1.1/unifiedconfig/config/calltype?q=supervisor&sort=name%20asc&startIndex=0&
       resultsPerPage=25]
     </lastPage>
     <resultsPerPage>25</resultsPerPage>
     <startIndex>0</startIndex>
     <totalResults>1</totalResults>
   </pageInfo>
  <callTypes>
    <callType>
      <changeStamp>2</changeStamp>
      <refURL>[/unifiedconfig/config/calltype/5001]</refURL>
      <description>Used for Supervisor and Emergency Assist</description>
      <name>Assist</name>
    </callType>
  </callTypes>
</results>
```

Note For sort only commands, the <searchTerm> is not returned.

Allowable Sort Attributes

The following table shows the allowable sort attributes for existing configuration objects.

Configuration Object	Allowable Sort Attributes
agent	• agentId • description • supervisor • agentStateTrace • person.firstName • person.lastName • person.userName • person.loginEnabled
agentDeskSettings	• id

	• name • description • wrapupDataIncomingMode • wrapupDataOutgoingMode • remoteAgentType • logoutNonActivityTime • workModeTimer • supervisorAssistCallMethod • emergencyCallMethod • idleReasonRequired • logoutReasonRequired • autoAnswerEnabled
agentTeam	• id • name (default) • description
attribute	• id • name • dataType • defaultValue • description
bucketIntervals	• id • name (default) • upperBound1-9
bulkJob	• id • description • jobType • jobState • jobHostName • createDateTime

	• startDateTime • endDateTime
callType	• name (default) • description • id • serviceLevelThreshold • serviceLevelType
dialedNumber	• id • dialedNumberString (default) • description
expandedCallVariable	• id • name (default) • description • maximumLength • maximumArraySize • eccArray • enabled • persistent • ciscoProvided
networkVruScript	• id • name (default) • description • vruScriptName • timeout • configParam • interruptible
precision Queue	• name • description

	• id
reasonCode	• id • text (default) • description • code
skillGroup	• id • name • description • serviceLevelThreshold • serviceLevelType • peripheralNumber

Error Results

Specifying an invalid sort field or sort option (**asc|desc**) or too many parameters in the sort request results in an apiError being returned with ErrorType set to **invalidInput.badSortField**.

For example, the sort parameter: **sort=name asc extra** results in the following apiError:

```
<apiErrors>
   <apiError>
     <errorType>invalideInput.badSortField</ErrorType>
     <errorData>name asc extra</ErrorData>
     <errorMessage> .... </ErrorMessage>
  </apiError>
</apiErrors>
```

Asynchronous API

Async calls can offer greater efficiency over synchronous calls and you may want to use async when the system is busy.

Whereas synchronous API calls are blocking calls and do not return until the change has been completed, or there has been an error, with async the response to the API call is returned immediately, with a polling URL, while the request continues to be processed. In heavier load conditions it can be more efficient to submit multiple async calls and periodically check their status than to wait for each call to complete before submitting the next one.

By default, if an API call is made from the UI it has five seconds to get a response from the API before the UI framework times out on that request, even if the request is still being processed. With synchronous calls, in a case where the request takes over five seconds to complete, a request timed-out error is returned to the user, even though the request may have completed successfully. With async, the UI can poll the status of the

request until it is completed and can display an appropriate processing symbol, such as an hour glass, until the request either completes, truly does error out, or reaches a timeout that is deemed too long.

This section explains how to make an Asynchronous (Async) call and outlines the expected responses. It describes the three supported operations for this API: **create**, **update**, and **delete**. The section also provides four use cases, and explains the exceptions returned and the conditions under which they are returned.

The examples shown describe how to use the Async feature to create a Call Type.

create

URL:	https://<server>/unifiedconfig/config/calltype?async=true
HTTP Method:	POST
Response:	If a request is successfully put on the queue for processing—that is, if it has passed the validation check before getting on queue—the result is the HTTP Response **202 Accepted** with the following Location URL in the header, for polling the status of the request: URL: https://<server>/unifiedconfig/config/asyncrequeststatus/<id> And, the following XML content is returned: <asyncResult> <progress>IN_QUEUE</progress> </asyncResult>
Exceptions	See Exceptions, on page 21.

update

URL:	https://<server>/unifiedconfig/config/calltype/<ID>?async=true
HTTP Method:	PUT
Response:	If a request is successfully put on the queue for processing—that is, if it has passed the validation check before getting on queue—the result is the HTTP Response **202 Accepted** with the following Location URL in the header, for polling the status of the request: URL: https://<server>/unifiedconfig/config/asyncrequeststatus/<id> And, the following XML content is returned: <asyncResult> <progress>IN_QUEUE</progress> </asyncResult>
Exceptions	See Exceptions, on page 21.

delete

URL:	https://<server>/unifiedconfig/config/calltype/<ID>?async=true
HTTP Method:	DELETE

Response:	If a request is successfully put on the queue for processing—that is, if it has passed the validation check before getting on queue—the result is the HTTP Response **202 Accepted** with the following Location URL in the header, for polling the status of the request:
	URL: https://<server>/unifiedconfig/config/asyncrequeststatus/<id>
	And, the following XML content is returned:
	`<asyncResult>`
	`  <progress>IN_QUEUE</progress>`
	`</asyncResult>`

Exceptions	See Exceptions, on page 21.

Values

The following table shows the values for <progress> in the returned XML content:

Value	Description
IN_QUEUE	The request passed validation and capacity checks and was put on the queue.
IN_PROGRESS	The request has been taken off the queue and is being processed.

Use Cases

For further explanation, this section provides four use cases.

Create (success)

1) User sends an Asynchronous request to create a Call Type.	Example request: https://<server>/unifiedconfig/config/calltype?async=true
2) System validates the request and puts it in queue and returns the following:	• **202 Accepted** status with progress of IN_QUEUE; and with • A URL in the location header that can be polled for further status information. • ◦ Example URL for polling: https://<server>/unifiedconfig/config/asyncrequeststatus/<id>

3) User polls the status using the provided URL:	https://<server>/unifiedconfig/config/asyncrequeststatus/<id>
4) Depending on timing, as this is a very short state, the system may return the following status to indicate that the request has been taken out of queue and is being processed:	**202 Accepted** status with progress of IN_PROGRESS.
5) User polls the status again using the previously provided URL:	https://<server>/unifiedconfig/config/asyncrequeststatus/<id>
6) System returns the following status to indicate the Call Type was created successfully:	**201 Created** status with a refUrl of https://<server>/unifiedconfig/config/calltype/<id>

Update (success)

1) User sends an Asynchronous request to update an existing Call Type object.	Example request: https://<server>/unifiedconfig/config/calltype/<id>?async=true
2) System validates the request and puts it in queue and returns the following:	• **202 Accepted** status with progress of IN_QUEUE; and with • A URL in the location header that can be polled for further status information. • ◦ Example URL for polling: https://<server>/unifiedconfig/config/asyncrequeststatus/<id>
3) User polls the status using the provided URL:	https://<server>/unifiedconfig/config/asyncrequeststatus/<id>
4) System returns a **200 OK** status to indicate the Call Type was updated successfully.	

Delete (success)

1) User sends an Asynchronous request to delete an existing Call Type object.	Example request: https://<server>/unifiedconfig/config/calltype/<id>?async=true
2) System validates the request and puts it in queue and returns the following:	• **202 Accepted** status with progress of IN_QUEUE; and with • A URL in the location header that can be polled for further status information. • ◦ Example URL for polling: https://<server>/unifiedconfig/config/asyncrequeststatus/<id>

3) User polls the status using the provided URL:	https://<server>/unifiedconfig/config/asyncrequeststatus/<id>
4) System returns a **200 OK** status to indicate the Call Type was deleted successfully.	

Create (failure)

1) User sends an Asynchronous request to create a Call Type.	Example request: https://<server>/unifiedconfig/config/calltype?async=true
2) System validates the request and puts it in queue and returns the following:	• **202 Accepted** status with progress of IN_QUEUE; and with • A URL in the location header that can be polled for further status information. • ◦ Example URL for polling: https://<server>/unifiedconfig/config/asyncrequeststatus/<id>
3) User polls the status using the provided URL:	https://<server>/unifiedconfig/config/asyncrequeststatus/<id>
4) Depending on timing, as this is a very short state, the system may return the following status to indicate that the request has been taken out of queue and is being processed:	**202 Accepted** status with progress of IN_PROGRESS.
5) User polls the status again using the previously provided URL:	https://<server>/unifiedconfig/config/asyncrequeststatus/<id>
6) System returns the following status to indicate that the system capacity has been exceeded for Call Type:	**400 Bad Request**, with error text.

Exceptions

This section explains the exceptions returned and the conditions under which they are returned:

Exceptions:

- Tasks that cannot be put on the queue due to max capacity return an HTTP status code of **503**, with an API error indicating the queue is full.
- If a task reaches its max time in queue of 30 seconds, it is removed from the queue. On the next poll for the status of this task, an HTTP status code of **503** is returned, with an API error indicating timed out.