SmartHub INFER™

API Reference Guide

Last updated on June 3, 2024

Find out more about our products and solutions at https://www.smarthub.ai/

Table of Contents

1 Introduction **3**
1.1 APIs . . . 3
1.2 Headers . . . 4
1.3 API Version . . . 4
1.4 Authentication . . . 4
1.5 Organizations . . . 7
1.6 Device Authentication . . . 8
1.7 Restricted Characters . . . 9

2 Server APIs **11**
2.1 Swagger Console . . . 11
2.2 Using the Server APIs . . . 11
2.3 Server API Types . . . 12

3 Edge APIs - Python SDK **14**
3.1 Python SDK . . . 14
3.2 Supported Operations . . . 15
3.3 Best Practices . . . 18

4 Running Campaigns using Agent SDK **20**
4.1 Running a Campaign using Default Properties . . . 20
4.2 Running a Campaign in On-Demand Mode . . . 20
4.3 Running a Campaign in Headless Mode . . . 22
4.4 Approving the OTA Update Phases . . . 22

5 Writing an Adapter using C SDK **24**
5.1 DefaultClient in IoTCAgent Package . . . 24
5.2 Data Structures . . . 26
5.3 Functions . . . 35
5.4 Macro Definitions . . . 50
5.5 Enumeration Types . . . 51
5.6 Writing a Client Application using IoTCAgent SDK . . . 53
5.7 Building a Client that uses the IoTCAgent SDK . . . 57
5.8 Running a Client that uses the IoTCAgent SDK . . . 57
5.9 Working with DefaultClient . . . 57
5.10 Using DefaultClient Daemon . . . 58

1 Introduction

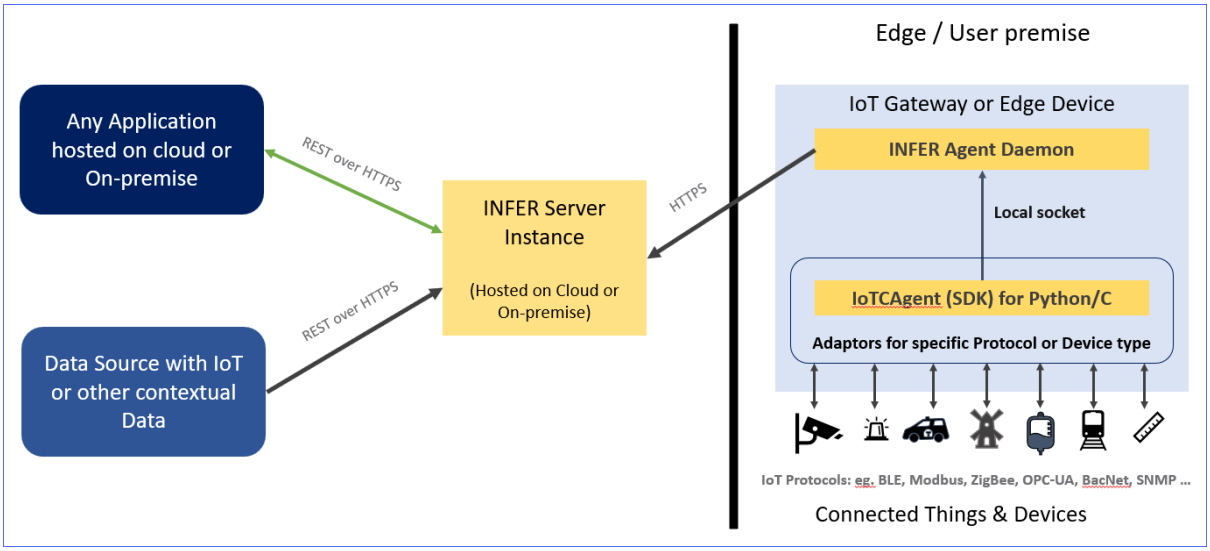
INFER™'s APIs power its platform for IoT and Endpoint Management. INFER™ has a host of REST APIs of all of its core features. Behind these APIs is a software layer connecting and optimizing your edge devices across your enterprise spaces to allow their seamless lifecycle management.

Using APIs, you can programmatically create, view, edit, and delete various entities such as:

- Devices,
- Campaigns,
- Alerts,
- Notifications,
- Groups, and
- Users.

1.1 APIs

INFER™ provides APIs for different types of integration as shown below:



1.1.1 Server APIs

Using this set of REST APIs, the Server enables your applications to:

- read data stored in ,
- write data into , and
- control actions performed by on your IoT and edge devices.

All Console functionalities are implemented using this same set of REST APIs.

Note: To consume these APIs, you must have inbound HTTPS access to the instance, regardless of whether it deployed on-premise or in the cloud.

1.1.2 Agent APIs

This is a set of Python Functions provided as an SDK library to enable Edge Adapters or other applications running at the Edge to inject IoT data into via Agent Daemon. The SDK library can be consumed by Python, C or other language programs that implement a particular protocol to interface with devices. For more information, see **Edge APIs - Python SDK**.

1.2 Headers

INFER™'s REST APIs support several standard and custom HTTP headers, including both request headers and response headers specific to.

You can use headers to pass parameters and customize options for HTTP requests.

Common headers used include:

- `x-current-org-id` —Enter this header if the user name is available across multiple organizations.
- HTTP Accept—Indicates the format that your client accepts for the response body. Possible values are `application/json` and `application/xml` . The default value is `application/json` .
- HTTP Content-type—Indicates the format of the request body that you attach to the request. Possible values are `application/json` and `application/xml` .
- HTTP Authorization—Provides the OAuth 2.0 access token to authorize your client. REST API supports the Bearer authentication type.

1.3 API Version

```
Content-Type: application/json Accept:
application/json;api-version=\<api-version\>
```

To get the current API version, use the following API:

API `/api/versions`

Method `GET`

Sample Response

```
{
  "currentApiVersion": "0.2",
  "supportedApiVersions": [
    "0.1",
    "0.2"
  ]
}
```

1.3.0.1 Response Parameters

Field	Type	Description
`currentApiVersion`	string	The current API version
`supportedApiVersions`	array of strings	List of API versions

1.4 Authentication

Use the following APIs to create and issue an authentication token for a user.

1.4.1 Acquire API Keys

There are two ways to authenticate using 's Server-side APIs.

The recommended method to access server-side APIs is to use API key. For more information, see the **API Keys** chapter in the INFER™ User Guide.

1. If you are logging in to INFER™ via SSO or LDAP, you can use an API key generated from the Console itself. These API keys give you life upto one year and is ideal for automation or enterprise integration.

Note:

a. For accessing server APIs, the following headers are mandatory: `Accept` : with server API version. `application/json;apiversion=0.17"` .

Header

`Authorization` : For **POST** and **PUT** call methods, specify the only supported content-type header as `application/json` .

2. You can also use your local credentials. However, these tokens come with a short life and need to be refreshed often. The maximum token life is 365 days.

Note: This option is not recommended for enterprise users accessing via SSO or LDAP integration.

1.4.2 Acquire Token using Credentials

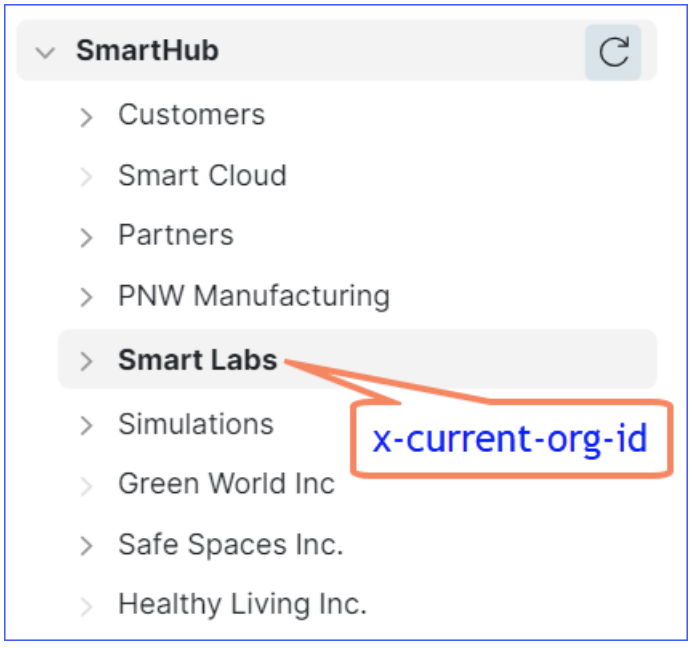
You require a user access token to perform API operations. However, if you already have an API key which is similar to a token, this is not required.

```
Authorization : {UserAccessToken}
```

Header

`Authorization` : This uses the basic auth technique explained here:

https://inst01.us01.infer.smarthub.ai/openapi/index.html.



If the user name is available across multiple organizations, enter the following header:

```
x-current-org-id
```

API `/api/tokens`

Method `GET`

Required Parameters `None`

Response

```
{
  "accessToken": “string”,
  "expiresInSecs": "1543317540",
  "accessTokenExpiresAt": "1543317540",
  "refreshToken": “string”,
  "refreshTokenExpiresAt": "1544519940"
}
```

1.4.2.1 Response Parameters

Field	Type	Description
`accessToken`	string	Access token
`expiresInSecs`	long	The remaining milliseconds left for expiry of access token
`accessTokenExpiresAt`	long	The time (in milliseconds) of the access token's expiry
`refreshToken`	string	The replacement token of the old token
`refreshTokenExpiresAt`	long	The remaining milliseconds left for expiry of new access token

1.4.3 Issue Access Token Using Refresh Token

API `/api/tokens/refresh`

Method `GET`

Required Parameters `None`

Response

```
{
  "accessToken": "string",
  "accessTokenExpiresAt": "1543317540"
  "expiresInSecs": 0,
  "refreshToken": null
}
```

1.4.3.1 Response Parameters

Field	Type	Description
`accessToken`	string	Access token
`accessTokenExpiresAt`	long	The time (in milliseconds) of the access token's expiry
`expiresInSecs`	long	The remaining milliseconds left for the access token's expiry
`refreshToken`	null	The replacement token of the old token

1.5 Organizations

1.5.1 Setting the Current Organization ID

Use this header to set the current organization ID for which you want to run the APIs.

```
x-current-org-id:\<orgId\>
```

1.5.2 Listing Organizations

Use the following API to list all organizations in your scope.

API `/api/organizations`

Method `GET`

Required Parameters `None`

Response

```
{
  "pageInfo": {
    "totalPages": "string",
    "totalElements": "string",
  "page": 0,
  "pageSize": 0
  },
  "tenants": [
  {
    "id": "string",
    "name": "string",
    "parentId": "string",
    "status": "ACTIVE",
    "ancestors": [
      "string"
      ],
      "orgId": "string",
      "lastUpdatedBy": "string",
      "createdBy": "string",
      "lastUpdatedTime": "string",
      "createdTime": "string",
      "updateVersion": 0
    }
  ]
}
```

1.5.2.1 Response Parameters

Field		Type	Description
`pageInfo`		string	Name of the column
	`totalPages`	string	The total number of pages
	`totalElements`	string	The total number of elements
	`page`	integer	The current page
	`pageSize`	integer	The number of elements in a page
`tenants`		string	Name of the column

Field		Type	Description
	`id`	string	The tenant's id
	`name`	string	The tenant's name
	`parentId`	string	The tenant's parent id
	`status`	string of arrays	The tenant's present state
	`ancestors`	array	The parent organizations of the current organization in the hierarchy
	`orgId`	string	The organization's id
	`lastUpdatedBy`	string	Name of person who last updated the organization's id
	`createdBy`	string	Name of person who created this organization's id
	`lastUpdatedTime`	string	The time when this organization's id was updated last
	`createdTime`	string	The time when this organization's id was created
	`updateVersion`	integer	The version number of the updated version

1.6 Device Authentication

Use the following API to issue device token and device credentials.

1.6.1 Creating Device Credentials

Required Permissions

You must have the **Create Device Credential** permission to perform this operation.

API `api/device-credentials/{id}`

Method `POST`

Input Examples

- JWT_NATIVE is a token based authentication where a gateway can be enrolled into the Server using this one time token.
  - Request body: `{}`
  - Path parameter: device id (string)
- PROPERTY_NATIVE refers to property based enrollment. The returned token is ignored. For more information, see the **Onboarding a Gateway using Property-based Authentication** chapter in the INFER™ User Guide.
  - Request body: `{"requestParams":"{\"DeviceKey\":\"1234\"}"}`
  - Path parameter: device id (string)
- TPM_NATIVE is TPM based enrollment. For more information, see the **Onboarding a Gateway using TPM-based Authentication** chapter in the INFER™ User Guide.

- Request body: `{"requestParams":"{\"tpm_ek\":\"123456\"}"}`
- Path parameter: device id (string)

Note: Enrollment flow is defined by the device template for a gateway.

Response

```
{
  "credentials": "string"
}
```

1.6.1.1 Response Object

Field	Type	Description
`credentials`	string	The device's credentials

1.6.2 Get Device Token

Required Permissions

You must have the **Get Device Token** permission to perform this operation.

API `/api/device-tokens`

Header `x-device-auth`

Enter the device credential that you created in the **[Create Device Credential]** API.

Method `GET`

Required Parameters

Field	Type	Description
`id`	string	Device ID

Response

```
{
  "deviceId": "string",
  "accessToken": "string"
}
```

1.6.2.1 Response Parameters

Field	Type	Description
`deviceId`	string	The device's id
`accessToken`	string	The access token

1.7 Restricted Characters

The following characters are restricted when creating a template name, device name, custom property, and metric name.

1.7.1 Template Name

`< > % $ ( ) { }`

1.7.2 Device Name

```
< > % $ ( ) { } [ ]
```

1.7.3 Custom Property

```
< > . % $ ( ) { }
```

1.7.4 Metric Name

```
: { } & "
```

1.7.5 User Name

- Usernames must begin and end with alphanumeric characters.
- Usernames can contain letters (a-z), and numbers (0-9).
- Usernames can contain zero or one occurrence of a separator (hyphen(-), underscore (_), or period (.).
- Usernames must not contain the following special characters:

  ‘ `~!@#$%^&*(){}[]+=;:'"<>?/|,`

Examples of disallowed usernames:

- ”.pulseuser”
- ”pulseuser-”
- ”pul.se-user”
- ”pulse@user”
- ”!pulseuser”

As a local user, you can:

- choose a username 1–50 characters long.
- use alphanumeric values along with one of the following special characters listed below:
  - `underscore ( _ )` ,
  - `hyphen ( - )` , and
  - `period ( . )`

As an SSO/LDAP user, your username must contain only one of the following special characters listed below:

- `underscore ( _ )` ,
- `hyphen ( - )` , and
- `period ( . )`

All other special characters are disallowed.

2 Server APIs

The INFER™ Server gives you this set of REST APIs to enable your applications to read data stored in , write data into and control actions performed by on your IoT and edge devices. All INFER™ Console (Console) functionalities are implemented using this same set of REST APIs.

Note: To consume these APIs, you must have inbound HTTPS access to the INFER instance, regardless of whether it deployed on-premise or in the cloud.

2.1 Swagger Console

INFER™'s Server APIs are OpenAPI compliant and provide the Swagger Console as part the Server instance.

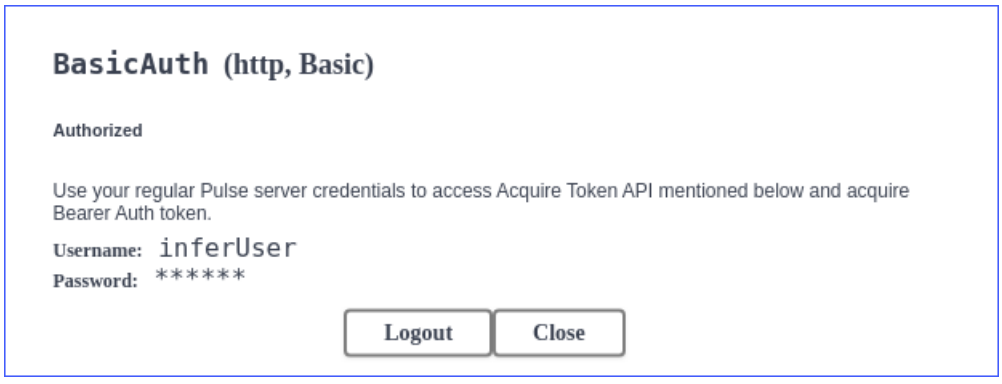
The Swagger Console provides detailed documentation for all RESTful APIs offered by the Server. It also provides the ability to invoke the APIs interactively with a live server connection. For more information about Swagger Console, see swagger.io.

To access the Swagger Console, point your browser to
`https://<INFER-SERVER-FQDN>/openapi/index.html`
where `<INFER-SERVER-FQDN>` is the Fully Qualified Domain Name (FQDN) or IP address of the Server instance.

2.2 Using the Server APIs

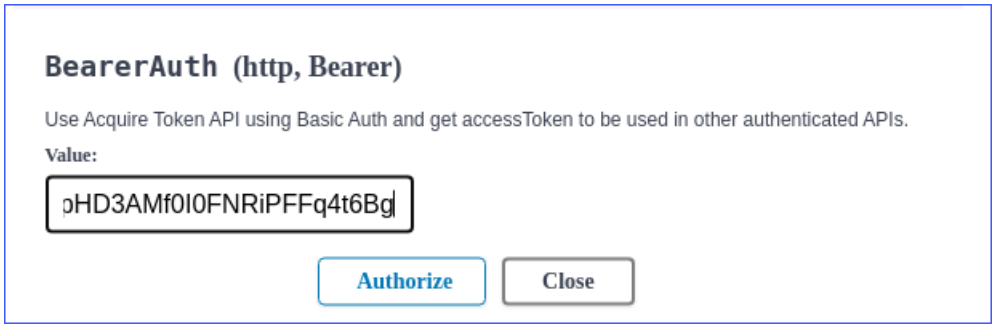
To invoke APIs on the Server, you need to be authenticated first. This is done by invoking the following APIs in sequence.

1. Ensure that the **Servers** drop-down list at the top-left of the page is showing the correct FQDN for the instance you want to connect to.
2. Click **Authorize** at the top right corner to perform `BasicAuth` to the Server.
3. Click **Close** once you are logged in.

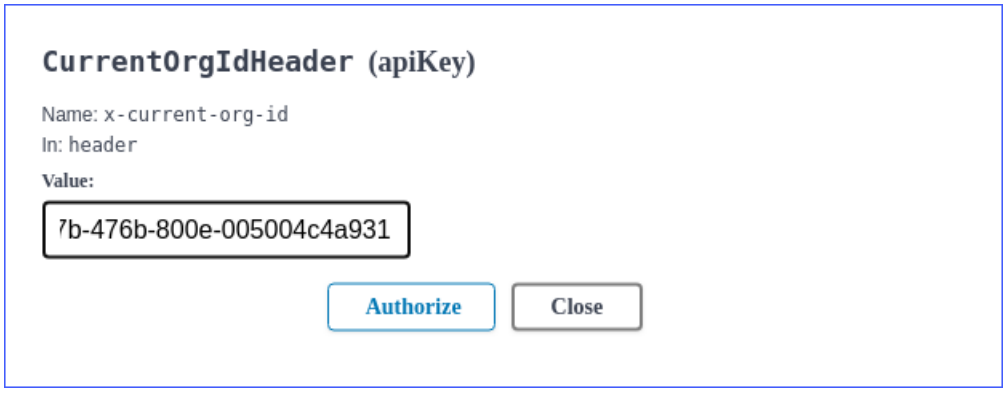


4. Next, get the API versions supported by the server by invoking **API Version** call. For all subsequent calls, ensure that a supported version is provided in the `Accept` header. In the Swagger Console, the api-version value is automatically added to the header, but when you are invoking the APIs, you must send this parameter in the header.
5. Acquire an **AccessBearer** token that you can use for all subsequent calls in the session. You can do this by invoking the **[Acquire Token]** call. The response will provide two tokens:
   - `accessToken`
   - `refreshToken` .

6. Authorize all other REST calls using the value of `accessToken` as **BearerAuth** token. Click the **Open Lock** icon for any of the REST APIs to open the popup and enter the value of `accessToken` returned by Acquire Token call.



6. Additionally and **optionally**, you can set the scope of the APIs for a particular sub-organization. This can be done by providing `orgId` as a header in the `CurrentOrgIdHeader` by opening the popup as mentioned above. If there's no value provided, the scope of organization is automatically identified by the server from the BearerAuth token. `OrgId` can be found in the organization list which could be retrieved from the **[List Organizations]** API.



2.3 Server API Types

This section provides a high-level overview of the Server APIs along with some sample use cases.

- **Device Management** - APIs for performing operations on devices, device templates, device authentication, device commands, and files.
- **Campaign Management** - Create, modify, delete, start, stop and other operations on Campaigns. Create, delete and management of Packages.
- **Alerting** - APIs to get, create, update, and delete alerts and alert definitions.
- **Identity & Access Management (IAM)** - APIs to perform Tenant management, Sub-org management, Role management, User management, and Group management operations.
  - **Organization Management** - Create, view, update, and delete an organization.
  - **User Management** - APIs to create, fetch, update, and delete users.
  - **Role Management** - APIs to create, fetch, update, and delete roles.
  - **Group Management** - APIs to create, fetch, update, and delete user groups.
  - **Permission Management** - API to fetch permissions.
  - **Organization Settings** - APIs to view and update your organization settings.

  - **Password Management** - APIs to generate a password recovery link and reset the password.
  - **Token Management** - APIs to generate access and refresh tokens.
- **Notification** - APIs for Notification Destinations, Notification Definitions, and Notification Instances.
  - **Notification Definition** - APIs to create, update, get, and delete notification definitions.
  - **Notification Instances** - APIs to retrieve notification instances.
  - **System Notifications** - APIs to view system notifications. System notifications are generated when there is a system downtime. The notifications are sent to the users through email or displayed on the Console.
- **Certificate Management** - APIs to create, update and delete certificates.
- **Advanced Search** - APIs to create, get, update, and delete a filtered device list.
- **Metric APIs** - APIs to query metrics.
- **Audit APIs** - APIs to get audit logs, get audit types, and get entity types.
- **Space APIs** - APIs to create, edit, delete, and assign/unassign parent spaces.
- **Space Template APIs** - APIs to create, clone, edit, delete, and assign parent space templates.

3 Edge APIs - Python SDK

This section provides information about working with the Agent's SDKs.

The Agent is a component that resides in the Gateway, and connects the INFER™ services to run commands and send operational metrics to IoTC services.

Note: The Agent makes an outbound connection to the Server on port 443 (HTTPS).

The Agent's SDK called as **IoTCAgent** exposes APIs on the gateway which Third-party applications can use to interact with .

3.1 Python SDK

All the operations you can use are part of the InferSession class. Sample adapter covers all such operations. However, sample adapter is just a reference on how to use the SDK. It will not run on all gateways since you will have different templates in scope than what is mentioned in the sample_adapter.py.

3.1.1 Requirements

- **Iotcagent** installed, and running.
- **Python3** and **pip3** installed.

3.1.2 Installation steps

1. Navigate to **pythonSDK** folder
2. Run -:
   i. Linux: sudo bash install.sh
   ii. Windows: .\install.ps1 (run the command using cmd/powershell window with Administrator privileges)

3.1.3 Configuration

To use the SDK, a mandatory configuration file adpater_config.json has to be present in the same working directory as the adapter code. Following are the major sections in the configuration:

```
"logging":{
"level": "INFO"
}
```

You can set the log level of logs of the adapter and SDK. Log levels can be the following:

- DEBUG
- INFO
- WARNING
- ERROR
- CRITICAL

```
"c-sdk": {
    "adapterName" : "sample-adapter"
    "number_of_init_session_retries": 100,
    "retry_period_seconds": 60,
    "inter_process_communication": {
    "ipc_mode": "TCP",
```

```
    "tcp_port": 5000
  }
}
```

Field		Type	Description
`c-sdk`		Top level	Name of the SDK
	`adapterName`	string	The name of the adapter
	`number_of_init_session_retries`	integer	The number of times adapter will try a reconnection to the agent in case the socket session is lost
	`retry_period_seconds`	integer	The duration after which a retry is attempted
	`inter_ process _ communication`	array	Specifies whether `ipcmode` is “UDS” or “TCP”. On **Linux**, default agent ipc is **UDS**. On Windows, default agent ipc is **TCP on port 5000**.

The SDK can potentially communicate to the agent via various means. For now, native SDK (C SDK) is being used underneath to communicate with the agent via inter process communication.

Apart from the aforementioned sections, you can add more sections to suit the adapter's needs. You can also use additional custom configuration files to configure the adapter.

3.1.3.1 InferSession

The Adapter communicates to the agent by opening a session using the class `InferSession` .

```
from infer_adapter_sdk.session import InferSession infer_session =
InferSession(application_id=\"com.smarthub.axis.adapter1\")
```

- `application_id` is a unique string which identifies the adapter with the agent. There would be few adapters talking to agent on a Gateway.

Note: Keep the `application_id` unique among `application_ids` of all adapters communicating with a particular agent.

3.2 Supported Operations

3.2.1 Get Gateway Device

```
infer_session.get_gateway_device()
```

3.2.2 Sending Properties

Multiple properties can be sent using the method:

```
infer_session.send_properties(device_id, {"abc": "def", "123":
"456"})
```

3.2.3 Sending Metrics

Metrics can be sent using:

```
infer_session.send_metric(device_id, metric_name="testint",
metric_type=Device.MetricType.INTEGER, metric_value=1)
infer_session.send_metric(device_id, metric_name="testdouble",
metric_type=Device.MetricType.DOUBLE, metric_value=2.3)
infer_session.send_metric(device_id, metric_name="teststring",
metric_type=Device.MetricType.STRING, metric_value="abc")
infer_session.send_metric(device_id, metric_name="testboolean",
metric_type=Device.MetricType.BOOLEAN, metric_value=True)
```

- A Metric can be sent along with an optional timestamp. In case no timestamp is provided, it would default to current time stamp.

```
infer_session.get_ginfer_session.send_metric(device_id, metric_
name="testint", metric_type=Device.MetricType.INTEGER, metric_
value=1, timestamp_milli_sec=<timestamp_in_
milliseconds>)ateway_device()
```

3.2.4 Enrolling a pre-registered Gateway

A pre-registered gateway has to be enrolled after the agent is installed on the same. There are multiple ways to enroll (and register) a gateway on the edge. One of the ways supported by this SDK is enrolling using an authentication token. This token can be generated on once the gateway is registered.

```
infer_session.enroll_gateway_pre_registered(authentication_token)
```

3.2.5 Registering and Enrolling a Device/Thing

- A registered device (registered in) can be enrolled on the edge using:

```
registered_device = next(

    (device for device in infer_session.devices if
(device.enrollment_state ==
Device.EnrollmentState.REGISTERED)),

    None)


infer_session.enroll_registered_thing(registered_
device.device_id).
```

- A device can also be both registered and enrolled at the edge:

```
thing1 = infer_session.enroll_thing(device_name="thing1",
template_name="testDevice", parent_device_id=gateway_
device.device_id)

print(f"thing1's deviceId: {thing1.device_id}")
```

3.2.6 Un-enrolling a Device

```
infer_session.un_enroll_device("<device-id>")
```

3.2.7 Refreshing Device Data

All the data related to devices connected to the Gateway can be refreshed from the Server:

```
infer_session.refresh_devices()
```

Note: This command will cause a round-trip to INFER™ Server.

3.2.8 Callbacks

Command Callbacks can be registered, which upon a command trigger from would be called back. Signature of callback function:

```
def callback_example_reboot(infer_session: InferSession,
command: Command) -> CommandResponse:

    yaaay_ = f"I'm a callback. yaay! got called. command
name: {command.name}"

    logging.info(yaaay_)
    # infer_session.send_properties(infer_session.get_
gateway_device().device_id, {"abc": "def"})

    # reboot logic

    print(yaaay_)
    for arg in command.arguments:

      print(arg.name + " " + arg.value_)

  '''
  return CommandResponse with command_result as :
CommandResult.SUCCESS or CommandResult.FAILURE.

  CommandResponse expects an enum CommandResult, and an
optional custom message.

  In case the command result is deferred, return
CommandResponse(CommandResult.UNKNOWN).

  For posting the result of a deferred command, do:

  infer_session.post_command_result(command_id,
CommandResponse(CommandResult.SUCCESS, "custom message") #
or CommandResult.FAILURE

  '''
  return CommandResponse(CommandResult.SUCCESS, "custom
message")
```

- To register a command callback:

```
infer_session.register_callback(command_name="reboot",
callback_function=callback_example_reboot)
```

- To post the result of a deferred command, do the following:

```
infer_session.post_command_response(command_id,
CommandResponse(CommandResult.SUCCESS, "custom message") #
or CommandResult.FAILURE
```

To post a command's result asynchronously, you need the **command_id**, which you can access from the command object as `command.id` .

3.2.9 Uploading a File

You can upload any file on the local system to the Server. **Device_id** is optional; however, if `device_id` is not provided, the file will be associated with the Gateway's `device_id` .

```
infer_session.upload_file(file_path="<path-of-the-file
-onthe-system>", device_id="<device_id>")
```

3.2.10 Adapter Sleep using SDK Function

You can use the `sleep_and_process_command` utility to put the THING (adapter) to sleep and execute the commands sent to it in the background, in a single thread.

```
sleep_and_process_command(infer_session, sleep_time_sec=60,
process_command_interval_sec=15)
```

- `infer_session` stands for the object of the InferSession class.
- `sleep_time_sec` is the value set for the duration of the adapter's sleep cycle, and which also denotes the frequency of sending metrics and properties.
- `process_command_interval_sec` is the value set for the adapter's poll frequency to fetch the commands from the agent and execute them.

3.2.11 Closing the Session

After file upload you can close the session once the work is done.

```
infer_session.close_session()
```

3.3 Best Practices

3.3.1 Debugging and Logging

Often the Adapters run on the edge Gateways with limited remote access facility. To debug, it is paramount to write meaningful logs. INFER™ can pull log files from the Gateway on demand. If you have access to the actual Gateway system, the logs could be visualized from logging utilities like:

- **journalctl**
- **syslog** (Linux), or
- **event log** (Windows).

3.3.2 Configuration

Often, adapter deployments happen on multiple Gateways catering to a group of devices each. In order to affect configuration on multiple adapters/devices at once, custom properties come handy.

INFER™ allows you to update custom properties in bulk at once. Configurations like `polling_period` , etc can be done using the custom properties rather than using configuration files.

3.3.3 Reliability

Each adapter runs as a service on the Gateway along with the agent service. To prevent the adapter processing exiting, use exception handling functions of `adapter_utils.py` .

3.3.4 Loop Structure

Most adapters need single-threaded execution. If the command-callback utility is used, make sure to run `get-commands` at frequent intervals to poll commands from the agent. For an example, see the sample adapter.

Note: You can use multi-threading/actor-frameworks as well.

4 Running Campaigns using Agent SDK

This chapter details the prerequisites and steps to run over-the-air (OTA) updates on a Gateway, using the Agent SDK.

Campaign services use the following properties from the **IoTCAgent**:

- `commandFetchIntervalSeconds` : The **IoTCAgent** makes periodic `get-command` requests to the micro services for every `commandFetchIntervalSeconds` expiry.
- You can configure the property value through the **Device Template** tab in the Console.

By default, the **IoTCAgent** runs with the following property values:

```
commandFetchIntervalSeconds=30

manifestExecution=ENABLE
```

When you start the **IoTC Agent** with default properties, it requests for command instructions from the Server by calling the `get-command` every 30 seconds.

Note: For each lifecycle phase, the **IoTCAgent** receives a command from the Server to perform the download, execute, and activate operations.

4.1 Running a Campaign using Default Properties

Perform the following steps to run an OTA update for the **IoTCAgent** using default properties.

- Using the `package-cli` tool, perform the following steps:
  1. Create an IoTC Package. For more information, see **Using Package Management CLI to Register Multiple Devices** chapter in the INFER™ User Guide.
  2. Upload the IoTC Package to the repository. Alternatively, use the Console to upload to the repository. For more information, see **Uploading the IoTC Package** chapter in the INFER™ User Guide.
- Enroll devices.
  1. Create a campaign using a distribution select query and the packages that you uploaded.
  2. Start the campaign.

The **IoTCAgent** auto-polls the command instructions every 30 seconds. The campaign states flow from `INITIALIZED` to `COMPLETED` after a series of get-commands calls to the Campaign Server.

4.2 Running a Campaign in On-Demand Mode

Perform the following steps to run an OTA update for the **IoTCAgent** in the On-Demand mode, that is, with the `commandFetchIntervalSeconds` property is set to `0` . This property value is defined in the device template.

1. In the specification file, set the value of the `headlessExecution` execution property to `false` .
2. Using the `package-cli` tool, perform the following steps:
   i. Create an IoTC Package.

ii. Upload the IoTC Package to the repository. Alternatively, use the INFER™ Console to upload to the repository.

3. Set the value of the `commandFetchIntervalSeconds` to `0` when creating the device template.

```
commandFetchIntervalSeconds = 0
```

4. Enroll the device.
5. Create a campaign using a distribution select query and the packages that you uploaded while creating the campaign.
6. Start the campaign.

The **IoTCAgent** invokes the `get-commands` when initiated from the `DefaultClient` binary.

The sample workflow below outlines the different states of the Gateway during an OTA update. The state of the Gateway is `INSTANTIATED` when the OTA campaign starts.

4.2.1 Sample Workflow

1. Invoke the get-commands to call from the `DefaultClient` or an Agent SDK extension. The state of the Gateway changes to `INVENTORY_UP_TO_DATE` .
2. Invoke the get-commands to call from the `DefaultClient` or an Agent SDK extension. The state of the Gateway changes to `WAITING_FOR_*_APPROVAL` .

   In the `WAITING_FOR_*_APPROVAL` state, schedule the next state. For example:

```
DefaultClient schedule --type=download --
campaignid=<campaign id>

DefaultClient schedule --type=download --
campaignid=<campaign id> --start-time=0 --end-time=0

DefaultClient schedule --type=download --
campaignid=<campaign id> --start-time=5000 --end-time=80000
```

Based on the campaign scheduled time, the state of the device changes from `SCHEDULED_DOWNLOAD` to `WAITING_FOR_DOWNLOAD` .

3. Invoke the get-commands to call from the `DefaultClient` or an Agent SDK extension. The Gateway starts downloading the package and the state of the device changes from `DOWNLOADING` to `DOWNLOAD_COMPLETE` .
4. Invoke the get-commands to call from the `DefaultClient` or the Agent SDK extension. The state of the Gateway changes to `WAITING_FOR_EXECUTION_APPROVAL` .

   Here, you can schedule a start and end time for running the campaign using the following command:

```
DefaultClient schedule --
type=<download|execution|activation> --
campaignid=<campaign Id> [--start-time=<start time window> --
end-time=<end time window>]
```

For example:

```
DefaultClient schedule --type=execution --
campaignid=<campaign id>
```

```
DefaultClient schedule --type=execution --
campaignid=<campaign id> --start-time=0 --end-time=0

DefaultClient schedule --type=execution --
campaignid=<campaign id> --start-time=5000 --end-time=80000
```

Based on the campaign scheduled time, the state of the device changes from `SCHEDULED_EXECUTION` to `WAITING_TO_EXECUTE` .

Here, you can schedule a start and end time for activating the campaign using the following command:

```
DefaultClient schedule --
type=<download|execution|activation> --
campaignid=<campaign Id> [--start-time=<start time window> --
end-time=<end time window>]

DefaultClient schedule --type=activation --
campaignid=<campaign id>

DefaultClient schedule --type=activation --
campaignid=<campaign id> --start-time=0 --end-time=0

DefaultClient schedule --type=activation --
campaignid=<campaign id> --start-time=5000 --end-time=80000
```

Based on the campaign scheduled time, the state of the device changes from `SCHEDULED_ACTIVATION` to `WAITING_TO_ACTIVATE` .

Note: Contact your Device Administrator or Campaign Administrator if the state of the Gateway changes to one of the following states: - DOWNLOAD_FAILED - EXECUTION_FAILED - ACTIVATION_FAILED

4.3 Running a Campaign in Headless Mode

This section lists the prerequisites for running a campaign for the **IoTCAgent** in Headless Mode.

- Run the **IoTCAgent** with the `manifestExecution` property set to `ENABLE` :

```
manifestExecution=ENABLE
```

On any campaign, the `get-commands` call ensures that the OTA updates are auto-delivered to the **IoTCAgent**. The `get-commands` calls from the **IoTCAgent** listens to the Campaign commands and the campaign downloads, executes, and activates updates.

4.3.1 Monitoring Campaign Progress

To monitor the progress of a campaign on the gateway, set the `agentLogLevel` to `6` in the iotc-agent.cfg file. You can then monitor the system logs to view the progress of the campaign using tools such as `journalctl -u` or `iotc-agent -f` .

4.4 Approving the OTA Update Phases

Depending on the **IoTCAgent** configuration and the package property for headless execution, there are check points in the device or gateway that may require an approval for the campaign to run.

You can configure your OEM or SI application to use these checkpoints to schedule a maintenance window for updates, or for approving the campaign to run the updates.

You can monitor the device or gateway's campaign progress from the **Campaigns** tab in the Console. To view the progress of the campaign, select the campaign from the list and click the **Devices** tab.

Note: The default interval for the **IoTCAgent** to fetch new commands from the Server is 30 seconds. You can change the interval value through the Device Templates settings in the Console.

Use the following commands to configure the campaign execution settings using the IoTCAgent SDK or the IoTCAgent CLI:

- After the campaign reaches the **Waiting for Download Approval** state:

```
DefaultClient schedule --type=download --
campaignid=<campaign Id>
```

Note: Copy the campaign ID from the Campaigns page of the Console.

- After the campaign reaches the **Waiting For Execution Approval** state:

```
DefaultClient schedule --type=execution --campaignid=<campaign Id>
```

- After the campaign reaches the **Waiting For Activation Approval** state:

```
DefaultClient schedule --type=activation --campaignid=<campaign Id>
```

5 Writing an Adapter using C SDK

This section provides information about writing an adapter using the Agent's C SDK.

The C SDK enables various other device-specific Adapters to interact with the Agent running at the Edge.

5.1 DefaultClient in IoTCAgent Package

The **IoTCAgent** package contains a directory that carries the source code of the `DefaultClient` binary file and a `makefile` to build your client. You can modify this source code according to your requirement.

The **IoTCAgent** package also contains a wrapper script to run the `DefaultClient` . The iotc-agent/example/directory contains the following files:

- clientDefaultClient.c
- DefaultClient.h
- DefaultClientDaemon.c
- base64.c
- Makefile

5.1.1 Send Metrics API Example

The following client program demonstrates the use of the Send Metrics API:

```
/* ***************************************************
 * Copyright (C) 2019 SmartHub, Inc. All rights reserved.
 * -- SmartHub Confidential
 * ***************************************************/

/**
 * @file ExampleMetric.c
 * @brief This file contains simple example code to demonstrate use of
 * iotc-agent-sdk send metrics API.
 *
 */

#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <unistd.h>
#include <sys/sysinfo.h>

#include "iotcAgent.h"

/* This sample client's application id */
#define TEST_CLIENT_ID "com.agent.test.metric"
#define MEM_USAGE "Memory-Usage"
/* timeout for waiting response from agent (in milliseconds) */
#define CLIENT_TIMEOUT  30000


/**
 * Return the current time in number of milliseconds since UNIX
```

```
 * epoch time.
 * @return A uint64_t that represents the current time.
 */
static uint64_t GetTimeStampMs(void)
{
   struct timeval timeVal = {.tv_sec = 0, .tv_usec = 0};
   uint64_t timeStamp;

   gettimeofday(&timeVal, NULL);
   //use milliseconds. Make sure the constants are unsigned long
   //long so that the computation does not overflow.
   timeStamp = (timeVal.tv_sec * 1000ULL) + (timeVal.tv_usec / 1000ULL);

   return timeStamp;
}


int main(int argc, char *argv[])
{
   IotcSession *session;
   IotcApplicationId clientAppId;
   struct sysinfo si;
   double memUsage;
   IotcMetric *memMetric;
   IotcGetResponse getResponse;
   int status;

   if (argc != 2) {
      printf("Usage: %s <deviceId>\n", argv[0]);
      return 1;
   }

   strncpy(clientAppId.id, TEST_CLIENT_ID, sizeof clientAppId.id);

   /* Initialize a session with the iotc-agent sdk */
   session = Iotc_Init(&clientAppId);
   if (session == NULL) {
      printf("Iotc_Init() failed");
      return -1;
   }

   /* Create a memory metric object */
   memMetric = malloc(sizeof (*memMetric) + sizeof (IotcDoubleValue));
   strncpy(memMetric->deviceId.id, argv[1],
           sizeof memMetric->deviceId.id);
   memMetric->deviceId.id[sizeof memMetric->deviceId.id - 1] = '\0';
   strncpy(memMetric->name, MEM_USAGE, sizeof memMetric->name);
   memMetric->name[sizeof memMetric->name - 1] = '\0';
   memMetric->type = IOTC_METRIC_DOUBLE;

   while (1) {
      if (sysinfo(&si) < 0) {
         printf("Error reading sysinfo\n");
         break;
      }

      /* Get the memory metric data */
```

```
      memUsage = (double) (si.totalram - si.freeram) *
       (double) 100 / (double) si.totalram;
      printf("mem: total=%ld free=%ld\n", si.totalram, si.freeram);
      memMetric->doubles[0].ts = GetTimeStampMs();
      memMetric->doubles[0].value = memUsage;
      /* Send the collected metric data */
      Iotc_SendMetric(session, memMetric);
      /* Get response for the send metric data */
      status = Iotc_GetResponseByType
      (session, IOTC_SEND_METRIC, CLIENT_TIMEOUT, &getResponse);
      if (status == -1) {
         fprintf(stderr, "Failed receiving send metric response\n");
      }

      Iotc_FreeGetResponse(&getResponse);
      sleep(5); // for 5 seconds
   }

   free(memMetric);
   /* Close the session with the agent */
   Iotc_Close(session);
   return 0;
}
```

5.2 Data Structures

5.2.1 IotcApplicationId

`IotcApplicationId` represents the application identifier.

Application identifier is any string with a maximum length of `IOTC_APP_ID_SIZE - 1` . It is used to identify an application uniquely during an exchange of data between the edge application and the server side application. Use the reverse domain name notation, such as `ai.smarthub.iotc.agent`

Data Fields

- Actual characters of the identifiers:

`char id[IOTC_APP_ID_SIZE]`

5.2.2 IotcAllowedMetricInfo

Stores the allowed metric information.

- Name of the metric:

`char metricName[IOTC_METRIC_NAME_SIZE]`

- Type of metric:

`IotcMetricType metricType`

5.2.3 IotcAllowedMetricSet

Set of allowed metrics of a device.

Data Fields

- Size of the allowed metric set:

`size_t size`

- Set of allowed metrics:

`IotcAllowedMetricInfo * allowedMetricInfo`

5.2.4 IotcBooleanValue

IotcBooleanValue represents the boolean type metric data point.

Data Fields

- `time_t ts`
- `unsigned char value`

5.2.5 IotcCampaignCallbacks

IotcCampaignCallbacks represents a collection of campaign callback functions.

Data Fields

- Update inventory info callback function:

`IotcUpdateInventoryInfoCb* IotcCampaignCallbacks::inventoryInfoCb`

- Pre-download callback function:

IotcCampaignPreDownloadCb * IotcCampaignCallbacks::preDownloadCb

- Pre-execution callback function:

IotcCampaignPreExecutionCb * IotcCampaignCallbacks::preExecutionCb

- Execute callback function:

`IotcCampaignExecuteCb * IotcCampaignCallbacks::executeCb`

- Pre-activate callback function:

`IotcCampaignPreActivationCb * IotcCampaignCallbacks::preActivationCb`

- Activate callback function:

IotcCampaignActivateCb * IotcCampaignCallbacks::activateCb

- Download progress callback function:

IotcCampaignDownloadProgressCb * IotcCampaignCallbacks::downloadProgressCb

- State change callback function:

IotcCampaignStateChangeCb * IotcCampaignCallbacks::stateChangeCb

5.2.6 IotcCampaignId

`IotcCampaignIdIotcCampaignId` represents the campaign identifier. Campaign identifier is any string with a maximum length of `IOTC_UUID_SIZE - 1`. It is provided by the Server as a response to an agent API or server API, or through campaign callbacks.

A campaign identifier could be in the GUID format such as `123e4567-e89b-12d3-a456-426655440000`, or any string such as `5b1656704cedfd000626bcaa`.

Data Fields

- Actual characters of identifiers:

`char id [IOTC_UUID_SIZE]`

5.2.7 IotcCampaignScheduleTimeWindow

`IotcCampaignScheduleTimeWindow` represents the campaign time window.

Contains the begin and end date and time for scheduling in the timestamp format expressed in epoch time. For example:

```
beginTime - Stamp : 1534855979, endTimeStamp : 1534856979
```

Data Fields

- Beginning timestamp for the time window:

```
time_t IotcCampaignScheduleTimeWindow::beginTimeStamp
```

- Ending timestamp for the time window:

```
time_t IotcCampaignScheduleTimeWindow::endTimeStamp
```

5.2.8 IotcClientConfig

`IotcClientConfig` represents client configuration SDKs.

Contains the begin/end date and time for scheduling in the timestamp format expressed in epoch time. For example:

```
beginTime - Stamp : 1534855979, endTimeStamp : 1534856979
```

Data Fields

```
IotcApplicationId appId
```

```
IotcClientLogLevel logLevel
```

5.2.9 IotcCommand

Command structure to hold details about a command message received from the server.

Data Fields

- Friendly name of the command:

```
char IotcCommand::name[IOTC_NAME_MAX_SIZE]
```

- Command identifier generated by the server:

```
char IotcCommand::id[IOTC_UUID_SIZE]
```

- Device identifier for the targeted device. This is an optional field:

```
IotcDeviceId IotcCommand::deviceId
```

- Absolute path to the executable. This is an optional field:

```
char IotcCommand::execPath[IOTC_PATH_MAX]
```

- Number of command arguments:

```
size_t IotcCommand::numArgs
```

- List of arguments for the command:

```
IotcCommandArg * args
```

5.2.10 IotcCommandArg

Command argument structure.

Data Fields

- The value type:

`IotcCommandArgValueType IotcCommandArg::type`

- Name of the argument:

`char IotcCommandArg::name[IOTC_NAME_MAX_SIZE]`

- Device identifier for the targeted device. This is an optional field:

`IotcDeviceId IotcCommand::deviceId`

- Number of items in the value array:

`size_t IotcCommandArg::numValues`

- Value array of the argument:

union { int64_t * intValues double * doubleValues char ** strValues };

5.2.11 IotcCommandResponse

Command response to hold the response received for a command.

Data Fields

- Error message to be sent to the server:

`char IotcCommandResponse::message[IOTC_PAYLOAD_MAX_SIZE]`

5.2.12 IotcDevice

(Deprecated) Represents a device entity.

Data Fields

- `IotcDeviceId deviceId`
- `IotcDeviceType type`

5.2.13 IotcDeviceDetails

Represents the device details.

Data Fields

- Name of the device:

`char IotcDeviceDetails::name[IOTC_NAME_MAX_SIZE]`

- Name of the device template:

`char IotcDeviceDetails::deviceTemplate[IOTC_NAME_MAX_SIZE]`

- Organization ID for the Device:

`char IotcDeviceDetails::deviceOrgId[IOTC_UUID_SIZE]`

5.2.14 IotcDeviceId

IotcDeviceId represents the device identifier.

Data Fields

`char id [IOTC_UUID_SIZE]`

5.2.15 IotcDeviceData

IotcDeviceData represents a device entity.

Data Fields

- `IotcDeviceId deviceId`
- `IotcTemplateId templateId`
- `IotcDeviceId parentId`
- `IotcDeviceId parentGatewayId`
- `IotcDeviceType type`
- `IotcEnrollmentState enrollmentState`
- `char deviceName [IOTC_NAME_MAX_SIZE]`
- `char templateName [IOTC_NAME_MAX_SIZE]`
- `IotcKeyValueSet systemProperties`
- `IotcKeyValueSet customProperties`
- `IotcAllowedMetricSet allowedMetrics`

5.2.16 IotcDeviceSet

Represents the set of devices.

Data Fields

- `IotcDevice*device`
- `size_t used`
- `size_t size`

5.2.17 IotcDeviceDataSet

5.2.18 IotcDoubleValue

IotcDoubleValue represents the float type metric data point.

Data Fields

- `time_t ts`
- `double value`

5.2.19 IotcEnrollmentCredentials

IotcEnrollmentCredentials represents the enrollment credentials.

Data Fields

- authToken contains the credentials required by the enrollment provider:

`char IotcEnrollmentCredentials::authToken[IOTC_PAYLOAD_MAX_SIZE]`

5.2.20 IotcEnrollmentData

IotcEnrollmentData represents the enrollment data. Enrollment data contains the type of enrollment and the required data for the enrollment.

Data Fields

- `parentId` is the device ID of the gateway device that the device connects to. For the root gateway device, the parent ID must be empty:

`IotcDeviceId IotcEnrollmentData::parentId`

- `deviceId` must be set for the IOTC_PRE_REGISTERED type:

`IotcDeviceId IotcEnrollmentData::deviceId`

- `deviceDetails` must be set for the IOTC_NOT_REGISTERED type:

`IotcDeviceDetails IotcEnrollmentData::deviceDetails`

5.2.21 IotcEnrollmentRequest

`IotcEnrollmentRequest` represents the enrollment request structure.

Data Fields

`IotcEnrollmentData IotcEnrollmentRequest::data`

- Data contains the required enrollment data:

```
union
{
    IotcEnrollmentCredentials enrollmentCredentials
    IotcUserCredentials userCredentials
}
```

`IotcEnrollmentCredentials IotcEnrollmentRequest::enrollmentCredentials`

- `enrollmentCredentials` must be set for the IOTC_PRE_REGISTERED type.

`IotcUserCredentials IotcEnrollmentRequest::userCredentials`

- userCredentials must be set for the IOTC_NOT_REGISTERED type.

5.2.22 IotcEnrollmentResponse

`IotcEnrollmentResponse` represents the enrollment response.

Data Fields

`IotcDeviceId deviceId`

`IotcDeviceId parentId`

5.2.23 IotcGetResponse

`IotcGetResponse` represents the GetResponse sent from the agent to the SDK.

Data Fields

- `uint64_t messageId`
- `IotcGetResponseMsgType type`
- `void * response`

5.2.24 IotcInt64Value

`IotcInt64Value` represents the integer type metric data point.

Data Fields

- `time_t ts`
- `int64_t value`

5.2.25 IotcKeyValue

IotcKeyValue represents a key value pair.

Data Fields

- `char key [IOTC_NAME_MAX_SIZE]`
- `char value [IOTC_VALUE_MAX_SIZE]`

5.2.26 IotcKeyValueSet

IotcKeyValueSet contains an array of device properties used. It represents the number of current keyValue set size and the capacity of the keyValue set.

Data Fields

- `IotcKeyValue*keyValue`
- `size_t used`
- `size_t size`

5.2.27 IotcMetric

IotcMetric represents the metric data point to be sent to the agent.

Data Fields

- `IotcMetricType type`
- `IotcDeviceId deviceId`
- `char name [IOTC_METRIC_NAME_SIZE]`

```
union
{
    struct IotcStringValue strings [0]
    struct IotcIntegerValue integers [0]
    struct IotcFloatValue floats [0]
    struct IotcBooleanValue bools [0]
};
```

5.2.28 IotcMetricResponse

IotcMetricResponse represents the metric response.

Data Fields

- Metric status of type:

`IotcMetricResponseStatus`

- Metric information that is received from agent:

`metric`

5.2.29 IotcNotificationDefinitionId

IotcNotificationDefinitionId represents the notification definition identifier.

Data Fields

- Holds the actual characters of the identifiers:

`char IotcNotificationDefinitionId::id[IOTC_UUID_SIZE]`

5.2.30 IotcNotificationResponse

IotcMetricIntvlResponse represents the notification response sent from the server to the client.

Data Fields

- Notification definition identifier:

`IotcNotificationDefinitionId IotcNotificationResponse::definitionId`

- Notification instance identifier:

`IotcNotificationId IotcNotificationResponse::notificationId`

- The payload byte size:

`char IotcNotificationResponse::payload[IOTC_PAYLOAD_MAX_SIZE]`

- Status flag:

`int IotcNotificationResponse::status`

5.2.31 IotcPackageId

`IotcPackageId` represents the package identifier string.The package identifier is a string with a maximum length of `IOTC_UUID_SIZE - 1` . It is in the GUID format such as, `98732222-1234-12d3-a456-426655440000` .

Data Fields

- Holds the actual characters of the identifiers:

`char IotcPackageId::id[IOTC_UUID_SIZE]`

5.2.32 IotcPropertySet

(Deprecated) IotcPropertySet represents information about the properties that are currently set.

Data Fields

- Contains an array of devices used:

`IotcDeviceId deviceId`

- Contains an array of properties used:

`IotcProperty * property`

- Represents the number properties currently set:

`size_t used`

- Represents the capacity of the property set:

`size_t size`

5.2.33 IotcSendNotificationRequest

IotcSendNotificationRequest represents the send notification request sent from the client to the server.

Data Fields

- Notification definition identifier:

`IotcNotificationDefinitionId IotcSendNotificationRequest::definitionId`

- Source of the request:

`IotcApplicationId IotcSendNotificationRequest::entityId`

- Array of key value pairs:

`IotcKeyValue* IotcSendNotificationRequest::keyValues`

- Number of key value pairs:

`size_t IotcSendNotificationRequest::numKeyValues`

5.2.34 IotcStringValue

IotcStringValue represents the string type metric data point.

Data Fields

- `time_t ts`
- `char value [IOTC_METRIC_STRING_VALUE_SIZE]`

5.2.35 IotcTemplateId

IotcTemplateId represents the template identifier.

Data Fields

- `char id [IOTC_UUID_SIZE]`

5.2.36 IotcUploadFileRequest

IotcUploadFileRequest represents the Upload File request sent from the agent to the server.

Data Fields

- File path at the local system to be uploaded:

`char IotcUploadFileRequest::srcFilePath[PATH_MAX]`

- Path with the destination file name appended at end of the URL to upload the file:

`char IotcUploadFileRequest::dstFilePath[PATH_MAX]`

5.2.37 IotcUserCredentials

IotcUserCredentials represents basic user credentials and organization domain name.

Data Fields

- `char username [IOTC_NAME_MAX_SIZE]`
- `char password [IOTC_NAME_MAX_SIZE]`
- `char orgDomainName [IOTC_NAME_MAX_SIZE]`

5.3 Functions

5.3.1 Iotc_AddMetricData

Adds metric data point in the metric data set.

API

```
int Iotc_AddMetricData (
    struct IotcMetricDataSet * metricDataSet,
    IotcMetric * metric )
```

Description

Sort the list based on device IDs. This ensures that all the metrics belonging to the same device are inserted from the device list when you fetch a device node.

Parameters

- Pointer to the metric data set:

`metricDataSet[IN,OUT]`

- Metric data to be sent to Agent:

`metric`

Returns

- `0` on success.
- `-1` on failure.

5.3.2 Iotc_AllocatePropertySet

(Deprecated) Allocates memory for the property set to hold the size and the number of properties.

API

`int Iotc_AllocatePropertySet ( IotcPropertySet * properties, size_t size )`

Parameters

- Pointer to the property set:

`in, out properties`

- Capacity of the property set in terms of number of properties:

`in size`

Returns

- `0` on success.
- `-1` on failure.

5.3.3 Iotc_AllocMetricDataSet

Allocates memory for metric data set to hold metrics data points.

API

```
struct IotcMetricDataSet* Iotc_AllocMetricDataSet ( void )
```

Returns

Pointer to allocated metric data set structure on success, and NULL on failure.

5.3.4 Iotc_CampaignScheduleActivation

Schedules the campaign for activation.

API

```
int Iotc_CampaignScheduleActivation (
IotcSession * session,
IotcCampaignId * campaignId,
IotcCampaignScheduleTimeWindow * timeWindow )
```

Description

Sends a request to the Server to schedule the campaign for activation. If the time window is empty, it indicates that client is ready to activate the campaign. Otherwise, the supplied time window is used by the server to schedule the campaign to activate for this gateway.

Parameters

- Connected session returned as part of Iotc_Init call:

`in session`

- Campaign ID of the campaign that is scheduled for activation:

`in campaignId`

- Schedule time window for the campaign to activate:

`in timeWindow`

Returns

`0` on success.

5.3.5 Iotc_CampaignScheduleDownload

Schedules the campaign for download.

API

```
int Iotc_CampaignScheduleDownload (
    IotcSession * session,
    IotcCampaignId * campaignId,
    IotcCampaignScheduleTimeWindow * timeWindow )
```

Description

Sends a request to the server to schedule the campaign for download. If the time window is empty, it indicates that the client is ready for downloading the campaign. Otherwise, the supplied time window is used by the server to schedule download of the campaign for this gateway.

Parameters

- Connected session returned as part of Iotc_Init call:

`in session`

- Campaign ID of the campaign that is scheduled for download:

`in campaignId`

- Schedule time window for the campaign to download:

`in timeWindow`

Returns

- `0` on success.

5.3.6 Iotc_CampaignScheduleExecution

Schedules the campaign for execution.

API

```
int Iotc_CampaignScheduleDownload (
    IotcSession * session,
    IotcCampaignId * campaignId,
    IotcCampaignScheduleTimeWindow * timeWindow )
```

Description

Sends a request to the Server to schedule the campaign for running. If the time window is empty, it indicates that client is ready to run the campaign. Otherwise, the supplied time window is used by the server to schedule the campaign to run for this gateway.

Parameters

- Connected session returned as part of Iotc_Init call:

`in session`

- Campaign ID of the campaign that is scheduled for running:

`in campaignId`

- Schedule time window for the campaign to run:

`in timeWindow`

Returns

- `0` on success.

5.3.7 Iotc_CampaignSetExecutionProgress

Updates the execution progress of the campaign.

API

```
int Iotc_CampaignSetExecutionProgress (
    IotcSession * session,
    IotcCampaignId * campaignId,
    const char * progress )
```

Description

Sends a request to the server to update the execution progress of the campaign.

Parameters

- Connected session returned as part of Iotc_Init call:

`in session`

- Campaign identifier:

`in campaignId`

- Progress string to be sent to the server:

`in progress`

Returns

- `0` on success.

5.3.8 IotcCommandCb

The command callback function type.

API

```
typedef int IotcCommandCb(
    const IotcCommand *command,
    IotcCommandResponse *response,
    void *context)
```

Parameters

- Command received from the server:

`in`

- Response data to be sent to the server for the command:

`out`

- context is the opaque context data that is supplied by the client during command callback registration:

`in`

Returns

- `0` on success.
- `-1` on error.

5.3.9 Iotc_Close

Closes the communication channel with the IoTCAgent.

API

```
void Iotc_Close ( IotcSession * session )
```

Parameters

- Connected session returned as part of Iotc_Init call:

`in session`

5.3.10 Iotc_DeletePropertySet

(Deprecated) Frees the memory used by the property set.

API

```
void Iotc_DeletePropertySet ( IotcPropertySet * properties )
```

Parameters

- Pointer to the property set:

`in, out properties`

5.3.11 Iotc_DeleteProperties

Frees the memory used by the properties.

API

```
void Iotc_DeleteProperties ( IotcKeyValueSet * properties )
```

Parameters

- Pointer to the property set:

`in, out properties`

5.3.12 Iotc_Enroll

Enrolls the gateway and generates a Gateway Identifier.

API

```
int Iotc_Enroll (
    IotcSession * iotcSession,
    IotcEnrollmentRequest * enrollmentRequest )
```

Parameters

- Connected session returned as part of Iotc_Init call:

`in session`

- Pointer to the enroll request object:

`in requestData`

- Contains the enroll response received for the request:

`out responseData`

Returns

- `0` on success.

5.3.13 Iotc_FreeMetricDataSet

Frees the metric data points in the metric data set. Mandatory if Iotc_AllocMetricDataSet() is called.

API

```
void Iotc_FreeMetricDataSet ( struct IotcMetricDataSet * metricDataSet )
```

Parameters

- Pointer to the metric data set:

`metricDataSet[IN]`

Returns

- `0` on success.
- `-1` on failure.

5.3.14 Iotc_GetCertificateIds

This function retrieves all the certificate Ids of the associated devices.

API

```
int Iotc_GetCertificateIds(
    IotcSession *iotcSession,
    const IotcDeviceId *deviceId )
```

Description

Get all the certificate Ids associated with a device.

Parameters

- Current IotcSession to be used:

`iotcSession`

- Pointer to the device identifier of the device.

`deviceId`

Returns

- `0` on success.
- `-1` on failure.

5.3.15 Iotc_GetCertificateIdsByIssuer

This function retrieves all the certificate Ids associated with devices matching the specified issuer.

API

```
int Iotc_GetCertificateIdsByIssuer(
    IotcSession *iotcSession,
    const IotcDeviceId *deviceId,
    const char *issuer )
```

Description

Get certificate ids associated with a device whose issuer entry matches the specified issuer.

Parameters

- Current IotcSession to be used:

`iotcSession`

- Pointer to the device identifier of the device:

`deviceId`

- NULL terminated UTF-8 string. This string is matched against all the entries in a certificate issuer(for example, Common Name, Organization Name). If one of the entries completely matches this string, the certificate Id is included in the response.

`issuer`

Returns

- `0` on success.
- `-1` on failure.

5.3.16 Iotc_GetCertificateIdsBySubject

This function retrieves all the certificate Ids of the associated devices matching the subject.

API

```
int Iotc_GetCertificateIdsBySubject(
    IotcSession *iotcSession,
    const IotcDeviceId *deviceId,
    const char *subject)
```

Description

Get certificate ids associated with a device whose subject entry matches the appropriate subject.

Parameters

- Current IotcSession to be used:

`iotcSession`

- Pointer to the device identifier of the device:

`deviceId`

- NULL terminated UTF-8 string. This string is matched against all the entries in a certificate subject(for example, Common Name, Organization Name). If one of the entries fully matches this string, the certificate id is included in the response.

`subject`

Returns

- `0` on success.
- `-1` on failure.

5.3.17 Iotc_GetCertificate

This function retrieves the certificate associated with a device.

API

```
int Iotc_GetCertificate(
    IotcSession *iotcSession,
    const IotcDeviceId *deviceId,
    const IotcCertificateId *certId,
    const char *filePath)
```

Description

Get private key as a PEM file. The PEM file is written to a specified file path.

Parameters

- Current IotcSession to be used:

`iotcSession`

- Pointer to the device identifier of the device:

`deviceId`

- Pointer to IotcCertificateId:

`certId`

- Path to which the certificate is written. Must be a full path.

`filePath`

Returns

- `0` on success.
- `-1` on failure.

5.3.18 Iotc_GetPrivateKey

This function retrieves the private key.

API

```
int Iotc_GetPrivateKey(
    IotcSession *iotcSession,
    const IotcDeviceId *deviceId,
    const IotcCertificateId *certId,
    const char *filePath)
```

Description

Get private key as a PEM file. The PEM file is written to a specified file path.

Parameters

- Current IotcSession to be used:

`iotcSession`

- Pointer to the device identifier of the device:

`deviceId`

- Pointer to IotcCertificateId:

`certId`

- The path to which the certificate is written. Must be a full path:

`filePath`

Returns

- `0` on success.
- `-1` on failure.

5.3.19 Iotc_GetCommands

Gets commands available for this gateway device from the Server.

API

```
int Iotc_GetData (
    IotcSession * session,
    IotcGetDataRequest * requestData )
```

Description

Sends a request to the server to check if there are any commands available for this gateway. If the retrieved command data is for the agent, then the agent processes it. Any command data that is not for the agent is returned to the client as response data.

Parameters

- Connected session returned as part of Iotc_Init call:

`in session`

Returns

- `0` on success.
- `-1` on failure.

5.3.20 Iotc_GetCustomProperties

(Deprecated) Retrieves the custom properties of the gateway device.

API

```
int Iotc_GetCustomProperties (
    IotcSession * iotcSession,
    IotcDeviceId * deviceId )
```

Parameters

- Connected session returned as part of Iotc_Init call:

`in iotcSession`

- Device identifier:

`in deviceId`

Returns

- `0` on success.
- `-1` on failure.

5.3.21 Iotc_FreeGetResponse

A general function to free internal resources used in a IotcGetResponse message.

API

`void Iotc_FreeGetResponse ( IotcGetResponse * getResponse )`

Parameters

- Pointer of the IotcGetResponse message:

`getResponse`

5.3.22 Iotc_GetResponseByType

Processes response messages and returns only the desired message based on the message type provided.

API

```
int Iotc_GetResponseByType (
    IotcSession * session,
    IotcGetResponseMsgType requestedType,
    int timeout,
    IotcGetResponse * getResponse )
```

Parameters

- Current IotcSession to be used:

`in session`

- Desired type of response message to be obtained:

`in requestedType`

- Duration to wait for a response from the agent, in milliseconds:

`in timeout`

- IotcGetResponse pointer for holding the result.

`out getResponse`

Returns

Returns `-1` on failure. This value comes from the status of the response message or from a communication error. To handle differences between these failures, check the returned message type.

5.3.23 Iotc_Sync

This function synchronizes device related information such as default properties with the server.

API

`int Iotc_Sync ( IotcSession * iotcSession )`

5.3.24 Iotc_GetDevices

(Deprecated) This function retrieves all the connected devices for a device using the ID and type.

API

`int Iotc_GetDevices`

Description

The devices would be returned with a pointer to IotcDeviceSet in the IotcGetResponse with IOTC_GET_DEVICES as the response message type.

Parameters

- Current IotcSession to be used:

`iotcSession`

- Device ID for which the connected device IDs must be retrieved:

`parentId`

5.3.25 Iotc_GetDevicesData

This function retrieves details such as device ID, device type, device name, template ID, template name, enrollment state, parent ID, parent gateway ID, system properties, custom properties and allowed metrics of all the connected devices of a device.

API

```
int Iotc_GetDevicesData (
    IotcSession * iotcSession,
    IotcDeviceId * parentId )
```

Description

The devices are returned with a pointer to IotcDeviceDataSet in the IotcGetResponse with IOTC_GET_DEVICES_DATA as the response message type.

Parameters

- Current IotcSession to be used:

`iotcSession`

- Device ID for which the connected device IDs must be retrieved:

`parentId`

5.3.26 Iotc_GetMessageId

Returns the messageId corresponding to the latest API invoked by the client. Invoke Iotc_GetMessageId before calling the next API.

API

`uint64_t Iotc_GetMessageId ( IotcSession * iotcSession )`

Parameters

- Connected session returned as part of Iotc_Init call:

`in iotcSession`

5.3.27 Iotc_GetResponse

Gets response from the agent.

API

```
int Iotc_GetResponse (
    IotcSession * iotcSession,
    IotcGetResponse * response )
```

Description

Returns GetData response to the Client. If the retrieved command data is for the agent, then agent processes it. Any command data that is not for the agent is returned to the client as response data.

Parameters

- Connected session returned as part of Iotc_Init call:

`in session`

- Contains the response data received for the request:

`out responseData`

Returns

- `0` on success.
- `-1` on failure.

5.3.28 Iotc_GetSessionSockfd

(Deprecated) Retrieves the system properties of the gateway device.

API

```
int Iotc_GetSystemProperties (
    IotcSession * iotcSession,
    IotcDeviceId * deviceId )
```

Parameters

- Connected session returned as part of Iotc_Init call:

`in iotcSession`

- Device identifier.

`in deviceId`

Returns

- `0` on success.
- `-1` on failure.

5.3.29 IotcSession* Iotc_Init

Initializes the communication channel with the IoTCAgent.

API

`IotcSession* Iotc_Init ( IotcApplicationId * applicationId )`

Parameters

- Application identifier of the invoking client:

`in applicationId`

Returns

Pointer to the session object on success or NULL on failure.

5.3.30 IotcSession*Iotc_InitWithConfig

Initializes a communication channel with the IoTCAgent using the supplied configuration.

API

`IotcSession*Iotc_InitWithConfig ( IotcClientConfig * config )`

Parameters

- Pointer to the client configuration object.

`in config`

Returns

Pointer to the session object on success or NULL on failure.

5.3.31 Iotc_InsertProperty

(Deprecated) Adds a property to the property set.

API

```
int Iotc_InsertProperty (
    IotcPropertySet * properties,
    IotcProperty * property )
```

Parameters

- Pointer to the property set:

`in, out properties`

- Pointer to the property to be added:

`in property`

Returns

- `0` on success.
- `-1` on failure.

5.3.32 Iotc_InsertProperties

Adds a property to the property set.

API

```
int Iotc_InsertProperties (
    IotcKeyValueSet * properties,
    IotcKeyValue * property )
```

Parameters

- Pointer to the property set:

`in, out properties`

- Pointer to the property to be added:

`in property`

Returns

- `0` on success.
- `-1` on failure.

5.3.33 Iotc_RegisterCampaignCallbacks

Registers campaign callback functions.

API

```
int Iotc_RegisterCampaignCallbacks (
    IotcSession * session,
    IotcCampaignCallbacks * cbs,
    void * userData )
```

Parameters

- Connected session returned as part of Iotc_Init call:

`in session`

- Campaign callback functions collection that is invoked by the IoTCAgent during state change, download progress, download, and so on:

`in cbs`

- User context data that is returned when invoking callback functions.

`in userData`

Returns

`0` on success.

5.3.34 Iotc_RegisterCommandCallback

The command callback registration function.

API

```
int Iotc_RegisterCommandCallback (
    IotcSession * iotcSession,
    IotcCommandCb * cb,
    void * context )
```

Parameters

- Current IoTCSession to be used:

`in IotcSession`

- Command callback function:

`in cb`

- Pointer to any context data that must be supplied when cb is called:

`in context`

Returns

- `0` on success.
- `-1` on error.

5.3.35 Iotc_SendMetric

Requests the agent to send a metric to the server.

API

`int Iotc_SendMetric ( IotcSession * iotcSession, IotcMetric * requestData )`

Parameters

- Connected session returned as part of Iotc_Init call:

`in iotcSession`

- Pointer to the send metric request data:

`in requestData`

5.3.36 Iotc_SendMetricSet

Sends multiple metrics to the Agent to be sent to the server.

API

```
int Iotc_SendMetricSet (
    IotcSession * iotcSession,
    struct IotcMetricDataSet * metricDataSet )
```

Description

Use following helper functions to add metrics data:

- `Iotc_MetricDataSet *Iotc_AllocMetricDataSet(void);`
- `Iotc_AddMetricData(IotcMetricDataSet *metricDataSet, IotcMetric *metric);`

5.3.37 Iotc_SendNotification

Sends the notification request to the server.

API

`IotcSession * session, IotcSendNotificationRequest * requestData`

Parameters

- Connected session returned from Iotc_Init call:

`in session`

- Pointer to the notification request object:

`in requestData`

Returns

- `0` on success.

5.3.38 Iotc_SendPropertySet

Sends the property set to the server.

API

```
int Iotc_SendPropertySet (
    IotcSession * iotcSession,
    IotcKeyValueSet * properties,
    IotcDeviceId * deviceId )
```

Parameters

- Connected session returned from Iotc_Init call:

`in iotcSession`

- Pointer to the property set:

`in properties`

- Device identifier:

`in deviceId`

Returns

- `0` on success.
- `-1` on failure.

5.3.39 Iotc_UnEnroll

Requests to un-enroll a device.

API

```
int Iotc_UnEnroll ( IotcSession * iotcSession, IotcDeviceId * deviceId )
```

Description

Sends a request to the Server to un-enroll the device specified by deviceId. If the deviceId is empty, then the root gateway device is un-enrolled.

Parameters

- Connected session returned as part of Iotc_Init call:

`in session`

- Pointer to the device identifier of the device:

`in deviceId`

5.3.40 Iotc_UploadFile

Uploads the specified file to the server.

API

```
int Iotc_UploadFile (
    IotcSession * session,
    IotcUploadFileRequest * requestData )
```

Parameters

- Connected session returned as part of Iotc_Init call:

`in session`

- Pointer to the post data request object:

`in requestData`

Returns

- `0` on success.

5.4 Macro Definitions

This section lists the macros and their definitions for the Agent APIs.

- Maximum size of the UUID:

`#define IOTC_UUID_SIZE 37`

- Maximum size for a name string. Used in device names and in device property name-value pairs:

`#define IOTC_NAME_MAX_SIZE 256`

- Maximum size for a value string. Used in device property name-value pairs:

`#define IOTC_VALUE_MAX_SIZE 512`

- Maximum size of an application identifier:

`#define IOTC_APP_ID_SIZE 65`

- Maximum size for the payloads:

```
#define IOTC_PAYLOAD_MAX_SIZE 4096
```

- Maximum size of the metric name:

```
#define IOTC_METRIC_NAME_SIZE 64
```

- Maximum size of the metric string data point:

```
#define IOTC_METRIC_STRING_VALUE_SIZE 32
```

5.5 Enumeration Types

This section lists the enumeration types and their definitions for the Agent APIs.

5.5.1 enum IotcValType

Denotes the metric unit type.

```
{
    BOOLEAN,
    FLOAT,
    STRING,
    INTEGER
}
```

5.5.2 enum IotcCampaignState

Denotes the supported campaign states.

```
{
    IOTC_CAMPAIGN_INITIALIZED, IOTC_CAMPAIGN_INSTANTIATED,
    IOTC_CAMPAIGN_INVENTORY_UP_TO_DATE,
    IOTC_CAMPAIGN_INVENTORY_UPDATE_FAILURE,
    IOTC_CAMPAIGN_WAITING_FOR_DOWNLOAD_APPROVAL,
    IOTC_CAMPAIGN_SCHEDULED_DOWNLOAD, IOTC_CAMPAIGN_WAITING_FOR_DOWNLOAD,
    IOTC_CAMPAIGN_DOWNLOADING, IOTC_CAMPAIGN_DOWNLOAD_COMPLETE,
    IOTC_CAMPAIGN_DOWNLOAD_FAILED,
    IOTC_CAMPAIGN_WAITING_FOR_EXECUTION_APPROVAL,
    IOTC_CAMPAIGN_SCHEDULED_EXECUTION, IOTC_CAMPAIGN_WAITING_TO_EXECUTE,
    IOTC_CAMPAIGN_EXECUTING, IOTC_CAMPAIGN_EXECUTION_COMPLETE,
    IOTC_CAMPAIGN_EXECUTION_FAILED,
    IOTC_CAMPAIGN_WAITING_FOR_ACTIVATION_APPROVAL,
    IOTC_CAMPAIGN_SCHEDULED_ACTIVATION, IOTC_CAMPAIGN_WAITING_TO_ACTIVATE,
    IOTC_CAMPAIGN_ACTIVATING, IOTC_CAMPAIGN_ACTIVATION_COMPLETE,
    IOTC_CAMPAIGN_ACTIVATION_FAILED
}
```

5.5.3 enum IotcGetResponseMsgType

Denotes the supported response message types.

```
{
    IOTC_INVALID_RESPONSE,
    IOTC_NOTIFICATION_RESPONSE,
    IOTC_ENROLL_RESPONSE,
    IOTC_UNENROLL_RESPONSE,
    IOTC_CAMPAIGN_STATE_CHANGE,
    IOTC_SCHEDULE_RESPONSE,
    IOTC_SET_PROGRESS,
```

```
    IOTC_SEND_METRIC,
    IOTC_UPLOAD_FILE,
    IOTC_GET_COMMANDS_FINISHED,
    IOTC_REGISTER_CB,
    IOTC_SEND_PROPERTIES,
    IOTC_GET_SYSTEM_PROPERTIES,
    IOTC_GET_CUSTOM_PROPERTIES,
    IOTC_GET_DEVICES,
    IOTC_GET_DEVICES_DATA,
    IOTC_CLIENT_COMMAND,
    IOTC_ERROR_RESPONSE,
    IOTC_NO_RESPONSE
}
```

5.5.4 enum IotcEnrollmentType

Denotes the supported enrollment types.

```
{
    IOTC_PRE_REGISTERED,
    IOTC_NOT_REGISTERED
}
```

5.5.5 enum boolean

Denotes the boolean state.

```
{
    FALSE,
    TRUE
}
```

5.5.6 enum IotcMetricType

Denotes the metric unit type.

```
{
    IOTC_METRIC_ERROR,
    IOTC_METRIC_STRING,
    IOTC_METRIC_INTEGER,
    IOTC_METRIC_FLOAT,
    IOTC_METRIC_BOOLEAN,
    IOTC_METRIC_UNKNOWN
}
```

5.5.7 enum IotcMetricResponseStatus

Status of the metric response sent from the Agent SDK.

- IOTC_METRIC_FAILED: Metric failed to be stored at the agent or sent to the server.
- IOTC_METRIC_NOT_ALLOWED: The metric is not in the allowed list.
- IOTC_METRIC_STORED: The metric is successfully stored in the agent.
- IOTC_METRIC_SUCCESS: The metric is successfully sent to the server.

```
{
    IOTC_METRIC_SUCCESS,
```

```
    IOTC_METRIC_STORED,
    IOTC_METRIC_NOT_ALLOWED,
    IOTC_METRIC_FAILED
}
```

5.5.8 IotcClientLogLevel

Denotes the SDK client log levels.

```
{
    IOTC_LOG_EMERG = 0,
    IOTC_LOG_ALERT = 1,
    IOTC_LOG_CRIT = 2,
    IOTC_LOG_ERROR = 3,
    IOTC_LOG_WARN = 4,
    IOTC_LOG_NOTICE = 5,
    IOTC_LOG_INFO = 6,
    IOTC_LOG_DEBUG = 7
}
```

5.5.9 enum IotcEnrollmentState

Supported device states.

5.6 Writing a Client Application using IoTCAgent SDK

To write a client application using the IoTCAgent SDK, perform the following steps.

1. Define an identifier for the client application:

```
IotcApplicationId clientAppId;
strncpy(clientAppId.id, "com.myclient", sizeof clientAppId.id);
```

2. Establish a session between the client application and IoTC Agent:

```
IotcSession *session;
session = Iotc_Init(&clientAppId);
if (session == NULL) {
   // Handle failure
}
```

3. After establishing a session, the client can invoke other APIs to perform operations.

   Currently, the IoTCAgent API works in an asynchronous mode. When an API is invoked, a request is sent to the IoTCAgent and the API returns to the client. Now, the client invokes the Iotc_GetResponse() API to receive a response from the previously invoked API. For example:

```
/** Enrollment wrapper function */
static int
EnrollGateway(IotcSession *session,
      const char* templateName,
      const char* gatewayName,
      const char* username,
      const char* password)
{
   IotcEnrollmentRequest enrollmentRequest;
   IotcGetResponse getResponse;
   IotcEnrollmentResponse *resp;
```

```
    int status;

    enrollmentRequest.data.type = IOTC_NOT_REGISTERED;
    strncpy(enrollmentRequest.data.deviceDetails.deviceTemplate,
          templateName,
          sizeof enrollmentRequest.data.deviceDetails.deviceTemplate);
    enrollmentRequest.data.deviceDetails.deviceTemplate
       [sizeof enrollmentRequest.data.deviceDetails.deviceTemplate - 1] =
        ↪  '\0';
    strncpy(enrollmentRequest.data.deviceDetails.name, gatewayName,
          sizeof enrollmentRequest.data.deviceDetails.name);
    enrollmentRequest.data.deviceDetails.name
       [sizeof enrollmentRequest.data.deviceDetails.name - 1] = '\0';
    strncpy(enrollmentRequest.userCredentials.username,
          username,
          sizeof enrollmentRequest.userCredentials.username);
    enrollmentRequest.userCredentials.username
       [sizeof enrollmentRequest.userCredentials.username - 1] = '\0';
    strncpy(enrollmentRequest.userCredentials.password,
          password,
          sizeof enrollmentRequest.userCredentials.password);
    enrollmentRequest.userCredentials.password
       [sizeof enrollmentRequest.userCredentials.password - 1] = '\0';

    if (Iotc_Enroll(session, &enrollmentRequest) == -1) {
       fprintf(stderr, "Failed sending enroll request\n");
       return -1;
    }

    /* Invoke GetResponse by supplying type of response */
    status = Iotc_GetResponseByType(session, IOTC_ENROLL_RESPONSE,
                                    CLIENT_TIMEOUT, &getResponse);
    if (status == -1) {
       fprintf(stderr, "Enroll response failed for this client\n");
       return -1;
    }

    /* if the GeResponse succeeded, fetch the response */
    resp = getResponse.response;
    printf("Device Id: %s\nParent Device Id: %s\n",
          resp->deviceId.id, resp->parentId.id);
    printf("Status of enroll response: %d\n", status);

    /* Cleanup the memory used by the response object */
    Iotc_FreeGetResponse(&getResponse);
    return 0;
}
```

4. To disconnect a client from the IoTCAgent, invoke the following API:

```
Iotc_Close(session);
```

5.6.1 Sample MyClient Source Code

```
/* ************************************************
 * Copyright (C) 2019 SmartHub, Inc. All rights reserved.
 * -- SmartHub Confidential
```

```
 * ************************************************/

/**
 * @file MyClient.c
 * @brief This file contains simple example code to demonstrate use of
 * iotc-agent-sdk APIs.
 * This example show how to use Iotc_Enroll API. Users can invoke
 * other APIs in similar manner.
 * This file also offers a utility function to read responses for a
 * API request.
 *
 * note: This is a simple demo example code.
 */

#include <stdio.h>
#include <string.h>

#include "iotcAgent.h"

/* timeout for waiting response from agent (in milliseconds) */
#define CLIENT_TIMEOUT  30000

/** Enrollment wrapper function */
static int
EnrollGateway(IotcSession *session,
      const char* templateName,
      const char* gatewayName,
      const char* username,
      const char* password)
{
   IotcEnrollmentRequest enrollmentRequest;
   IotcGetResponse getResponse;
   IotcEnrollmentResponse *resp;
   int status;

   enrollmentRequest.data.type = IOTC_NOT_REGISTERED;
   strncpy(enrollmentRequest.data.deviceDetails.deviceTemplate,
         templateName,
         sizeof enrollmentRequest.data.deviceDetails.deviceTemplate);
   enrollmentRequest.data.deviceDetails.deviceTemplate
      [sizeof enrollmentRequest.data.deviceDetails.deviceTemplate - 1] = '\0';
   strncpy(enrollmentRequest.data.deviceDetails.name, gatewayName,
         sizeof enrollmentRequest.data.deviceDetails.name);
   enrollmentRequest.data.deviceDetails.name
      [sizeof enrollmentRequest.data.deviceDetails.name - 1] = '\0';
   strncpy(enrollmentRequest.userCredentials.username,
         username,
         sizeof enrollmentRequest.userCredentials.username);
   enrollmentRequest.userCredentials.username
      [sizeof enrollmentRequest.userCredentials.username - 1] = '\0';
   strncpy(enrollmentRequest.userCredentials.password,
         password,
         sizeof enrollmentRequest.userCredentials.password);
   enrollmentRequest.userCredentials.password
      [sizeof enrollmentRequest.userCredentials.password - 1] = '\0';

   if (Iotc_Enroll(session, &enrollmentRequest) == -1) {
```

```
      fprintf(stderr, "Failed sending enroll request\n");
      return -1;
   }

   /* Invoke GetResponse by supplying type of response */
   status = Iotc_GetResponseByType(session, IOTC_ENROLL_RESPONSE,
                                   CLIENT_TIMEOUT, &getResponse);
   if (status == -1) {
      fprintf(stderr, "Enroll response failed for this client\n");
      return -1;
   }

   /* if the GeResponse succeeded, fetch the response */
   resp = getResponse.response;
   printf("Device Id: %s\nParent Device Id: %s\n",
         resp->deviceId.id, resp->parentId.id);
   printf("Status of enroll response: %d\n", status);

   /* Cleanup the memory used by the response object */
   Iotc_FreeGetResponse(&getResponse);
   return 0;
}


int main(int argc, char *argv[])
{
   IotcSession *session;
   IotcApplicationId clientAppId;
   const char *usage = "<template name> <gateway name> <username> <password>";

   if (argc != 5) {
      fprintf(stderr, "Usage:\n %s %s\n", argv[0], usage);
      return 1;
   }

   strncpy(clientAppId.id, "com.myclient", sizeof clientAppId.id);
   session = Iotc_Init(&clientAppId);
   if (session == NULL) {
      /* Handle failure */
      fprintf(stderr, "Could not initialize a session with iotc-agent\n");
      return 1;
   }

   /* Invoke a iotc-agent sdk API
      Note password is consumed as command line parameter for
      keeping thus example program simple */
   if (EnrollGateway(session, argv[1], argv[2], argv[3], argv[4]) == -1) {
      fprintf(stderr, "Enrollment failed\n");
   }

   /* Close the session */
   Iotc_Close(session);
   return 0;
}
```

5.7 Building a Client that uses the IoTCAgent SDK

Use the following steps to build a client that uses the IoTCAgent SDK.

1. Extract the IoTCAgent SDK to a directory such as IOTC_DIR=/opt/iotc-sdk.
2. Compile the client application by entering the `include` directory and the libraries to link.

```
LD_LIBRARY_PATH=../lib gcc -o MyClient MyClient.c -I $IOTC_
DIR/include/ -L $IOTC_DIR/lib -liotc-agent-sdk
```

5.8 Running a Client that uses the IoTCAgent SDK

Clients using IoTC Agent SDK require the `iotc group` privilege or the `root` user privilege.

To run a client program with a non-root user privilege, you must include the `iotc group` in the supplemental groups and run the client program with the `iotc group` permission:

```
sudo usermod -a -G iotc $USER
sudo runuser $USER -G iotc  -m -c "LD_LIBRARY_PATH=
/opt/smarthub/iotc-agent/lib ./MyClient"
```

5.9 Working with DefaultClient

The IoTC Agent CLI is IoTC Agent's default client binary `DefaultClient` . On Windows, this tool is available as `DefaultClient.exe`

This tool provides a command-line interface (CLI) to perform IoTC Agent SDK operations. With the IoTC Agent CLI tool, you can build a client that operates with SmartHub INFER IoT Center using the IoTC Agent SDK. You can use the `DefaultClient` binary as a reference for building your client.

The IoTC Agent CLI provides multiple CLI options. Please run the following command to know more.

```
/opt/smarthub/iotc-agent/bin# ./DefaultClient help
```

Use the IoTC Agent CLI to perform operations such as enrolling a device and setting properties for a device quickly.

Note:

Declare the library path explicitly if you see error messages such as:

`error while loading shared libraries: libiotc-agent-sdk.so:`

`cannot open shared object file: No such file or directory` .

Run the following command:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/smarthub/iotc-agent/lib/
```

The IoTC Agent CLI is available in the bin directory of IoTC Agent:
`/opt/smarthub/iotc-agent/bin/DefaultClient`

For more information on DefaultClient, see **DefaultClient in IoTCAgent Package**.

5.10 Using DefaultClient Daemon

You can run the `DefaultClient` binary file as a daemon process in the background. In the daemon mode, `DefaultClient` connects to the IoTC Agent daemon and authorizes campaign call-backs automatically.

It also fetches commands from the server at regular intervals. When additional options are specified, `DefaultClient` gathers the default CPU and Memory Usage metrics from the Gateway device and sends them periodically. You can perform the following operations using the `DefaultClient` daemon:

- Start the `DefaultClient` daemon without sending the default metrics:

```
$ DefaultClient start-daemon
```

- Start the `DefaultClient` daemon with default metrics every 10 minutes:

```
$ DefaultClient start-daemon --device-id=<device_id> --interval=600
```

- Stop the `DefaultClient` daemon.

```
$ DefaultClient stop-daemon
```

Using the IoTC Agent connection, the `DefaultClient` daemon accepts requests from the following pipe files if necessary:

- /tmp/iotc-defclient/input for an input request.
- /tmp/iotc-defclient/output for an output request.

The following sample illustrates how to get system properties using the `DefaultClient` daemon:

```
$ echo "get-properties --device-id=13c425e1-873a-
43f0-a529-cb05289a8a40 --type=system" > /
tmp/iotc-defclient/input
$ cat /tmp/iotc-defclient/output
```

To see a sample workflow that shows how to get system properties using the `DefaultClient` daemon, see **Send Metrics API Example**.