



This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

# Developer's Guide

Context Services 8.1.x

# Table of Contents

<b>Developer's Guide</b>	<b>3</b>
<b>Business Attributes</b>	<b>4</b>
<b>Anonymous Service</b>	<b>6</b>
<b>Server Mode</b>	<b>7</b>
<b>Basic Access Authentication</b>	<b>8</b>
<b>Role-Based Access Control</b>	<b>11</b>
<b>Extensions</b>	<b>18</b>
<b>Profiles and Identification</b>	<b>25</b>
<b>Grouping Customer Profiles</b>	<b>30</b>
<b>Services, States, and Tasks</b>	<b>34</b>

# Developer's Guide



**Purpose:** These developer pages, primarily intended for programmers developing strategies for contact center agents, assume that you have a basic understanding of:

- Computer-telephony integration (CTI): concepts, processes, terminology, and applications.
- Network design and operation.
- Your own network configurations.

## Introduction

This developer's guide covers the writing and the optimization of your applications on top of the Context Services. Representations, requests, and responses are detailed in the [API Reference](#) page. Use this developer guide to learn about the operations and representations used in this REST API. Developer pages are intended to help you to:

- Understand the design of the Context Services
- Get details to optimize your application's architecture
- Give general directions to your implementation on top of this product.

💡 If pages are missing information or not helpful enough, use the comment form at the bottom of the page to submit questions and feedbacks.

# Business Attributes



**Purpose:** Introduces Business Attributes in Context Services.

## Definition

Management Framework creates and manages enumerations known as Business Attributes. These attributes are modeled in Context Services as integers which represent the Management Framework DB ID for a given enumerated value. For example, an organization might define the "service type" Business Attribute, made of two enumerated values:

- "New Account"(DB ID = 1);
- "Bill Payment" (DB ID = 2).

The applications calling Context Services are responsible for the further use of those values and should only use the appropriate business attribute values.

## Business Attributes in Context Services

The following specific fields are validated against specified mapped Business Attributes in the Configuration Server:

- service\_type
- state\_type
- task\_type
- application\_type
- resource\_type
- media\_type
- disposition

Their use concerns the Service, State, and Task representations. UCS automatically rejects wrong unknown enumerated values, and returns a proper response which directs you to use the valid enumerated values of the configured Business Attributes. The system includes a service for returning information on attributes mapped to the Configuration Server attributes, including information on the DB ID, unique name, display name, and description for all values of the mapped Business Attributes.

- UCS only validates incoming data against the current Business Attribute definitions. It does not guarantee referential integrity over time. More specifically, both Genesys Administrator and Configuration Manager allow the modification of the Business Attributes definitions over time.

## Business Attributes

---

- When Business Attributes are deleted, this operation does not modify the historical service records which may reference the DB IDs of the deleted Business Attributes.

For further details about the configuration of these business attributes, refer to the [Business Attributes configuration section](#).

## Read Also

- [Profiles and Identification](#)
- [Services, States, and Tasks](#)

# Anonymous Service



**Purpose:** Introduces the Anonymous service and its management.

## Definition

An **anonymous** service is a service which is assigned to an anonymous customer. This customer is still unknown, so no customer ID is assigned to the service. Your application is in charge of assigning this customer ID as soon as the customer is identified. Read also [Services, States, and Tasks](#).

## Use Case

In many situations, your application can identify the customer prior to the creation of the service, which ensures the possibility of adding the customer ID to the service in the [Start Service](#) operation. Two examples: the customer explicitly logs in the website before invoking the service, or the IVR identifies the customer and then chooses a service. In those cases, your application cannot specify the customer ID at the service creation. However, in other cases, your application may start the service before the customer is identified. Therefore, if your application cannot specify the customer's ID at the service creation, the service is **anonymous**. Let's consider a customer who is filling out an order on a web site before he or she has explicitly logged in, or a preliminary service delivered in the IVR before the customer is prompted for identity information. In these cases, the application is not able to provide the customer identifier.

## The Contact Key

Your application can create an anonymous service with the [Start Service](#) operation. In that case, although the customer is not identified, your application must pass a contact key, based on the current information available. The "contact key" is supplied at the service creation. Then, your application is able to query the service even if it is anonymous. See [Query Anonymous Services](#). Examples of contact key can be the following: e-mail address, phone number, lastname+firstname.

## Related Operations

- [Associate Service](#)
- [Start Service](#)
- [Query Anonymous Services](#)

# Server Mode



**Purpose:** Describes the two UCS server modes, maintenance and production, available for Context Services.

In versions 8.1.000.10 and higher, you need to check the privileges set according to roles prior to using the operations described on this page. See [Role-Based Access Control](#) for additional details.

## Introduction

Universal Contact Server provides Context Services with two modes: production and maintenance. The current mode can be changed by using the [Set Server Mode](#) operation. If your application attempts to access a method unavailable in the given mode, it receives the HTTP error code 503 ("Not available"). The body of that HTTP response includes the error message which confirms that the selected method is unavailable while the system is in "production" or "maintenance" mode, respectively. See [W3 RFC2616](#).

## Maintenance Mode


In this mode, UCS authorizes only the [Schema Operations](#), and does not accept requests for managing customer profiles or service-related data. If no profile schema is defined, UCS returns the HTTP Status Code 404 (Not Found) and automatically switches to maintenance mode.

## Production Mode

In production mode, UCS accepts incoming requests for managing the customer profiles and service-related data. Your application should not update or modify attribute schemas, so [Schema Operations](#) are unavailable unless they are explicitly stated.

If the production mode is required to use a given Context Services method, then the related wiki page for that method includes a note in the prerequisites of the operation.

# Basic Access Authentication

	<b>Purpose:</b> Offers guidelines for managing Authentication with the Context Services.
	<b>Available since:</b> 8.0.300.02

## About Basic Authentication

[Wikipedia Basic Access Authentication](#) states that: *In the context of an HTTP transaction, the basic access authentication is a method designed to allow a web browser, or other client program, to provide credentials – in the form of a user name and password – when making a request.* The Context Services provide support for basic access authentication once enabled in the [authentication section](#) of your configuration.

- When basic access authentication is enabled, the REST requests must contain a valid username and password in the HTTP/HTTPS header. As a result, the Context Services send descriptive error messages if they receive an incorrect username/password combination.
- If basic authentication is disabled, the Context Services ignore any username or password passed in HTTP/HTTPS header.

If the authentication is enabled and valid information is not provided, the Context Services return the HTTP response **401 Unauthorized**. In that case, the user application should resubmit the request with the proper authentication header.

## Base64 Encoding

The authentication string to transmit is the result of the concatenation of the username and password separated by a colon (*username:password*). It must then be encoded with the Base64 algorithm. For example, if the username is 'kent' and the password 'superman', the string to encode is kent:superman and results in the string 'a2VudDpzdXB1cm1hbg=='. If you are using a framework, it may provide the Base64-encoding transparently. If your framework does not include the Base64-encoding feature then you must encode your string. The following code snippet shows how to proceed with a Restlet application:

```
final Request request = new Request();
String url = "http://" + host + ":" + port + "/server/status";
request.setResourceRef(url);
request.setMethod(Method.GET);
final Client myClient = new Client(Protocol.HTTP);
ChallengeResponse authentication = new ChallengeResponse(ChallengeScheme.HTTP_BASIC, "kent",
"superman");
request.setChallengeResponse(credential);
Response response = client.handle(request);
```

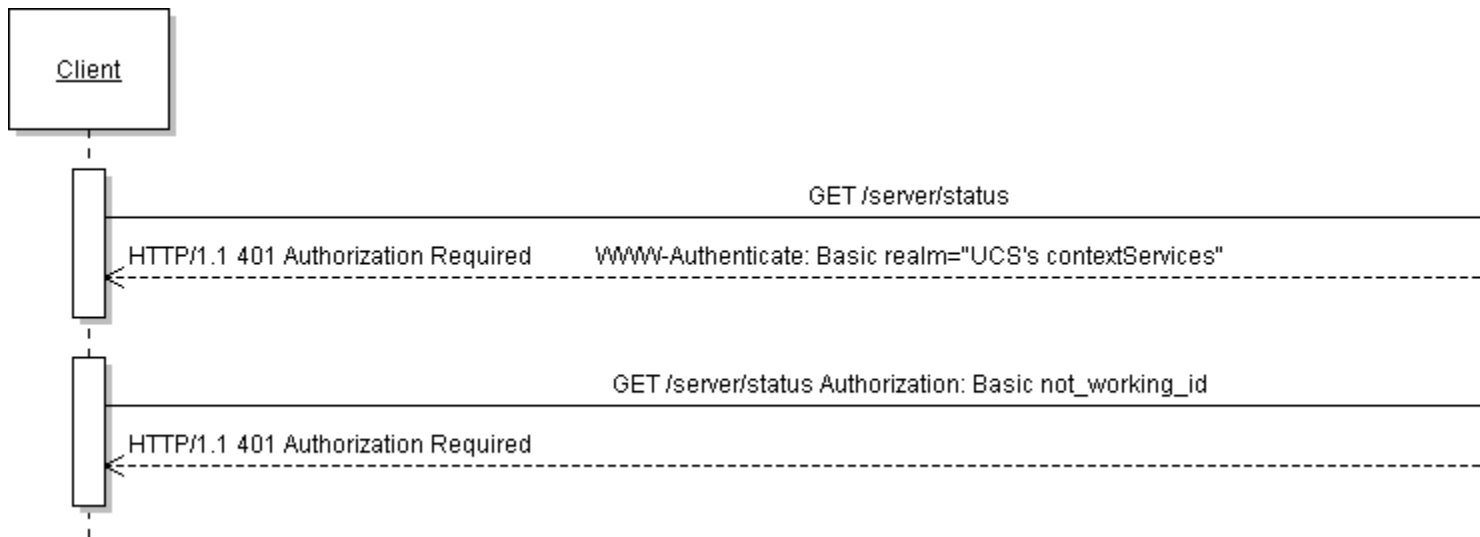
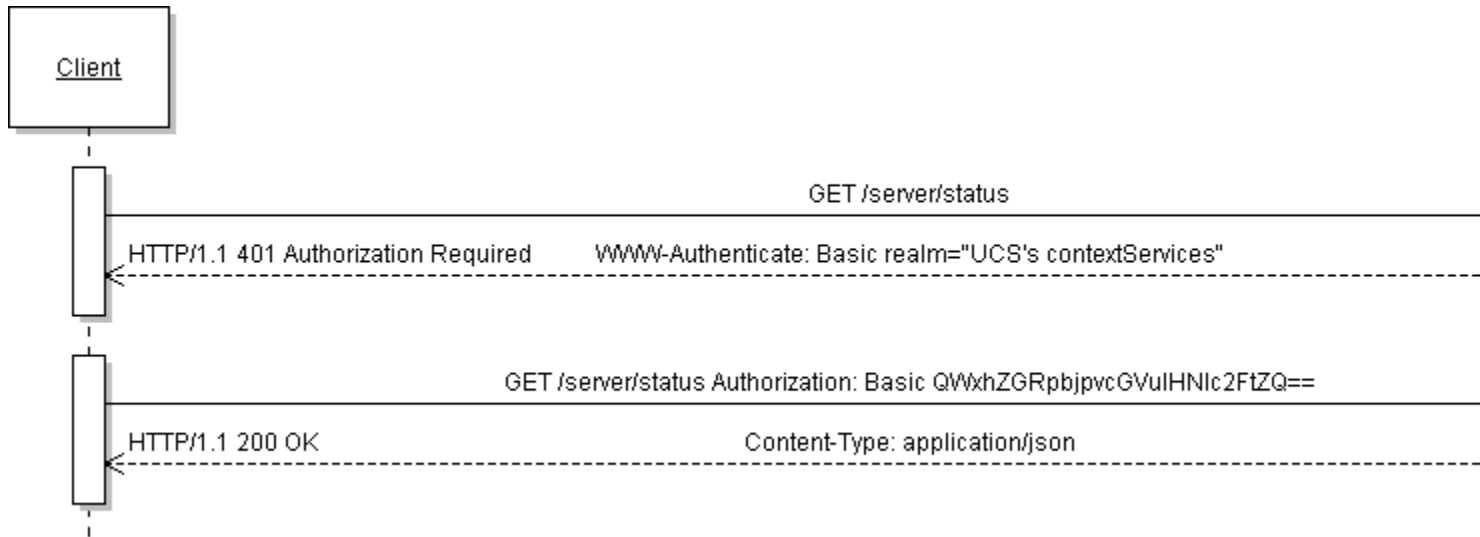
Additional examples of Base64 encoding are available in [Wikipedia Basic Access](#)



## Authentication.

### Request Flow and Returned Errors

The following sequence diagrams show the protocol request and answer flow when basic access authentication is enabled.



If the request returns the **401 Unauthorized** error, your application should retry with a correct HTTP header. The Context Services returns **401 Unauthorized** error due to authentication issues in the following scenarios:

- The authentication is enabled and the request is not authorized

- The request provides the correct header for authentication, but wrong credential information (the username or the password is wrong).

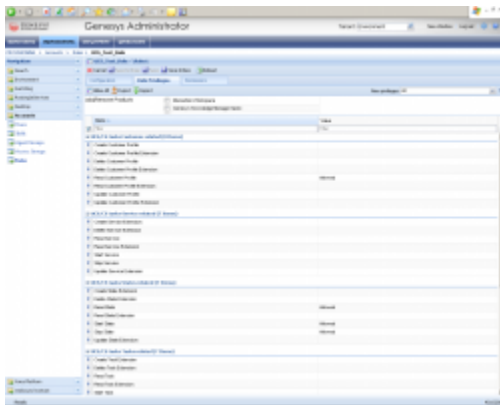
# Role-Based Access Control



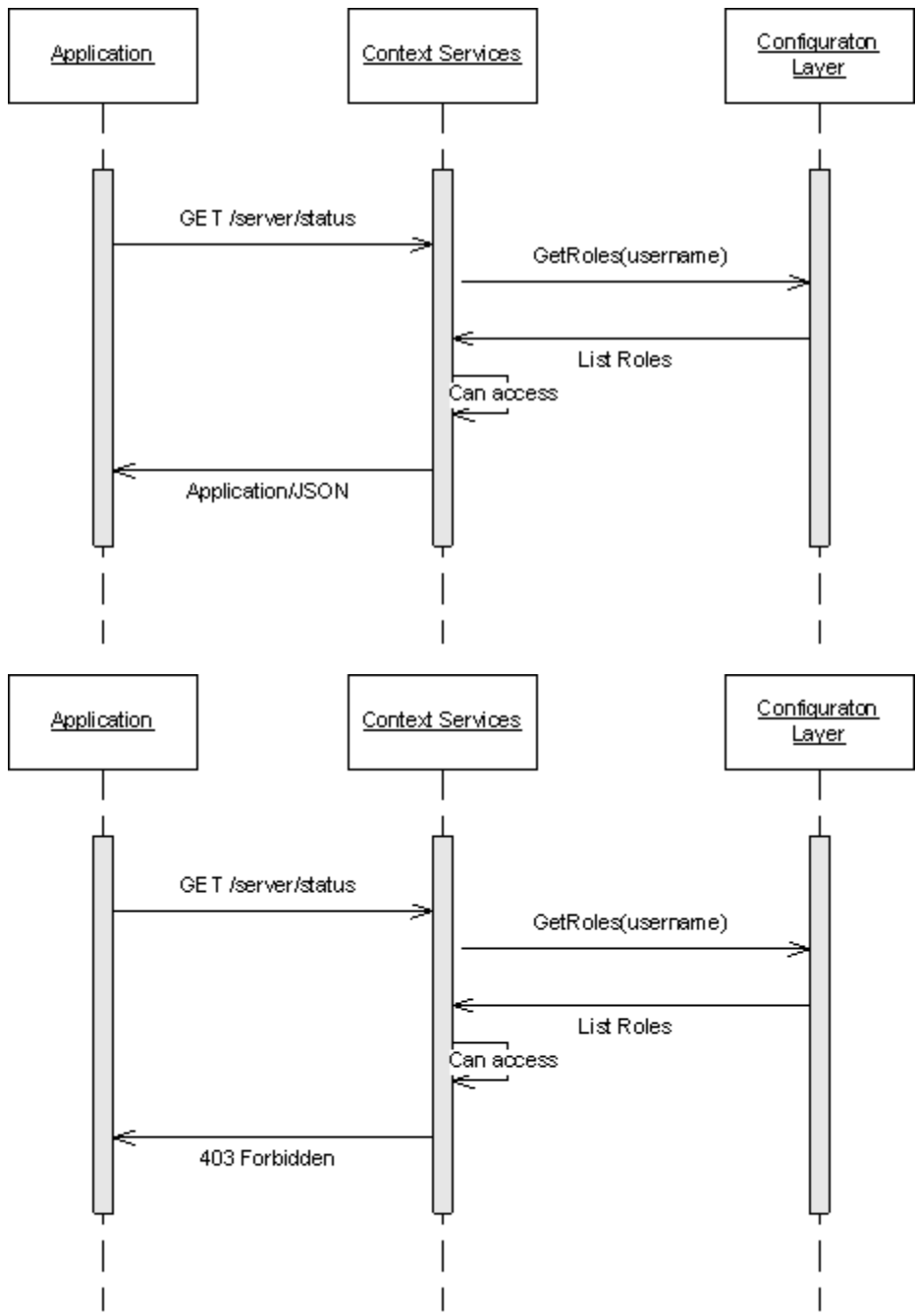
**Purpose:** Understand how to deal with the role-based access implemented in the Context Services.  
**Available since:** 8.1.000.10

## Role-Based Access Configuration

Through Configuration Manager or Genesys Administrator, you can define roles for your application built on top of the Context Services. To do this, you assign one or more roles to your users when creating your application's configuration in the Context Services. You are responsible for creating and defining these roles, where each role is a collection of Genesys Administrator Tasks associated with permissions.



Once authenticated, if the use-role option is set to true in the configuration (see the options defined in [authentication Section](#)) then the Universal Contact Server checks that each operation is allowed. If not, Error 403 forbidden is returned.



## Mapping Genesys Administrator Task with Context Services

Operations can require that one or more Genesys Administrator Tasks are allowed, depending on what data is modified by the request. If your application's role does not allow all of the rights required for an operation then that operation will not proceed. For example, consider that your application performs a **Start Service** operation with extensions. If your application's role allows

UCS.Service.startService but not UCS.Service.createServiceExtension then the service is neither created nor started. Your application instead receives a HTTP 403 Forbidden error.

Operation	Genesys Administrator Tasks
<b>Profile Operations</b>	
<b>Create Customer Profile</b> POST /profiles	<ul style="list-style-type: none"> <li>UCS.Customer.createProfile</li> <li>UCS.Customer.createProfileExtension (if extensions)</li> </ul>
<b>Delete Customer Profile</b> DELETE /profiles/{customer_id}	<ul style="list-style-type: none"> <li>UCS.Customer.deleteCustomerProfile</li> </ul>
<b>Delete Record From Profile Extension</b> PUT /profiles/{customer_id}/extensions/{ext_name}/by/unique	<ul style="list-style-type: none"> <li>UCS.Customer.deleteProfileExtension</li> </ul>
<b>Identify Customer</b> GET /profiles	<ul style="list-style-type: none"> <li>UCS.Customer.readCustomerProfile</li> <li>UCS.Customer.readProfileExtension (if include_extensions is specified in the query)</li> </ul>
<b>Insert Extension Records</b> POST /profiles/{customer_id}/extensions	<ul style="list-style-type: none"> <li>UCS.Customer.createProfileExtension</li> </ul>
<b>Bulk Profile Import</b> POST /profiles/import	<ul style="list-style-type: none"> <li>UCS.Customer.executeBulkImport</li> <li>UCS.Customer.createProfile</li> <li>UCS.Customer.createProfileExtension</li> </ul>
<b>Query Customer Profile</b> GET /profiles/{customer_id}	<ul style="list-style-type: none"> <li>UCS.Customer.readCustomerProfile</li> <li>UCS.Customer.readProfileExtension (if extensions)</li> </ul>
<b>Update Customer Profile</b> PUT /profiles/{customer_id}	<ul style="list-style-type: none"> <li>UCS.Customer.updateCustomerProfile</li> <li>UCS.Customer.updateProfileExtension (if extensions)</li> </ul>
<b>Merge Customer Profile</b> PUT /profiles/{customer_id}/merge/{src_id}/	<ul style="list-style-type: none"> <li>UCS.Customer.mergeCustomerProfile</li> </ul>
<b>Update Record In Profile Extension</b> PUT /profiles/{customer_id}/extensions/{ext_name}/by/unique	<ul style="list-style-type: none"> <li>UCS.Customer.updateProfileExtension</li> </ul>
<b>Service Operations</b>	

Operation	Genesys Administrator Tasks
<b>Associate Service</b> POST /customers/{customer_id}/services/{service_id}	<ul style="list-style-type: none"> <li>UCS.Service.startService</li> <li>UCS.Service.updateServiceExtension (if extension)</li> </ul>
<b>Complete Service</b> POST /services/{service_id}/end	<ul style="list-style-type: none"> <li>UCS.Service.stopService</li> <li>UCS.Service.updateServiceExtension</li> </ul>
<b>Delete Record From Service Extension</b> PUT /services/{service_id}/extensions/{ext_name}/delete/by/unique	<ul style="list-style-type: none"> <li>UCS.Service.deleteServiceExtension</li> </ul>
<ul style="list-style-type: none"> <li><b>Query Services</b></li> </ul> GET /services/anonymous/{contact_key} GET /customers/{customer_id}/services  <ul style="list-style-type: none"> <li><b>Query Service by ID</b></li> </ul> GET /services/{service_id}	<ul style="list-style-type: none"> <li>UCS.Service.readService</li> <li>UCS.Service.readServiceExtension (if extensions)</li> <li>UCS.States.readState</li> <li>UCS.Tasks.readTask</li> </ul>
<b>Start Service</b> POST /services/start	<ul style="list-style-type: none"> <li>UCS.Service.startService</li> <li>UCS.Service.createServiceExtension</li> </ul>
<b>Update Service Extension</b> PUT /services/{service_id}/extensions/{ext_name}	<ul style="list-style-type: none"> <li>UCS.Service.updateServiceExtension</li> </ul>
<b>Update Record In Service Extension</b> PUT /services/{service_id}/extensions/{ext_name}/by/unique	<ul style="list-style-type: none"> <li>UCS.Service.updateServiceExtension</li> </ul>
<b>State Operations</b>	
<b>Complete State</b> POST /services/{service_id}/states/{state_id}/end	<ul style="list-style-type: none"> <li>UCS.States.stopState</li> <li>UCS.States.updateStateExtension</li> </ul>
<b>Delete Record From State Extension</b> PUT /services/{service_id}/states/{state_id}/extensions/{ext_name}/delete/by/unique	<ul style="list-style-type: none"> <li>UCS.States.deleteStateExtension</li> </ul>
<b>Perform State Transition</b> POST /services/{service_id}/states/transition	<ul style="list-style-type: none"> <li>UCS.States.startState</li> <li>UCS.States.stopState</li> <li>UCS.States.createStateExtension (if extensions)</li> </ul>

Operation	Genesys Administrator Tasks
	<ul style="list-style-type: none"> <li>UCS.States.updateStateExtension (if extensions)</li> </ul>
<ul style="list-style-type: none"> <li>Query States</li> </ul> GET /services/{service_id}/states  <ul style="list-style-type: none"> <li>Query State by ID</li> </ul> GET /services/{service_id}/states/{state_id}	<ul style="list-style-type: none"> <li>UCS.States.readState</li> <li>UCS.States.readStateExtension (if extensions)</li> <li>UCS.Tasks.readTask</li> </ul>
<b>Start State</b> POST /services/{service_id}/states/start	<ul style="list-style-type: none"> <li>UCS.States.startState</li> <li>UCS.States.createStateExtension</li> </ul>
<b>Update State Extension</b> PUT /services/{service_id}/states/{state_id}/extensions/{ext_name}	<ul style="list-style-type: none"> <li>UCS.States.updateStateExtension</li> </ul>
<b>Update Record In State Extension</b> PUT /services/{service_id}/state/{state_id}extensions/{ext_name}/by/unique	<ul style="list-style-type: none"> <li>UCS.States.updateStateExtension</li> </ul>
<b>Task Operations</b>	
<b>Complete Task</b> POST /services/{service_id}/tasks/{task_id}/end	<ul style="list-style-type: none"> <li>UCS.Tasks.stopTask</li> <li>UCS.Tasks.updateTaskExtension</li> </ul>
<b>Delete Record From Task Extension</b> PUT /services/{service_id}/task/{task_id}/extensions/{ext_name}/delete/by/unique	<ul style="list-style-type: none"> <li>UCS.Tasks.deleteTaskExtension</li> </ul>
<ul style="list-style-type: none"> <li>Query Tasks</li> </ul> GET /services/{service_id}/tasks  <ul style="list-style-type: none"> <li>Query Task by ID</li> </ul> GET /services/{service_id}/tasks/{task_id}	<ul style="list-style-type: none"> <li>UCS.Tasks.readTask</li> <li>UCS.Tasks.readTaskExtension</li> </ul>
<b>Start Task</b> POST /services/{service_id}/tasks/start	<ul style="list-style-type: none"> <li>UCS.Tasks.startTask</li> <li>UCS.Tasks.createTaskExtension</li> </ul>
<b>Update Task Extension</b> PUT /services/{service_id}/tasks/{task_id}/extensions/{extension_name}	<ul style="list-style-type: none"> <li>UCS.Tasks.updateTaskExtension</li> </ul>
<b>Update Record In Task Extension</b> PUT /services/{service_id}/task/{task_id}extensions/{ext_name}/by/unique	<ul style="list-style-type: none"> <li>UCS.Tasks.updateTaskExtension</li> </ul>

Operation	Genesys Administrator Tasks
<b>Schema Operations</b>	
<b>Create Profile Extension Schema</b> POST /metadata/profiles/extensions	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.createProfileExtensionSchema</li> </ul>
<b>Create Identification Key</b> POST /metadata/identification-keys	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.createIdKeys</li> </ul>
<b>Create State Extension Schema</b> POST /metadata/states/extensions	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.createStateExtensionSchema</li> </ul>
<b>Create Task Extension Schema</b> POST /metadata/tasks/extensions	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.createTaskExtensionSchema</li> </ul>
<b>Create Service Extension Schema</b> POST /metadata/services/extensions	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.createServiceExtensionSchema</li> </ul>
<b>Get Identification Keys</b> GET /metadata/identification-keys	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.readIdKeys</li> </ul>
<b>Query Profile Schema</b> GET /metadata/profiles/	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.readProfileExtensionSchema</li> </ul>
<b>Query Profile Extension Schema</b> GET /metadata/profiles/extensions	UCS.SchemaMgt.readProfileExtensionSchema
<b>Query State Extension Schema</b> GET /metadata/states/extensions	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.readStateExtensionSchema</li> </ul>
<b>Query Task Extension Schema</b> GET /metadata/tasks/extensions	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.readTaskExtensionSchema</li> </ul>
<b>Query Service Extension Schema</b> GET /metadata/services/extensions	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.readServiceExtensionSchema</li> </ul>
<b>Query Business Attribute Schema</b> GET /metadata/business-attributes/{business-attribute-name}	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.readBusinessAttributes</li> </ul>
<b>Get Metadata Cache</b> GET /metadata/cache	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.handleMetadata</li> </ul>
<b>Change Metadata Cache</b> PUT /metadata/cache	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.handleMetadata</li> </ul>
<b>Get Metadata</b> GET \${contentType} /metadata	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.handleMetadata</li> </ul>
<b>Delete Metadata Profile Extensions</b> DELETE /metadata/profiles/extensions/{extension-name}	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.deleteProfileExtensionSchema</li> </ul>



Operation	Genesys Administrator Tasks
<b>Delete Metadata Services Extensions</b> DELETE /metadata/services/ extensions/\${extension-name}	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.deleteServiceExtensionSchema</li> </ul>
<b>Delete Metadata States Extensions</b> DELETE /metadata/states/extensions/\${extension- name}	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.deleteStateExtensionSchema</li> </ul>
<b>Delete Metadata Tasks Extensions</b> DELETE /metadata/tasks/extensions/\${extension- name}	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.deleteTaskExtensionSchema</li> </ul>
<b>Delete Metadata Identification Keys</b> DELETE /metadata/identification-keys/\${id_key- name}	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.deleteIdKeys</li> </ul>
<b>Interaction Operations</b>	
<b>Query Interactions</b> GET /customers/\${customer_id}/interactions GET /services/\${service_id}/interactions GET /interactions/\${interaction_id}	<ul style="list-style-type: none"> <li>UCS.SchemaMgt.readInteraction</li> </ul>
<b>Server Operations</b>	
<b>Query Server Status</b> GET /server/status	<ul style="list-style-type: none"> <li>UCS.SystemMgt.readServerInfo</li> </ul>
<b>Set Server Mode</b> POST /server/mode	<ul style="list-style-type: none"> <li>UCS.SystemMgt.changeServerMode</li> </ul>

# Extensions



**Purpose:** Offers guidelines for managing Extensions provided by the Context Services.

In versions 8.1.000.10 and higher, you need to check the privileges set according to roles prior to using the operations described on this page. See [Role-Based Access Control](#) for additional details.

## About Extensions

Extensions are additional information which extend the standard contents of resources such as [Customer Profile](#), [Service](#), [State](#), and [Task](#). An [Extension](#) is a record-a list of attributes-or an array of records, associated with a resource ID.

- You can define as many extension types as you need by creating an [Extension Schema](#) for each of them.
- Extension schema are created through Context Services (see [List of Schema Operations](#)), not through the Configuration Layer (Configuration Manager).

Extension records can be either:

- "single-valued": The extension contains a single record across the resource (for instance, LastName, FirstName, identifiers, etc.)
- "multi-valued": The extension can contain several values (for instance, phone numbers, e-mail addresses, etc.)

Extensions are provided at the same time and at the same level than the attributes of the resource. For instance, the following output presents a profile containing the attributes FirstName, LastName, DOB<ref>DOB: Date Of Birth</ref> and one multi-valued extension *EmailAddress*:

```
{
  "FirstName": "Bruce",
  "LastName": "Banner",
  "DOB": "1962-05-10",
  "EmailAddress": [
    "bruce.banner@marvelous.com",
    "b.banner@hulk.dom"
  ]
}
```

## Unique Attributes


In the case of multi-valued extensions, the attributes which are part of the 'unique' list (specified in the [Extension Schema](#)) are used to identify records. The combination of these attributes' values must be unique across the related resource, and this enables UCS to identify a given record in the given extension. For example, consider a 'Bill' extension which includes the attribute *bill\_id*. To ensure that

a given service does not have two 'Bill' extensions with the same *bill\_id*, set the following unique array in the extension schema:

```
unique = ["bill_id"]
```

The attributes of the unique list are mandatory at the extension record's creation. You need to provide values for the 'unique' attributes:

- At the creation of an extension record.
- In operations which update or delete a specific record, such as [Update Record In Profile Extension](#) or [Delete Record From Profile Extension](#).

 Operations which manage extension records are part of the related resource operations. For instance, the operations which manage records of profile extensions are part of the [List of Profile Operations](#).

## Limitations

- Once created, you cannot update the schema.
- When you are dealing with extensions or extension schema, make sure that you do not use one of the

[Unauthorized Strings](#) as an attribute name or value.

## Managing Extension Schema

Operations and resources in this section are part of

Before you can start using extensions, you must create their schema.

 Once created, you cannot update or remove them.

You can create schema with the following operations:

- [Create Profile Extension Schema](#)
- [Create State Extension Schema](#)
- [Create Task Extension Schema](#)
- [Create Service Extension Schema](#)

Then, you can retrieve extension schema.

- [Query Profile Schema](#)
- [Query Profile Extension Schema](#)
- [Query State Extension Schema](#)
- [Query Task Extension Schema](#)
- [Query Service Extension Schema](#)

**Example: Retrieving the schema for profile extensions:**

GET /metadata/profiles/extensions

**Result**

```

200 OK
[
{
  "name": "Phone",
  "type": "multi-valued",
  "attributes": [
    {"name": "PhoneType", "type": "integer", "default": 0, "mandatory": "true"},
    {"name": "prefix", "type": "string", "length": "3", "default": "555", },
    {"name": "PhoneNumber", "type": "integer", "length": 15, "mandatory": "true"},
    {"name": "description", "type": "string", "length": 32, "mandatory": "true"},
    {"name": "start_availability", "type": "datetime"},
    {"name": "end_availability", "type": "datetime", "mandatory": "false"}
  ]
},
{
  "name": "Address",
  "type": "single-valued",
  "attributes": [
    {"name": "AddressType", "type": "integer", "default": 0},
    {"name": "Address", "type": "string", "length": 256},
    {"name": "City", "type": "string", "length": 32},
    {"name": "County", "type": "string", "length": 32},
    {"name": "PostCode", "type": "string", "length": 10},
    {"name": "Country", "type": "string", "length": 32}
  ]
} ]

```

## Managing Extensions

### Adding Extensions to a given Resource

You can add extensions when managing the resources with related operations which authorize the <extension n> attribute in the operation's body (the following list may not be exhaustive): [Associate Service](#), [Start Service](#), [Complete Service](#), [Start State](#), [Complete State](#), [Perform State Transition](#), [Start Task](#), [Complete Task](#).

⚠ In that case, if a former value of the extension exists for the given resource, this former extension value is replaced with the new extension value specified in the body.

Let's consider the following multi-valued extension record named 'Satisfaction'. The unique field which identifies records is "place" (the name of the proposed place for the booking. **Example: Records for a 'Satisfaction' extension**

```

[
{
  "rating": 2,
  "pertinence": 8,
  "usefull": true,
  "place": "Terranova mexico resort"
},

```

## Extensions

---

```
{
  "rating":8,
  "pertinence":4,
  "usefull":false,
  "place":"Fancy resort Paris"
}
```

The following operation **Complete State** indicates a single value for the 'Satisfaction' extension.  
**Example: Operation which updates the 'Satisfaction' extension for a given state.**

```
POST /services/6739/states/5362/end
{
  "interaction_id":"00001a57JGQ00BVS",
  "disposition": 10,
  "disposition_desc": "SUCCESS",
  "application_type":"customer_online_survey",
  "application_id":40,
  "resource_type":"html",
  "resource_id":20,
  "media_type":"webform",
  "Feedback":
  {
    "FeedbackType":"survey",
    "rating":7,
    "notes":"warm welcome at frontdesk, thanks for the nice trip"
  },
  "Satisfaction": [
    {
      "rating":2,
      "pertinence":6,
      "usefull":true,
      "place":"Marina Porto Vecchio"
    }
  ]
}
```

As a result, the previous records 'Fancy resort Paris ' and 'Terranova mexico resort ' are lost. In this case, to add a new record to the extension, you must specify the whole extension content. For instance, note the following: **Example: Operation which updates the 'Satisfaction' extension without losing records**

```
POST /services/6739/states/5362/end
{
  "interaction_id":"00001a57JGQ00BVS",
  "disposition": 10,
  "disposition_desc": "SUCCESS",
  "application_type":"customer_online_survey",
  "application_id":40,
  "resource_type":"html",
  "resource_id":20,
  "media_type":"webform",
  "Feedback":
  {
    "FeedbackType":"survey",
    "rating":7,
    "notes":"warm welcome at frontdesk, thanks for the nice trip"
  },
  "Satisfaction": [
    {
      "rating":2,
      "pertinence":6,

```

## Extensions

---

```
    "usefull":true,
    "place":"Marina Porto Vecchio"
  },
  {
    "rating":2,
    "pertinence":8,
    "usefull":true,
    "place":"Terranova mexico resort"
  },
  {
    "rating":8,
    "pertinence":4,
    "usefull":false,
    "place":"Fancy resort Paris"
  }
}]
}
```

## Retrieving Extensions

GET operations which enable to retrieve resources include the "extensions" parameter to specify a list of extensions to retrieve. By default, extensions are not returned. The following list is not exhaustive:

- [Query Customer Profile](#)
- [Query Services](#)
- [Query Service by ID](#)
- [Query States](#)
- [Query State by ID](#)
- [Query Tasks](#)
- [Query Task by ID](#)

## Retrieving service with the extensions ClientInfo,relatedOffers

```
GET /services/3005?extensions=ClientInfo,relatedOffers
```

Result:

```
{
  "service_id" : 3005,
  "ClientInfo" : {
    "userAgent" : "Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.9.2) Gecko/20100115
Firefox/3.6 (.NET CLR 3.5.30729)",
    "clientId" : "192.168.1.1",
    "contentType" : "Content-Type : application/json;charset=UTF-8"
  },
  "service_type" : 100,
  "est_duration" : 300,
  "started" : {
    "timestamp" : "2010-09-07T07:58:16.313Z",
    "application_type" : 400,
    "resource_id" : 10,
    "media_type" : 2,
    "resource_type" : 200,
    "application_id" : 40,
    "interaction_id" : "56"
  }
}
```

```
  },
  "disposition" : 5,
  "completed" : {
    "timestamp" : "2010-06-03T08:51:54.380Z",
    "interaction_id" : "1587"
  }
} ],
"relatedOffers" : [ {
  "offer_name" : "VIP credit card black ed.",
  "type" : "9",
  "comments" : "proposed to all client"
}, {
  "offer_name" : "3 times payment GOLD",
  "type" : "4",
  "comments" : "limited offer"
}, {
  "offer_name" : "life insurance",
  "type" : "3",
  "comments" : "healt check to be done before approval"
} ],
"contact_key" : "bob"
}
```

## Managing Extension Records

The Context Services Restful API includes dedicated operations such as update or delete a record, or update the whole extension. They are available in the [List of Service Operations](#), [State Operations](#), and [Task Operations](#) pages. When dealing with these operations, you must provide [unique attributes](#) to identify the targeted record. For instance, to manage State Extensions, the [State Operations](#) provide the following operations:

- [Update State Extension](#)
- [Update Record In State Extension](#)
- [Delete Record From State Extension](#)

### **Example: Updating the *Tons of Hill, Paris* record of the extension *relatedOffers* in the service *8389*.**

```
PUT /services/8389/states/1/extensions/Satisfaction/by/unique
{
  "rating":2,
  "pertinence":8,
  "usefull":true,
  "place":"Tons of Hill, Paris"
}
```

## Deleting an Extension

To delete the extension of a given resource, use the related *Update XXX Extension* operation with no attributes in the operation's body.

- [Update Customer Profile](#) and [Update Record In Profile Extension](#)
- [Update Service Extension](#)
- [Update State Extension](#)

- [Update Task Extension](#)

**Example: Deleting the *relatedOffers* multi-valued extension of the service 8389**

```
PUT /services/8389/extensions/relatedOffers  
[]
```

In versions 8.1.000.10 and higher, as explained in [Role-Based Access Control](#), you need update privileges to clear the extension, as follows:

- Clear Profile Extension—UCS.Customer.updateProfileExtension
- Clear Service Extension—UCS.Service.updateServiceExtension
- Clear State Extension—UCS.States.updateStateExtension
- Clear Task Extension—UCS.Tasks.updateTaskExtension

## Read More

- [Profiles and Identification](#)
- [Services, States, and Tasks](#)



---

# Profiles and Identification



**Purpose:** Gives guidelines for managing Customer Profiles with Context Services.

In versions 8.1.000.10 and higher, you need to check the privileges set according to roles prior to using the operations described on this page. See [Role-Based Access Control](#) for additional details.

## Learn about the Customer Profile and Associated Resources

The [Customer Profile](#) resource associates a customer ID with:

- A list of attributes, built on top of the existing UCS<ref>UCS: Universal Contact Server</ref> Contact model.
- A list of extensions, defined at runtime through Context Services

As stated in [Profile Basics](#), the attributes correspond to Core information in UCS. Their schema fulfills defined through the list of Business Contact Attributes that you can define in Configuration Manager (see [Configuration Options](#) for additional details). The extensions are additional information that your application can create so that you can extend the profile at runtime, as explained in [Extending the Customer Profile](#). Identification Keys define which attributes should be used to identify a customer in the database. For instance, the association of [LastName, Firstname], or the e-mail address.

### Attribute Values

Attributes and extension records can be either:

- "single-valued"(for instance, LastName, FirstName, identifiers, and so on);
- "multi-valued": values can be multiple (for instance, phone numbers, e-mail addresses, and so on).

The following output presents a sample of Customer profile, where "FirstName", "LastName", and "DOB" (Date Of Birth) are single-valued contact attributes, and the other fields are multi-valued extension records created at runtime.

```
{
  "FirstName": "Bruce",
  "LastName": "Banner",
  "DOB": "1962-05-10",
  "EmailAddress": [
    "bruce.banner@marvelous.com",
    "b.banner@hulk.dom"
  ],
  "Phone": [
    {
      "PhoneType": 0,
      "prefix": "+33",
    }
  ]
}
```

```
    "PhoneNumber": "3145926535",
    "description": "family phone",
    "start_availability": "2009-12-18T18:30:00.000Z",
    "end_availability": "2009-12-18T21:40:00.000Z"
  },
  {
    "PhoneType": 2,
    "prefix": "+33",
    "PhoneNumber": "6543210",
    "description": "business calls only, no sales",
    "start_availability": "2009-12-18T09:30:00.000Z",
    "end_availability": "2009-12-18T17:45:00.000Z"
  },
  {
    "PhoneType": 5,
    "prefix": "+33",
    "PhoneNumber": "951357456",
    "description": ""
  }
]
}
```

## Profile Content

The content of the Customer Profile follows a schema (a translation of the Business Contact Attributes), which describes its content with a list of [Attribute Schema](#), apart from extension content, as shown in the following output example:

```
{"encrypt": false, "name": "PIN", "length": 256, "type": "string"},
{"encrypt": false, "name": "Title", "length": 256, "type": "string"},
{"encrypt": false, "name": "CustomerSegment", "length": 256, "type": "string"},
{"encrypt": false, "name": "LastName", "length": 256, "type": "string"},
{"encrypt": false, "name": "AccountNumber", "length": 256, "type": "string"},
{"encrypt": false, "name": "FirstName", "length": 256, "type": "string"},
{"encrypt": false, "name": "PhoneNumber", "length": 256, "type": "string"},
{"encrypt": false, "name": "ContactId", "length": 256, "type": "string"},
{"encrypt": false, "name": "EmailAddress", "length": 256, "type": "string"},
```

The profile schema does not contain information related to extensions.

At runtime, your application can retrieve this schema through the [Query Profile Schema](#) operation. Your application cannot modify the profile schema through Context Services:

- If you wish to modify the profile schema, make modifications to in Configuration Manager via the Business Attribute "ContactAttributes".
- If you wish to add additional information to the profile at runtime, use the profile extensions.

## Extensions

Extensions allow you to extend the content of a profile with additional records. Your application is fully responsible for the creation and management of this optional content. As stated in the [Preliminary Steps to Using Profiles](#) section, your application must define the extensions' content by creating the extension. Once the schema is defined, your application can use the extensions by specifying extension information to the profile submitted with the operations' body, or by using dedicated operations, as presented below.

## Identification Key

The identification key is a combination of attributes used to identify a customer. These attributes (one or more) belong to the customer profile and/or its extensions.

 Identification keys cannot be issued from both the profile and its extensions.

Some identification key samples:

- An identification key consisting of the "name" and "birthdate" attributes from the profile.
- A key based on the "pin" attribute of a single-valued extension used to hold customer pass codes.
- A key based on the "number" attribute of a multi-valued extension used to record the phone numbers we have seen a customer call from.

When UCS receives a new customer profile and the associated extension data, it builds the indexing structures according to the specified identification keys, in order to ensure efficient customer identification.

 Read [Identification Key](#) for a resource example.

## Preliminary Steps to Using Profiles

Before your application can start interacting with customers and recording their information by using [Customer Profile](#) resources, you must create certain types of information that enable you to manage these profiles. This includes the profile schema, extension schemas, and identification keys.

### Creating Extension Schema (Optional)

Operations and resources in this section are part of

If your application only needs to use the predefined profile attributes, you can use them as provided. But if you need to work with information that is not contained in the existing attributes -- for instance, for use as identification keys -- you can create extensions that contain this extra information by creating an extension schema for each additional piece of information.

 Your application cannot use an extension if the associated schema does not exist.

As for the profile schema, the profile extension schema is composed of [Attribute Schema](#) which define the schema content. The following output shows the definition of the single-valued "Address" extension.

```
"name": "Address",
  "type": "single-valued",
  "attributes": [
    {"name": "AddressType", "type": "integer", "default": 0},
    {"name": "Address", "type": "string", "length": 256},
    {"name": "City", "type": "string", "length": 32},
    {"name": "County", "type": "string", "length": 32},
```

```
{ "name": "PostCode", "type": "string", "length": 10 },  
  { "name": "Country", "type": "string", "length": 32 }  
]
```

The creation of the profile extension is possible through the [Create Profile Extension Schema](#) operation. See [this example section](#) for detailed information.

### Creating Identification Keys (Mandatory)

Operations and resources cited in this section are part of [Schema Operations](#)

If your application does not specify identification keys, the only way to access customer information is to [retrieve the customer profile](#) with the customer ID. However, you can use the [Create Identification Key](#) operation to set up an [Identification Key](#). You create these keys by using profile attributes. For example, you can use a single attribute, or a combination of attributes ("LastName" and "FirstName" for instance), or a combination of extension attributes if you have created a schema for a profile extension. You can define as many identification keys as you need, but you should consider that creating too many identification keys will slow down creation, update, and removal operations. Once identification keys have been created, your application can continue to work with customer data, accessing it by means of the [Get Identification Keys](#) operation.

Your application only needs to register these schemas and identification keys once.

## Managing Profiles: Creation, Identification, Extensions

The Customer Profile and its extensions should only be created for the storage of customer-level information. You should use the results of interactions and dialogs with the customer to fill in service, state, and task information.

### Managing a Profile

The [Create Customer Profile](#) operation creates an entry for the Customer Profile in UCS Database. This step must be completed prior to using the profile data in other operations such as:

- [Query Customer Profile](#)
- [Update Customer Profile](#)

### Identify a Customer

Operations and resources in this section are part of [Schema Operations](#)

The operation [Identify Customer](#) enables your application to retrieve customer profiles based on a few attribute values passed in as parameters, without specifying the customer ID, as shown in this example:

```
GET /profiles/  
contacts.phone_number=408-888-3214&extensions=contacts,purchases&exclude_profile=yes
```

`Exclude_extensions=unique`

If no customer profile is returned, your application can create a new profile based on the current set of information available.

This enables your application to determine the customer's identity without having to gather as much information. For instance, if your application deals with calls by using a customer phone number, the customer is easy to identify if he or she calls back.

### Extending the Customer Profile

As stated in the [introduction sections](#), your application can add new types of information to the basic customer profile by using extensions. If your application needs to record a specific set of data (for instance, e-mail addresses), your application can create a schema for this extension. Once an [extension schema has been created in UCS](#), your application can use the new extension and create associated records. Your application can either:

1. [Insert Extension Records](#) for a given customer, or
2. [Create Customer Profile](#) or [Update Customer Profile](#) with extension records.

### Importing Customer Profiles

#### **Available since:** 8.1.000.10

The [Bulk Profile Import](#) operation enables to import a .csv file which contains a wide set of contacts. The .csv file must be compliant with [RFC4180](#) and already available on the UCS local file system. Profiles must match metadata and can include extensions. An identification key can be set to avoid ambiguities with former profiles when updating the profile database. See [Bulk Profile Import](#) for examples and details.

### References

<references />

# Grouping Customer Profiles



**Purpose:** To group customer profiles.

## Overview

In the following example, the requirement is to enable an "account" or "family" view of multiple customer profiles. If a new object is created, the profile entity and attribute Group is set to true. If a multi-valued profile extension points to a group member profile, id-keys can be used. It then becomes possible to either attach services to the group, or to the member depending on the scenario.

## Examples

The following examples are similar with the difference being the way the profiles are created. Relationship between profiles and group are either "direct link" from entity account owner to other profiles, or "indirect link" where each profile belongs to a family profile.

### Account

There are many Profiles. Some are **Admin** for one or more other Profiles. For example, a telephony provider has a single account (billing account) for several persons. One of the persons is an *admin* for the account and has rights to change options for each cellular.

Example data:

- Profile with Id="XXXXXXXXXXXX-JOHN":

```
{
  CustomerId="XXXXXXXXXXXX-JOHN",
  LastName:"Doe",
  FirstName:"John",
  Cellular:"555-123456",
  EmailAddress:"john@doe.net",
  GroupAdmin: [
    {AdminForProfile:"XXXXXXXXXXXX-JANE"},
    {AdminForProfile:"XXXXXXXXXXXX-PETER"}
  ],
  ProviderOptions: { AccountInfo:"account number", MemberLevel:"High" }
}
```

- Profile with Id="XXXXXXXXXXXX-JANE":

```
{
  CustomerId="XXXXXXXXXXXX-JANE",
  LastName:"Doe",
}
```

```
FirstName:"Jane",
Cellular:"555-987654",
EmailAddress:"jane@doe.net",
ProviderOptions: { AccountInfo:"account number", MemberLevel:"Untouchable" }
}
```

- Profile with Id="**XXXXXXXXXX-PETER**":

```
{
  CustomerId="XXXXXXXXXX-PETER",
  LastName:"Doe",
  FirstName:"Peter",
  Cellular:"555-654321",
  EmailAddress:"peter@doe.net",
  ProviderOptions: { AccountInfo:"account number", MemberLevel:"Untouchable" }
}
```

- Two extensions for the example:
  - Single-valued extension **ProviderOptions** with any type of attributes used for the example to customize Cellular options.
  - Multi-valued extension **GroupAdmin** with attribute **AdminForProfile**
  - Identification keys :
    - on attribute **EmailAddress** from Core Profile
    - on attribute **Cellular** from Core Profile
    - on attribute **LastName+FirstName** from Core Profile
    - on attribute **AdminForProfile** of extension **GroupAdmin**
- Example scenario:
  - John calls to update his cellular account.
  - He is identified by his Cellular="555-123456".
  - The system knows that he is and admin for Jane and Peter because of the **GroupAdmin** extension.
  - He is asked "Do you want to change settings for your account #1, for Jane's account #2 or for Peter's account #3?".
  - If he enters #2 or #3, the system picks the correct Profile by the Id and can retrieve specific cellular options.
  - ...
  - Peter calls to change some options.
  - He is identified by Cellular="555-654321".
  - The system knows he is not Admin.
  - Depending on the requests, the system may "identify" who is admin for the cellular by the id-key on **GroupAdmin.AdminForProfile="XXXXXXXXXX-PETER"**.
  - The system might fail the request stating "need admin rights".

### Family

Each member of a family has its own profile. They belong to the Family Group (same house hold). This result in slightly different from the previous example because the Family itself is identified as a profile.

**Note:** The Account example can also be implemented this way.

Example data:

- Profile with Id="XXXXXXXXXXXX-JOHN":

```
{
  CustomerId="XXXXXXXXXXXX-JOHN",
  LastName:"Doe",
  FirstName:"John",
  Cellular:"555-123456",
  EmailAddress:"john@doe.net",
  GroupFamily: { ProfileFamily:"XXXXXXXXXXXX-DOE" }
}
```

- Profile with Id="XXXXXXXXXXXX-JANE":

```
{
  CustomerId="XXXXXXXXXXXX-JANE",
  LastName:"Doe",
  FirstName:"Jane",
  Cellular:"555-987654",
  EmailAddress:"jane@doe.net",
  GroupFamily: { ProfileFamily:"XXXXXXXXXXXX-DOE" }
}
```

- **Family = Profile 'XXXXXXXXXXXX-DOE':**

```
{
  CustomerId="XXXXXXXXXXXX-DOE",
  LastName:"Doe",
  PhoneNumber:"555-1592648",
  EmailAddress:"family@doe.net",
  PostalAddress: { Address:"5, This Road", ZipCode:65536 }
}
```

- Extensions:
  - Single-valued extension **PostalAddress** with attributes like 'Zip Code', 'State', etc. This extension may have values only for the **Family** since all profiles are to live at the same place.
  - Single-valued extension **GroupFamily** with attribute **ProfileFamily** pointing to the main family profile.
- Identification keys:
  - on attribute **EmailAddress** from Core Profile.
  - on attribute **Cellular** from Core Profile.
  - on attribute **PhoneNumber** from Core Profile.
  - on attributes **LastName+FirstName** from Core Profile.



- on attribute **ProfileFamily** from extension **GroupFamily**.
- Example scenario:
  - Assuming John sends the request from e-mail or cellular:
    - He is identified by id-key **Profile.Cellular="555-123456"** or **Profile.EmailAddress="john@doe.net"**.
    - Then his family information is gathered from querying Profile with Id **GroupFamily.ProfileFamily="XXXXXXXXXXXX-DOE"**.
  - Assuming John calls from Home:
    - His **Family** information is matched by id-key **Profile.PhoneNumber="555-1592648"**.
    - Members of the family can be identified by id-key on **GroupFamily.ProfileFamily="XXXXXXXXXXXX-DOE"**.
    - The IVR might question "Who are you? Jane or John?".

# Services, States, and Tasks

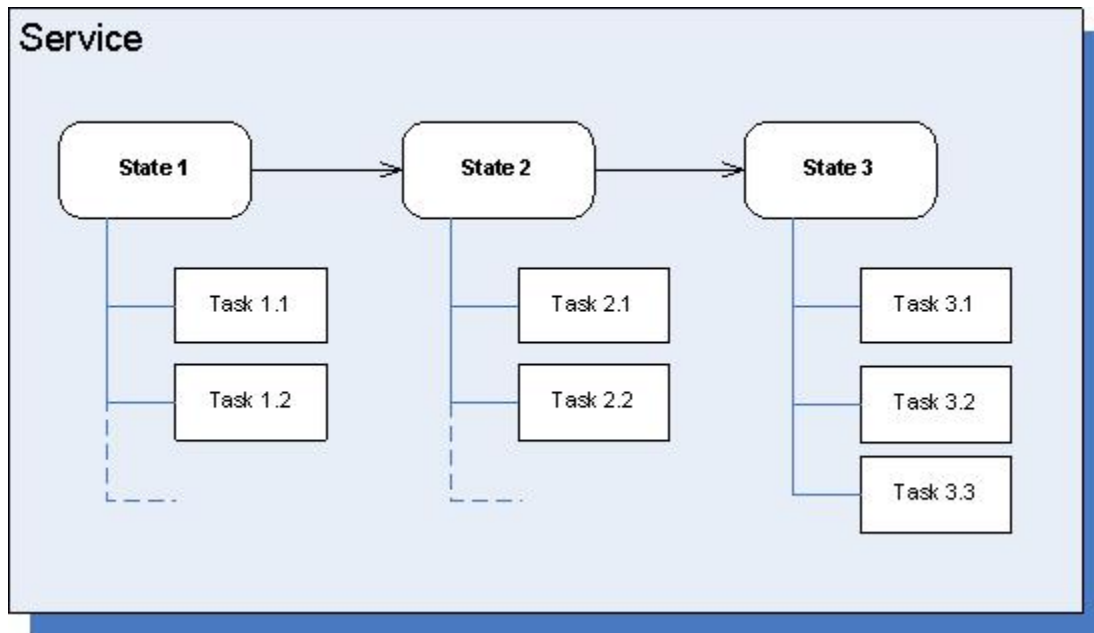


**Purpose:** Gives guidelines for managing Service information with Context Services.

In versions 8.1.000.10 and higher, you need to check the privileges set according to roles prior to using the operations described on this page. See [Role-Based Access Control](#) for additional details.

## About the Service, State, and Task Resources

Services are customer commitments defined by the business application (IVR, Orchestration, Agent Desktop, etc.) which interacts with the customer. Each service potentially spans multiple interactions over a variety of media channels and should link to a [Customer Profile](#) as soon as it is created or retrieved through identification operations (read [Profiles and Identification](#).) The Context Services REST API uses [Service](#), [State](#), and [Task](#) resources to manage and store the context information of your application. Basically, the [Service](#) resource is equivalent a top-level container associated with an overall commitment, which can be divided into a set of [States](#) to transition from one to another. These additional states can be divided into tasks.



Consider, for instance, an application that is a web-based interface, and that includes several online services, such as 'Booking a hotel reservation'. The service 'Booking a hotel reservation' is in charge of collecting information for the reservation.

- State 1: Collect Hotel Search Information

- Task 1: Collect Time Information (arrival, departure)
- Task 2: Collect Localisation Information
- Task 3: Collect Hotel Criteria
- State 2: Get Proposals
  - Task 1: Search offers in the database
  - Task 2: Propose offers
  - Task 3: Get Detailed Information about the offer
- State 3: Validate Proposal
  - Task 1: Get customer approval
  - Task 2: Make payment
  - Task 3: Validate reservation in the system
  - Task 4: Send bills and additional details by e-mail
  - Task 5: Collect customer feedback
- And so on.

If a customer starts interacting with the service, the application creates a new service resource to manage the service's context data, and then nested state and task resources to manage further states and tasks' context data.


## Services, States, and Tasks contents

The standard content of these resources is formal core information, as described in the related [Service](#), [State](#), [Task](#) pages, to determine:

- When the given service, state, or task started;
- Whether it is active or completed;
- Which interactions or customer are related to the given instance.

For each type of resources, the Context Services provide you with a set of operations which manage this basic data. For instance, in the case of a service, standard use cases imply that your application should:

1. [Start the service](#),
2. [Associate the service with a customer ID](#)-see [Anonymous Service](#) for further details;
3. Start and complete similarly states and tasks-see [List of State Operations](#) and [List of Task Operations](#);
4. [Complete the service](#) once all the nested state and tasks are completed.

 Note that your application is fully responsible for managing the status changes for the nested states and tasks. In other words, if you want to complete a given service, you must also complete the nested states and tasks.

## Active Resources

A service, state, or task is active if a customer is still interacting with it. In that case, the service, state, or task is started, but not complete. Once the resource is completed, it is no longer part of the active list, but part of the completed list.

 Read also [Query Services/States/Tasks](#).

## Extensions

The Context Services enable your application to record additional data related to the management of services, states, or tasks. In our introduction sample of 'Booking a hotel reservation', this would represent all the information collected through the service. If at any time the customer is disconnected and reconnects, your application is able to recover all this information with the Context Services. In this case, your application can define a set of [Extension Schema](#), and then add extension records which contain this information. As for profile extensions, extension records can be either "single-valued" or "multi-valued" (see [profile attribute values](#) for example).

 Read also [Extensions](#).

## Preliminary Steps to Using Services

### Creating Extension Schemas

Operations and resources in this section are part of [Schema Operations](#)

The creation of service, state, and task extensions is optional. However, if your application needs to extend the information of the service, state and task resources, you must create an extension schema to define the new extension content prior to any use.

- Use [Create Service Extension Schema](#) to provide extensions at the service level.
- Use [Create State Extension Schema](#) to provide extensions at the state level.
- Use [Create Task Extension Schema](#) to provide extensions at the task level.

The following sample create an [Extension Schema](#) which defines the Content of the Feedback extension:

```
POST /metadata/services/extensions
{
  "attributes":
  [
    {"name": "FeedbackType", "type": "string", "length": "10", "mandatory": "true"},
    {"name": "rating", "type": "integer", "mandatory": "true"},
    {"name": "notes", "type": "string", "length": 256, "mandatory": "false"}
  ],
  "name": "Feedback",
  "type": "single-valued"
}
```

## Basic Service, State, and Tasks Management

Operations and resources in this section are part of **Service Operations** and its subcategories.

### Start the Service

1. **Start Service**: first step in your service management. Your application creates a service instance each time that a new information context needs to be created. (In our example, each time a customer enters in the Booking reservation service, UCS creates the core service information, including a service ID returned as a result of this operation.)
  - If you have no information to **create** or **identify** the customer, your service is **anonymous**. In that case, use a contact key.
  - When you start the service, you pass in the operation's body the **Service Start Event** which describes the start information.
2. **Associate Service**: To use later, once you have a customer ID to associate with your service. To get a customer ID, you need to retrieve profile information (see **List of Profile Operations**).

The following operation starts a new service with a contact key:

```
POST /services/start
{
  "timestamp": "2009-05-12T12:05:12.145Z",
  "interaction_id": "123ABCAADFJ1259ACF",
  "application_type": 400,
  "application_id": 40,
  "est_duration": 60,
  "contact_key": "42",
  "service_type": 100,
  "media_type": 1,
  "resource_id": 5005,
  "resource_type": 2,
  "disposition": 10,
  "coupon": {
    "coupon_name": "DISCOUNTCODE15"
  },
  "satisfaction": {
    "score": 85,
    "agentID": 2025
  },
  "relatedOffers": [
    {
      "offer_name": "VIP credit card black ed.",
      "type": 9,
      "comments": "proposed to all client"
    },
    {
      "offer_name": "3 times payment GOLD",
      "type": 4,
      "comments": "limited offer"
    },
    {
      "offer_name": "life insurance",
      "type": 3,
      "comments": "healt check to be done before approval"
    }
  ]
}
```

```
}
```

## Manage States or Tasks for a given Service

Your application can use both States and Tasks, or Tasks only.

1. **Start State** or **Start Task**: In the corresponding Start Event, you must specify to which service the state or task belongs by filling the 'service\_id' parameter.
2. **Perform State Transition**: if your service contains several states, you can perform state transition instead of completing a state and starting a new state.
3. The state transition does not complete the tasks which belong to the completed state. Your application must complete them before performing this operation.

The following example shows a state transition:

```
POST /services/735692/states/transition
{
  "timestamp": "2009-05-07T12:05:20.157",
  "session_id": "11000ABC-80236C1A-1010",
  "interaction_id": "123ABC908ABFFD8080",
  "from": {
    "state_id": 1001,
    "disposition": 1,
    "disposition_desc": "SUCCESS",
    "Feedback": {
      "FeedbackType": "survey", "rating": 7,
      "notes": "warm welcome at frontdesk, thanks for the nice trip"
    },
    "Satisfaction": [
      {
        "rating": 2,
        "pertinence": 8,
        "usefull": true,
        "place": "Terranova mexico resort"
      },
      {
        "rating": 8,
        "pertinence": 4,
        "usefull": false,
        "place": "Fancy resort Paris"
      }
    ]
  },
  "to": {
    "state_type": 8,
    "est_duration": 500,
    "Sponsoring": { "Rank": "first", "expire": 7,
      "notes": "give customer free meal" }
  }
}
```

## Query the Services/States/Tasks for a Given Profile

In [Query Services](#), [Query States](#), and [Query Tasks](#) operations, you can query lists of active or completed resources, by filtering in the URL the active or completed status of the resources. For

---

instance, in [Query Services](#):

- Active Services: GET /customers/\${customer\_id}/services/active
- Completed Services: GET /customers/\${customer\_id}/services/completed

In addition, the [Query Services](#) and [Query States](#) operations enable to retrieve the nested states and/or tasks, within the results. For instance, the following query operation returns the active services within their active states associated with the customer profile *ABC1234*. **Operation**

```
GET /customers/ABC1234/services/active?active_states=true
```

### Response


```
[ // returned in an array
  { "customer_id": "ABC1234",
    "service_id": 4692834,
    "est_duration": 86400,
    "started": {
      "timestamp": "2009-05-07T12:05:20.157",
      // additional Start Event fields
    },
    "active_states":
      [// included given specification of "results" attribute
        { // array of one or more State objects
          "state_id": 5005,
          "state_type": 8, // service delivery
          "started": {
            "timestamp": "2009-05-07T12:08:53.298",
            // additional Start Event fields
          }
        }
      ]
    }
  ]
}]
```

### Complete Service/State/Task

When your customer stops interacting with the given service, state, or task, you must complete this resource and mark it as terminated in the UCS database. This enables you to filter the result of query operations based on the resource status (as described in the [Active Resources section](#)).

- You are responsible for performing the [Complete Service](#), [Complete State](#), [Complete Task](#) operations for any service, state, or task that you started.
- These operations apply only to the resource specified in the operation's parameter and they do not modify the status of the nested states and tasks, if any.

The only case which does not force you to explicitly complete a state with the [Complete State](#) operation, is [Perform State Transition](#), which completes the given state then starts a new state.

 **Tip:** to make sure that you correctly completed the states and tasks of a given service, use the [Query States](#) and [Query Tasks](#) operation with active filters to check that no resources remain active.

## Read More

- [Extensions](#)