SJS SOLUTIONS
IMPROVING CUSTOMER CARE

Optymyse

HOWTO: Use the Optymyse REST API

Contents

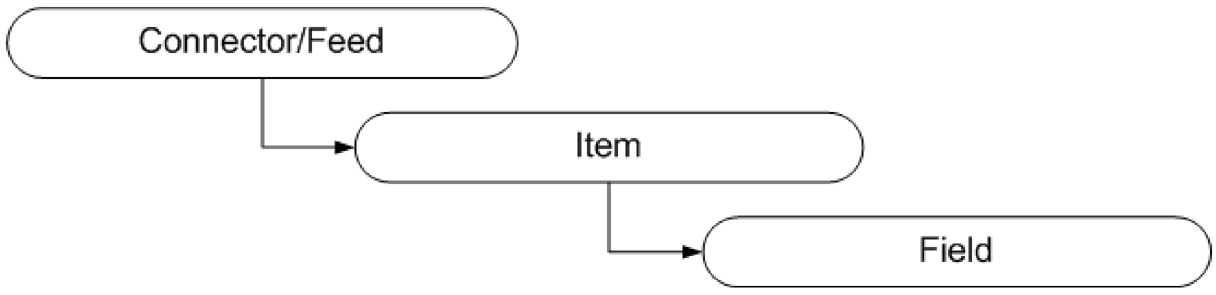
The Optymyse REST API has been designed to allow third-party applications to pass data into Optymyse in such a way that data can be added to pages from any data source, including customer databases and other third-party systems.

Optymyse accepts data from a FEED, which is implemented within a CONNECTOR. A CONNECTOR may provide many FEEDs. For example, the ShoreTel ECC Connector provides up to 3 feeds - Agents, Groups and DNIS.

Each FEED requires a Feed licence within Optymyse. There is no limit on the number of connectors, but Optymyse will refuse to register new feeds once the license limit has been met.

There are no limits to the number of data items (e.g. Agents), or the number of fields per item.

Data in Optymyse is presented to the user using a 3-tier selection:



Conceptually, the easiest way to imagine the required data is to format it in a table, thus:

Connector/Feed	Field1	Field2	Field3	Field4	...	Field_N
Item_1_id Item_1_Name	Data	Data	Data	Data	...	Data
Item_2_id Item_2_Name	Data	Data	Data	Data	...	Data
Item_3_id Item_3_Name	Data	Data	Data	Data	...	Data
...	...	...	...	...	...	...
Item_N_id Item_N_Name	Data	Data	Data	Data	...	Data

The Data Transmission Packet format is used to pass large amounts of data to Optymyse in one transmission.

For smaller feeds, items can be passed in individually. The same format applies to the item identifier - connector/feed, Item Name, Field Name - but you may individually name any item & field name as you required, within certain limitations described below.

1 Connectors

1.1 Is your connector registered?

The first action any connector must carry out is determining whether or not it is registered with Optymyse. To achieve this, simply call:

```
GET /DataPool/v1/Connector/$ConnectorID$
```

replacing $ConnectorID$ with the name of your connector.

> $ConnectorID$ should be unique! If you re-use another connector's name, any feeds you define will either over-write those of the other connector, or supplement them. Note that you can't use identical connector names to increase the number of feeds into Optymyse beyond the licence limit.

$ConnectorID$ should be a "dropdown friendly" description of your connector, e.g. "ShoreTelECC"; this will appear in the Optymyse UI whenever a field is selected.

GET /DataPool/v1/Connector/$ConnectorID$ will return a status of:

200 if the connector is already registered
404 if the connector is NOT registered or has been manually deleted.

If you receive status 200, the connector is registered and you do not need to take further action.

In addition to the code, the following data is returned by GET:

```
{
  "Name": "<your_connector_name>",
  "Detail": "<your_connector_description>",
  "State": "<your_connector_state>",
  "StatusCode": <your_connector_status_code>,
  "StatusDescription": "<your_connector_status_text>"
}
```

Status is a code indicating the last known status of your connector

Note: If you omit the $ConnectorID$ parameter, the system will return a list of registered connectors in the following data format:

```
{
  "Connectors": [
    {
      "Name": "<connector_name>",
      "Detail": "",
      "State": "<connector_state>",
      "StatusCode": <connector_statuscode>,
      "StatusDescription": ""
    }
  ]
}
```

Alternatively, it will return code 404 if no connectors are registered.

1.2 Registering a new Connector

If you receive a 404 code when checking the connector, you must register with Optymyse. This is achieved by simply calling:

```
POST /DataPool/v1/Connector
```

The following data items are required:

ID = the $ConnectorID$ value
Detail = An optional description of what your connector does

e.g.

```
{
    "ID": "MyConnector",
    "Detail": "Retrieve data from my database"
}
```

If POST succeeds, it returns a status of 201, and the URL to the new connector's details.

If the connector is already registered when POST is called, the status 409 is returned.

POST will return minVer and maxVer:

```
{
    "MinVer": <min_version>,
    "MaxVer": <max_version>
}
```

These are reserved for future use.

1.3 Deleting a Connector

If you wish to delete a connector (e.g. to uninstall it), call:

```
DELETE /DataPool/v1/Connector/$ConnectorID$
```

$ConnectorID$ should be replaced with your connector's name

The service will respond with code 204 if the delete succeeds, or 404 (not found) if the connector name was not recognised. Any feeds registered to this connector will automatically be deleted along with the connector details.

The user may also delete a connector in the Optymyse UI.

2 Feeds

2.1 Is your Feed registered?

Once the connector is registered with Optymyse, you can add the feeds. This step should be carried out when your connector starts; unless it's a new registration (in which case, you know you need to add the feeds).

To verify a feed, simply call:

```
GET /DataPool/v1/Connector/$ConnectorID$/Feed/$FeedID$
```

$ConnectorID$ should be replaced with your connector's name
$FeedID$ should be replaced with the feed name you wish to check.

Note: $FeedID$ can be omitted, in which case the API will return an array of feed names registered to the supplied Connector ID.

If the feed exists, the call will return a status code of 200 OK and the following JSON (or XML if you are using XML format):

```
{
    "ConnectorName": "<name_of_connector>",
    "FeedName": "<your_feed_name>",
    "Definition": "<field_list_in_encoded_JSON_format>"
}
```

If the feed does not exist, the call will return 404; use the section below to register the new feed.

2.2 Registering a new Feed / Updating an existing feed.

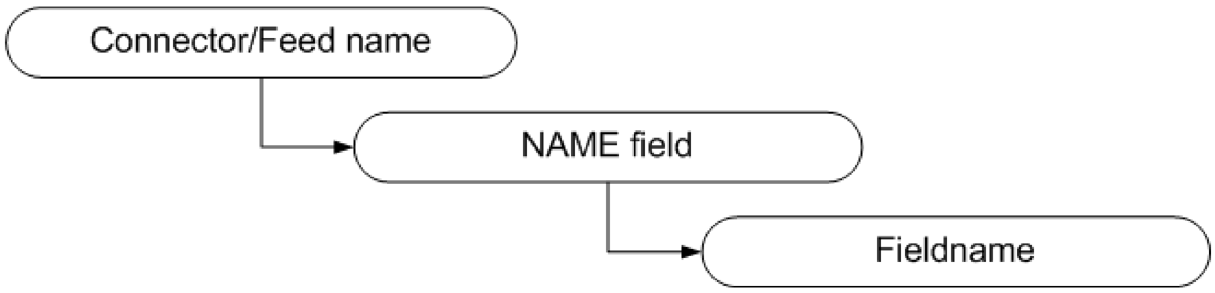
Each Feed can be considered as a contract with Optymyse to supply certain data items. To that end, each feed needs to tell Optymyse what fields it provides, this information is used to construct the field selection drop-downs within the Page Designer UI; and, of course, to actually populate the data on a page.

A Contract, therefore, consists of a list of fields. Each field name within a feed MUST be unique (and is case insensitive - so ID and id are identical). Each field is considered to be an object, with the following properties:

- ID: Integer, unique within this feed
- DataType: String, e.g. INTEGER DATE, STRING, PERCENTAGE, CURRENCY
- AggregateType: String, e.g. KEY, NAME, SUM, AVG, MAX, MIN
- AggregateFunction: String, a mathematical function telling Optymyse how to aggregate this field
- DeaggregateFunction: String, a mathematical function telling Optymyse how to un-aggregate this field

One field must be nominated the KEY field. The values in this field will be used to uniquely identify the item when sending data. e.g. AgentID, GroupID, MyDataBasePrimaryKeyID. A KEY field is nominated by setting the AggregateType field to KEY.

One field must be nominated the "NAME" field. The values in this field will be used to populate the middle drop-down in our field selector:



The NAME field is identified by setting the AggregateType value to NAME.

Currently, no other values for AggregateType are used, any other value in this field is silently ignored. However, you may supply any one of SUM, AVG, MAX, MIN, FIRST, LAST; these may be used in future.

⚠ If you supply more than one key field, only the first field will be used! Do NOT supply more than one key field. There is no guarantee that the field order will be the same between the serialisation and de-serialisation of the list, so if you have multiple KEY fields, you may end up with unexpected results!

Also, do not supply more than one NAME field - again, the system will use the first name field it finds as the name; as the field order is not guaranteed, this may cause unexplained results!

The JSON format of a field is as follows:

```
"<fieldName>": {
    "ID":<fieldID>,
    "DataType":"<dataType>",
    "AggregateType":"<aggregateType>",
    "AggregateFunction":"<aggregateFunction>",
    "DeaggregateFunction":"<deaggregateFunction>"
}
```

AggregateFunction and DeaggregateFunction are reserved for future use, and should be sent as blank strings.

All fields are then wrapped into a single object. For example, a typical collection of fields, ready for transmission to Optymyse, might look like this:

```
{
    "agentID":{"ID":1,"DataType":"KEY","AggregateType":"","Aggrega
teFunction":"","DeaggregateFunction":""}
    "agentName":{"ID":2,"DataType":"NAME","AggregateType":"","Aggr
egateFunction":"","DeaggregateFunction":""}
    "salesTarget":{"ID":3,"DataType":"INT","AggregateType":"","Agg
regateFunction":"","DeaggregateFunction":""}
    "salesToday":{"ID":4,"DataType":"INT","AggregateType":"","Aggr
egateFunction":"","DeaggregateFunction":""}
    "salesValue":{"ID":5,"DataType":"CURRENCY","AggregateType":"",
"AggregateFunction":"","DeaggregateFunction":""}
}
```

⚠ Fieldnames are case sensitive, but must be unique case insensitively. Therefore agentId is not the same as AGENTID, but you can only have either agentId OR AGENTID, not both, in one field set.

To register the feed, call:

```
POST /DataPool/v1/Connector/$Connector$/Feed
```

$ConnectorID$ should be replaced with your connector's name

The body of your POST request should contain:

```
{
    "FeedID":"<feed_name>",
    "Fields":<field_list>
}
```

where <feed_name> is the unique (within this connector) name of the feed, and field_list is the JSON formatted list of fields as described above.

The POST request will return a response code of 201 if it works, and the following JSON:

```
{
    "Fingerprint":"<datacontract_fingerprint>",
    "MaxVer":MaxVersion,
    "MinVer":MinVersion
}
```

where <datacontract_fingerprint> is a hash representing the configuration of the contract; this is used internally by Optymyse to ensure that the data you send into the feed matches the contract.

MaxVersion and MinVersion are integers, and are reserved for future use.

If the connector is not yet registered, a response code of 409 is returned. In the event of receiving a 409 code, try re-registering the connector. If you get another 409 error, there was a problem registering your connector name; further detail may appear in the error message.

If you've run out of feed licences, a response code of 503 is returned. In this instance, you cannot register the feed and should give up. If your connector provides multiple feeds, and some of them have registered, then the rest of the connector should be allowed to continue running.

2.3 Deleting a Feed

If you wish to delete a feed at any time; simply call:

```
DELETE /DataPool/v1/Connector/$ConnectorID$/Feed/$FeedID$
```

$ConnectorID$ should be replaced with your connector's name
$FeedID$ should be replaced with the feed name you wish to delete.

The service will respond with code 204 if the delete succeeds, or 404 (not found) if the connector/feed combination was not found

⚠ If you've already deleted the connector, the feeds are deleted automatically and don't need manually removing.

3 Connector Configuration

3.1 Downloading your connector's configuration

Optymyse offers you the option of saving much of your connector's configuration details within its database. At the time of writing, this is implemented as simple text box, which the user can edit to alter settings. Altering this configuration & saving changes will trigger a configuration changed status command to the connector.

To load your configuration from the server, call the following:

```
GET /DataPool/V1/Connector/$ConnectorID$/Configuration
```

The $ConnectorID$ is your connector name.

If a configuration block is saved for your connector, the system will respond with code 200, and will output the configuration via JSON (if your connector's configuration is also JSON, this will be encoded):

```
{
    "Configuration":"<your_configuration_string>"
}
```

If you receive code 404 (not found), your connector should send a default or initial configuration to Optymyse using the Put option, below.

3.2 Uploading your connector's configuration

To store your connector's configuration (e.g. if you were sent a 404 code when loading, you can upload a default or template configuration) using the following:

```
PUT /DataPool/V1/Connector/$ConnectorID$/Configuration
```

Again, $ConnectorID$ must be your connector's unique name.

You must also pass the configuration string in the same format as above. If your connector's configuration string is JSON, don't forget to escape it!

4 Pulse

4.1 Keeping the dream alive

The REST API does not poll attached connectors, therefore connectors must poll Optymyse; this is done by calling the PULSE code:

```
PUT /DataPool/V1/Connector/$ConnectorID$/Pulse
```

As usual, $ConnectorID$ is your connector's name.

PULSE allows you to send a status code & text to Optymyse. This code/text can be used by Optymyse to alert the user of any issues it may be experiencing - e.g. loss of connection to its data source, etc. The format for the data is as follows:

```
{
    "Status": <your_connector_status>,
    "StatusDescription": "<connector_status_text>"
}
```

Status is an integer value. Currently it is not interpreted by Optymyse, and is reserved for future use. Please send the code 200 to indicate your service is running.

StatusDescription is a string value. Whatever you send here, will be displayed in the Optymyse UI next to your connector's record. Keep it clean!

Pulse, if successful, will return a simple text value with code 200. An invalid ConnectorID will return code 404. Malformed JSON data will result in a code 400.

The JSON returned on success is as follows:

```
{
    "Reply": "<reply_text>"
}
```

Defined responses are:

- "ok" - Do nothing, all is well.
- "configuration-changed" - Your configuration has been changed. Handle it.

Pulse will continue to return "configuration-changed" until you call GET /DataPool/V1 /Connector/$ConnectorID$/Configuration to retrieve your new configuration record; once you have retrieved your new configuration, pulse will return to sending "ok".

Additional commands may be added to this list in the future.

5 Sending Data

Finally, we get to the whole purpose of your connector! Sending data to Optymyse.

There are two ways to achieve this: Single data item, or data packet.

For small amounts of data (less than 20 items or so), the single item transfer is probably the one to use. For larger quantities, it's more efficient to package an entire block of data in one go.

5.1 Sending individual data items

To transfer data on a per-item-field basis, we need to call the following URL with a POST command:

```
POST /DataPool/v1/Item
```

POST requires the following data:

```
{
    "ConnectorID": "<your_connector_name>",
    "FeedName": "<this_feed_name>",
    "ItemID": <item_key_value>,
    "ItemName": "<item_name_value>",
    "Field": "<field_name>",
    "Value": "<display_value>",
    "UnformattedValue": "<evaluation_value>",
    "Order": <sort_order>
}
```

Connector ID, FeedName have already been covered.

- ItemID is the value which matches your KEY field (e.g. AgentID, GroupID, MyDatabaseTablePrimaryKey).
- ItemName is the value which matches your NAME field (e.g. AgentName, GroupName, etc.)
- Field is the field name you wish to set, and must match one of the field names you configured when registering the feed. e.g. CallsOffered, TimeInAUX, NumSales, etc.
- Value is the value you want to see on the page, with all formatting (e.g. percentage signs, currency symbols, etc.) applied. Optymyse does not apply any formatting of its own to these values.

- UnformattedValue - should be the bare numeric equivalent (where applicable) of the data item you wish to display.

Typical values for unformatted data are shown below:

e.g. 100% should be sent as 100, 55% as 55, 43.2% as 43.2
e.g. £1200 = 1200; $56.65 = 56.65
e.g. Fred = Fred - send strings as basic strings. String values can only be compared with "=".

Time values should be converted into seconds:

e.g. 0:30 = 30, 2:15 = 135, 1:00:00 = 3600

The purpose of UnformattedValue is to allow the threshold rules to operate in a consistent manner no matter what the format of the underlying data.

Finally Order; this is reserved for future use and should be set to zero.

A succesful POST will return code 200 and the following response:

```
{
    "result":"ok"
}
```

If the item cannot be found (because Connector, Feed, ItemID or ItemName are incorrect), you will receive code 404.

If a value already exists for this item, POST will update it. If there is no value, it will create it for you.

5.2 Sending Bulk Data

Transferring bulk data requires the data to be pre-packaged into an object structure, GZIPed, Base64 encoded, and POSTED to the following URL:

`POST /DataPool/v1/Packet/{ID}`

where ID is the Connector's name.

POST also requires a packet of data to be sent, in the data property:

```
{
    "data": "<gzip_base64_data_packet>"
}
```

See below for the definition of a data packet.

If the POST is successful, the request will send back code 200 and the following:

```
{
    "result": "ok"
}
```

If unsuccessful, you may still get a code 200, but the result will differ as follows:

- Result = "warnings|message": The data packet did not match the recorded contract. Check the "format" field has all the required fields present, and that each entity has all necessary fields, and no extras. Additional feedback will be supplied after the pipe.
- Result = "fail|message": The data packet failed for an unknown or other reason (e.g. connector not found). The message will give more information.

Additional result codes may be returned in future.

5.2.1 Data Packet Definition

```
{  "<Feed_1_Name>": {
     "Format":"<comma_separated_field_list>",
     "Entities": {
       "<Entity_1_Name>": {
         "Fields": {
           "<field_1_name>":{"Value":"<value>","UnformattedValue":
"<value>"},
           "<field_2_name>":{"Value":"<value>","UnformattedValue":
"<value>"},
           "<field_3_name>":{"Value":"<value>","UnformattedValue":
"<value>"},
           ...
           "<field_N_name>":{"Value":"<value>","UnformattedValue":
"<value>"},
         }
       },
       "<Entity_2_Name>": {
         "Fields": {
           "<field_1_name>":{"Value":"<value>","UnformattedValue":
"<value>"},
           "<field_2_name>":{"Value":"<value>","UnformattedValue":
"<value>"},
           "<field_3_name>":{"Value":"<value>","UnformattedValue":
"<value>"},
           ...
           "<field_N_name>":{"Value":"<value>","UnformattedValue":
"<value>"},
        }
      },
      ...
      "<Entity_N_Name>": {
        "Fields": {
          "<field_1_name>":{"Value":"<value>","UnformattedValue":"
<value>"},
          "<field_2_name>":{"Value":"<value>","UnformattedValue":"
<value>"},
          "<field_3_name>":{"Value":"<value>","UnformattedValue":"
<value>"},
          ...
          "<field_N_name>":{"Value":"<value>","UnformattedValue":"
<value>"},
        }
      }
    },
    "<Feed_2_Name>": {
      "Format":"<comma_separated_field_list>",
      "Entities": {
      ...
    },
    ...
    "<Feed_N_Name>": {
      "Format":"<comma_separated_field_list>",
      "Entities": {
      ...
    }
}
```

For example, here's a simplified group feed:

```
{
 "Groups":{
   "Format":"GroupId,GroupName,queuedCalls,currentMaxQueTime",
   "Entities": {
     "Marketing":
       {"Fields":{
         "GroupId":{"Value":"11","UnformattedValue":"11"},
         "GroupName":{"Value":"Marketing","UnformattedValue":"Mark
eting"},
         "queuedCalls":{"Value":"0","UnformattedValue":"0"},
         "currentMaxQueTime":{"Value":"00:00","UnformattedValue":"
0"}
       }
     },
     "Events":{
       "Fields":{
         "GroupId":{"Value":"5","UnformattedValue":"5"},
         "GroupName":{"Value":"Events","UnformattedValue":"Events"
},
         "queuedCalls":{"Value":"0","UnformattedValue":"0"},
         "currentMaxQueTime":{"Value":"00:00","UnformattedValue":"
0"}
       }
     },
     "Freephone":{
       "Fields":{
         "GroupId":{"Value":"12","UnformattedValue":"12"},
         "GroupName":{"Value":"Freephone","UnformattedValue":"Free
phone"},
         "queuedCalls":{"Value":"0","UnformattedValue":"0"},
         "currentMaxQueTime":{"Value":"00:00","UnformattedValue":"
0"}
       }
     },
   }
}
```

5.3 Related articles

- Page: HOWTO: Configure ShoreTel ECC versions 6-10