

# Developer's Guide

## Services, States, and Tasks

12/15/2025

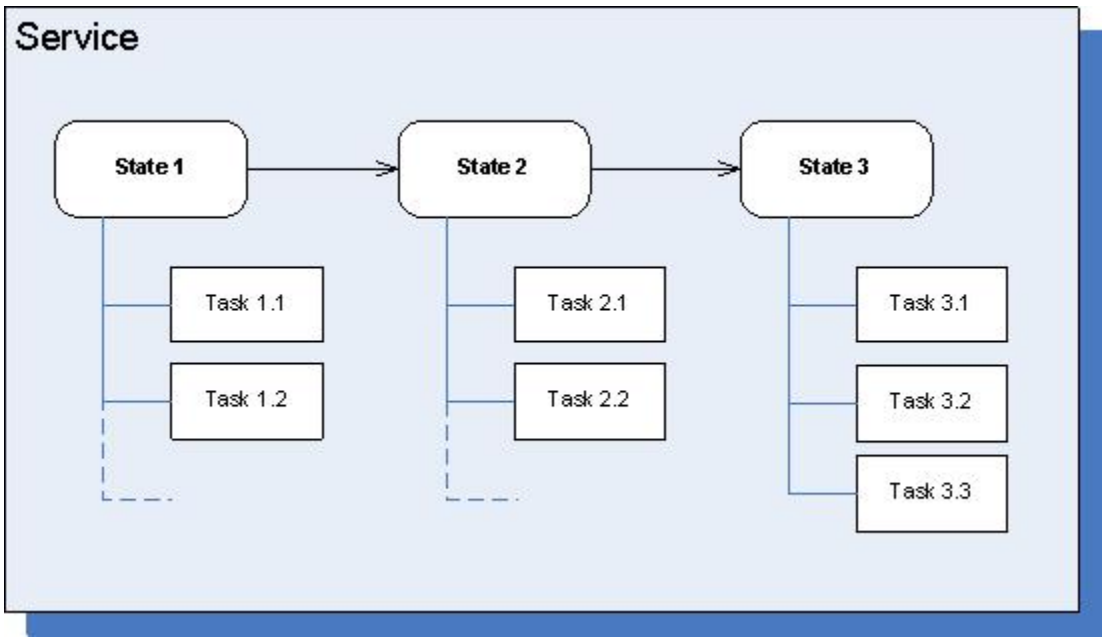
# Services, States, and Tasks

This page gives guidelines for managing Service information with Context Services. In 8.5, all the service management is handled on top of GMS and is simplified to facilitate the whole data process.

## 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 email
    - 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 Operations;
4. Complete the service once all the nested states and tasks are completed.

> ### Important
> 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.

> **Important**
> Read also Query Services/States/Tasks

.

### Extensions

You can use extensions 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, you can fetch this information in Context Services and recover the conversation.

To add or update extension data, you simply add key-value pairs to REST queries where the key is a string, and the value is some JSON data (for example, a string, a JSON array, or a JSON object).

> **Tip**
> Context Services ensures backward compatibility with 8.1. If you migrate from 8.1 to 8.5, you do not need to modify your service queries, and you no longer need to handle schemas for services and their nested resources. You will find related instructions here.

In the API reference, you can see whether you can add or update extensions if the following attribute is available:

| Name | Type | Mandatory | Description |
|---|---|---|---|
| <extension key> | Any JSON type | No | Service attached data as key-value pairs. You can add as many key-value pairs as needed. |

## 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. You create 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"
          }
  ]
}
```

In the above query, `coupon`, `satisfaction`, and `relatedOffers` attributes are extensions.

## Manage States or Tasks for a given Service

You 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.

## Auto-Complete Service

**Introduced in 8.5.111**

Instead of completing a service manually, you can enable auto-completion by setting the auto-complete-enabled option to `true` and by configuring an auto-complete-after time in your GMS configuration. In that scenario, the system will wait for the given `auto-complete-after` time to complete the service after its last update and/or the last update to its nested states and tasks.

If the feature is enabled, you can also choose to override this auto-complete-after time by passing this value in the `auto-complete-after` parameter of your Start Event at the service start.

Additionally, if the auto-complete feature is enabled and if you neither configure a default value nor pass one at service creation, you will have to complete the service with a REST query to make it end.

## Improve Performance while Querying Data Associated with a Customer

**Introduced in 8.5.111**

When you query customer or services resources, you can speed up the results by limiting the number of services to be returned. To implement this limit, you can add the parameter `max_services` to your GET query and only the last created services will be returned.

This parameter can be added to the following queries, detailed in the Query Services page:

- GET /genesys/1/cs/customers/${customer_id}/services
- GET /genesys/1/cs/customers/${customer_id}/services/active
- GET /genesys/1/cs/customers/${customer_id}/services/completed
- GET /genesys/1/cs/services/anonymous/${contact_key}
- GET /genesys/1/cs/services/anonymous/${contact_key}/active
- GET /genesys/1/cs/services/anonymous/${contact_key}/completed

## Bulk Updates of Service, State, Task

### Introduced in 8.5.200

To perform a bulk update, use the composite start query and specify `update_extensions=true` in the start event. Instead of creating a new object, the operation will update the provided extensions of the given objects. This means that only the specified extensions will be updated. For example, if your service's extensions include the `Satisfaction` extension, and if you do not provide this extension in the query, the extension will neither be updated nor removed from the service.

For details and examples, refer to Composite Start and Bulk Update.