Edge Storage Quick Start

This document is a quick tour of getting data into the Edge Storage component using the OSisoft Message Format (OMF), and then retrieving the data using the Sequential Data Store (SDS) API. Both OMF data ingress and SDS data retrieval are accomplished using REST APIs. This tour assumes the Edge System has been installed, and is accessible via a REST API using the default installed port (5590). This tour will use curl, a commonly available tool on both Windows and Linux, and use command line commands. The same operations can be used with any programming language or tool that supports making REST calls. In addition data retrieval steps (GET commands) can be accomplished using a browser if one is available on the device.

Creating an OMF Type

The first step in OMF data ingress is to create a OMF type that describes the format of the data to be stored in a container. In our example the data to be written is a timestamp and a numeric value, so the OMF JSON describing the type is:

```
[{
    "id": "MyCustomType",
    "classification": "dynamic",
    "type": "object",
    "properties": {
        "Timestamp": {
            "type": "string",
            "format": "date-time",
            "isindex": true
        },
        "Value": {
            "type": "number",
            "format": "float32"
        }
    }
}]
```

The value is indexed by a timestamp and the numeric value that will be stored is a 32 bit floating point value. In order to create the OMF type in the Edge Storage, the JSON should be stored as a file with the name OmfCreateType.json and the following curl script run:

```
curl -i -d "@OmfCreateType.json" -H "Content-Type: application/json" -H
"producertoken: x " -H "omfversion: 1.1" -H "action: create" -H "messageformat:
json" -H "messagetype: type" -X POST
http://localhost:5590/api/v1/tenants/default/namespaces/default/omf/
```

When this command completes successfully, an SDS type with the same name will have been created on the server. Any number of containers can write data to the same type, as long as they use a timestamp as an index and a 32 bit floating point value. This only needs to be done the first time you send using a custom application, but it does not cause an error if you resend the same definition at a later time.

Creating an OMF Container

The next step in writing OMF data is to create a container. As with an OMF Type, this only needs to be done once before sending data events, and resending the same definition repeatedly does not cause an error.

```
[{
    "id": "MyCustomContainer",
    "typeid": "MyCustomType"
}]
```

This container references the OMF Type that was created in the last step, and an error will occur if the OMF Type does not exist when the container is created. In order to create the OMF container in the Edge Storage, the JSON should be stored as a file with the name OmfCreateContainer.json and the following curl script run:

```
curl -i -d "@OmfCreateContainer.json" -H "Content-Type: application/json" -H
"producertoken: x " -H "omfversion: 1.1" -H "action: create" -H "messageformat:
json" -H "messagetype: container" -X POST
http://localhost:5590/api/v1/tenants/default/namespaces/default/omf/
```

When this command completes successfully, an SDS stream will have been created to store data defined by the type.

Writing Data Events to the OMF Container

Now that the OMF Type and Container have been created, we can write data using OMF:

```
[{
    "containerid": "MyCustomContainer",
    "values": [{
            "Timestamp": "2019-07-16T15:18:24.9870136Z",
            "Value": 12345.6789
        },
        {
            "Timestamp": "2019-07-16T15:18:25.9870136Z",
            "Value": 12346.6789
        }
    ]
}]
```

This example includes two data events that will be stored in OMF Container/SDS Stream that was created in the previous steps. It is generally a best practice to batch OMF values when writing them for the best performance. In order to write the data in the Edge Storage, the JSON should be stored as a file with the name OmfCreateDataEvents.json and the following curl script run:

```
curl -i -d "@OmfCreateDataEvents.json" -H "Content-Type: application/json" -H
"producertoken: x " -H "omfversion: 1.1" -H "action: create" -H "messageformat:
json" -H "messagetype: data" -X POST
http://localhost:5590/api/v1/tenants/default/namespaces/default/omf/
```

When this command completes successfully, two values will have been written to the SDS stream.

Reading Last Data written using OMF

In order to read the data back from the server that has been written, you can use the SDS REST API. Here is an example curl script that reads back the last value entered:

```
curl
http://localhost:5590/api/v1/tenants/default/namespaces/default/streams/MyCustomCo
ntainer/Data/Last
```

When run this GET command returns the last value written:

```
{"Timestamp":"2019-07-16T15:18:25.9870136Z","Value":12346.6787}
```

Reading a range of data events written using OMF

In order to read the data back from the server that has been written, you can use the SDS REST API. Here is an example curl script that reads back a time range of values that have been written:

```
curl
"http://localhost:5590/api/v1/tenants/default/namespaces/default/streams/MyCustomC
ontainer/Data?startIndex=2019-07-08T13:00:00Z&count=100"
```

```
[{"Timestamp":"2019-07-16T15:18:24.9870136Z","Value":12345.6787},
{"Timestamp":"2019-07-16T15:18:25.9870136Z","Value":12346.6787}]
```


Both values that were entered were returned - up to 100 values after the specified timestamp would be returned.

For more information on SDS APIs: https://ocs-docs.osisoft.com/Documentation/SequentialDataStore/Data_Store_and_SDS.html