OMF Documentation

Release 1.1-alpha1

OSIsoft, LLC

Oct 03, 2018

Contents

CHAPTER 1

v1.1

CHAPTER 2

Overview

The OSIsoft Message Format (OMF) defines a set of message headers and bodies that can be used to generate messages for ingestion into a compliant back-end system.

OMF can be used to develop data acquisition applications on platforms and in languages for which there are no supported OSIsoft libraries.

OMF itself does not define or depend on any particular binary message protocol (HTTP, AMQP, Kafka, etc.) It is instead based on an abstract message type, where a message consists of a set of key / value pairs, called the headers, and a binary payload, called the body. OMF messages can thus be constructed using any message protocol that defines headers and bodies. For up to date information on the specific binary protocols currently supported by OSIsoft systems, consult the OSIsoft Cloud Services and PI Connector Relay documentation.

Please note that anything defined in this document, such as keywords and enumerated values, is treated as case insensitive. Case sensitivity for all user specified values is determined by the rules of the backend system.

CHAPTER 3

Contents

3.1 What's New

Version 1.1 introduces the following incremental changes from version 1.0:

- *Nullable type properties* which can be declared by specifying an array of accepted values including the type and `null`.
- A `uom` keyword for *Type properties* representing the optional unit of measure.
- An `indexes` keyword for *Container messages* representing the optional array of Type Property ids to be used as secondary indexes for the Container.

3.2 Headers

The following headers are supported in version 1.1 of the message specification. Headers may be added or removed over time. Backwards compatibility is achievable through the `omfversion` header.

Name	Value
`producertoken`	A unique token used to identify and authorize a given OMF producer. Consult the OSIsoft Cloud Services or PI Connector Relay documentation for further information.
`messagetype`	Describes the type of message contained in the message body. One of: `type`, `container`, or `data`. See *Message Types*.
`messageformat`	Describes the data serialization format employed in the message body. Currently limited to `json`.
`omfversion`	Specifies the version of the OSIsoft Message Format used in the message. The version for the current specification is 1.1.
`action`	Optional: One of: `create`, `update`, or `delete`. Describes the action to be performed using the data in the message body. If omitted, `create` is assumed.
`compression`	Optional: The compression algorithm used to compress the message body. Currently limited to `gzip`. If not specified, the message body is assumed to be uncompressed.

3.3 Message Types

OMF messages fall into three categories: Type, Container, and Data messages. Within each category, three types of actions can be specified: Create, Update, and Delete.

Each message bulks individual JSON objects into a JSON array. Within a given array, all objects must be of the same message type: Type, Container, or Data.

To achieve optimal throughput, bulking and compression of messages is recommended.

3.4 Type Messages

Types are defined using the JSON Schema specification (http://json-schema.org/). For details on supported property data types and formats, see *Type Properties and Formats*.

Types can be created, updated, or deleted. Updating a Type without updating the Type's version will result in a failure. Once a Type is deleted, any operations on Containers or Data using that Type will fail.

The body of a Type message consists of an array of objects. The following keywords are used to define a Type:

Name	Value
`id`	Unique identifier of the Type.
`classification`	One of `dynamic` or `static`.
`version`	Optional version of the Type. The version must be of format x.x.x.x, where x must be an integer greater than or equal to 0. If omitted version 1.0.0.0 is assumed.
`name`	Optional friendly name for the Type.
`description`	Optional description for the Type.
`tags`	Optional array of strings to tag the Type.
`metadata`	Optional key-value pairs associated with the Type.
`properties`	Key-value pairs defining the properties of the Type.
`type`	Inherited from JSON Schema. Must be set to `object`.

The `id` cannot begin with the character sequence __. This is reserved for predefined Types. One currently supported predefined Type is __Link.

A `static` classification represents metadata describing a device being observed and should be used to capture data that is descriptive and relatively unchanging. A `dynamic` classification represents observed or calculated measurements taken from a device.

3.4.1 Link Type

A Link is a pre-defined type with the typeid __Link. It has the following properties.

Name	Value
`source`	An object representing the source of the link
`target`	An object representing the target of the link

Each `source` and `target` object has the following keywords:

Name	Value
typeid	Optional id of the type. If omitted, containerid is expected.
containerid	Optional id of the container. If omitted, typeid is expected.
typeversion	Optional version of the type to be linked to or from. The version must be of format x.x.x.x, where x must be an integer greater than or equal to 0. If omitted version 1.0.0.0 is assumed.
index	Index of the data. If typeid is specified, index is mandatory. If containerid is specified it is optional.

If typeid and index are specified, the link is for a particular non-container value.

If containerid is specified with an index, the link is for a particular container value.

If containerid is specified without an index, the link is for a container.

3.4.2 Type Properties and Formats

The following keywords are used to define a Type Property:

Name	Value
type	Required type of the Type Property which must match a type listed in the Supported Formats table below.
format	Optional format of the Type Propety type that, if specified, must be from the table below.
isindex	At least one Type Property must be designated as the index by supplying the isindex keyword with a value of true. The designated isindex property is used to uniquely identify discrete Data objects so that they can be updated or deleted after their initial creation. For a compound index, the order of index properties within the message determines the order within the index.
isname	One Type Property may be optionally designated as the name by supplying the isname keyword with a value of true. Because the index must be unique across all Data objects, the isname keyword allows for multiple distinct Data objects to share a common name.
name	Optional friendly name for the Type Property.
description	Optional description for the Type Property.
uom	Optional unit of measure for the Type Property.

OMF supports the array, boolean, integer, number, and string data types defined by JSON Schema. Timestamps, dictionaries, and bit length-specific numeric properties may also be defined by setting the format keyword, as described in the Supported Formats table below.

Supported Formats

Type	Format	Default Value	Description
array		null	An array of objects. The required `items` keyword defines the type of the objects in the array.
boolean		false	A value of either "true" or "false".
integer	int64	0	64-bit integer.
integer	int32 (default)	0	32-bit integer.
integer	int16	0	16-bit integer.
integer	uint64	0	64-bit unsigned integer.
integer	uint32	0	32-bit unsigned integer.
integer	uint16	0	16-bit unsigned integer.
number	float64	0	64-bit floating point.
number	float32 (default)	0	32-bit floating point.
number	float16	0	16-bit floating point.
object	dictionary	null	A dictionary of objects, indexed by a string key. The `additionalProperties` keyword defines the dictionary's value type.
string		null	A string.
string	date-time	0001-01-01T00:00:00Z	A string representation of a timestamp, formatted as YYYY-MM-DDThh:mm:ssZ, with optional subsecond precision.

Nullable type properties are supported by specifying an array of accepted values including the type and `null`. The type and `null` may appear in any order in the array. For example:

```
"Nullable Int64": {"type": ["integer", "null"], "format": "int64"}
```

```
"Nullable DateTime": {"type": ["null", "string"], "format": "date-time"}
```

```
"Nullable Boolean": {"type": ["boolean", "null"]}
```

3.4.3 Type Example

Headers

```
producertoken = b7CNvN36cq
omfversion = 1.1
messagetype = type
messageformat = json
action = create
```

Body

```
[{
        "id": "Plant",
        "version": "1.0.0.0",
        "type": "object",
        "classification": "static",
        "properties": {
                "PlantId": {
                        "type": "string",
                        "isindex": true
                },
                "PlantName": {
                        "type": "string",
                        "isname": true
                },
                "Address": {
                        "type": "string"
                },
                "Contact": {
                        "type": "string"
                }
        }
}, {
        "id": "Tank",
        "version": "1.0.0.0",
        "type": "object",
        "classification": "static",
        "properties": {
                "Name": {
                        "type": "string",
                        "isindex": true
                },
                "Serial": {
                        "type": "string"
                },
                "Model": {
                        "type": "string"
                }
        }
}, {
        "id": "TankMeasurement",
        "version": "1.0.0.0",
        "type": "object",
        "classification": "dynamic",
        "properties": {
                "Time": {
                        "format": "date-time",
                        "type": "string",
                        "isindex": true
                },
                "Pressure": {
                        "type": "number",
                        "name": "Tank Pressure",
                        "description": "Tank Pressure in Pa",
                        "uom": "pascal"
                },
                "Temperature": {
                        "type": "number",
```

(continues on next page)

(continued from previous page)

```
                        "name": "Tank Temperature",
                        "description": "Tank Temperature in K",
                        "uom": "K"
                }
        }
}, {
        "id": "Tank2",
        "version": "1.0.0.0",
        "type": "object",
        "classification": "static",
        "properties": {
                "Name": {
                        "type": "string",
                        "isindex": true
                },
                "Dimensions": {
                        "type": "object",
                        "format": "dictionary",
                        "additionalProperties": {
                                "type": "number",
                                "format": "float64"
                        }
                },
                "Maintenance Schedule": {
                        "type": "array",
                        "items": {
                                "type": "string",
                                "format": "date-time"
                        }
                },
                "Location": {
                        "type": "object",
                        "properties": {
                                "Latitude": {
                                        "type": "number"
                                },
                                "Longitude": {
                                        "type": "number"
                                }
                        }
                }
        }
}, {
        "id": "Tank2",
        "version": "1.0.0.0",
        "type": "object",
        "classification": "static",
        "properties": {
                "Name": {
                        "type": "string",
                        "isindex": true
                },
                "Dimensions": {
                        "type": "object",
                        "format": "dictionary",
                        "additionalProperties": {
                                "type": "number",
```

(continues on next page)

(continued from previous page)

```
                                "format": "float64"
                        }
                },
                "Maintenance Schedule": {
                        "type": "array",
                        "items": {
                                "type": "string",
                                "format": "date-time"
                        }
                },
                "Location": {
                        "type": "object",
                        "properties": {
                                "Latitude": {
                                        "type": "number"
                                },
                                "Longitude": {
                                        "type": "number"
                                }
                        }
                }
        }
}]
```

3.5 Container Messages

A Container is defined as a collection of Data events of identical Type and version. Each Container has a unique ID, defined upon creation of the Container.

Once a Type has been registered with a Type message, Containers can begin using that Type. Containers can be created, updated, or deleted.

The body of a Container message consists of an array of objects with the following keywords:

Name	Value
`id`	Unique identifier of the Container.
`typeid`	ID of the Type used by the Container.
`typeversion`	Optional version of the Type used by the Container. The version must be of format x.x.x.x, where x must be an integer greater than or equal to 0. If omitted, version 1.0.0.0 is used.
`name`	Optional friendly name for the Container.
`description`	Optional description for the Container.
`tags`	Optional array of strings to tag the Container.
`metadata`	Optional key-value pairs associated with the Container.
`indexes`	Optional array of Type Property ids to be used as secondary indexes for the Container.

3.5.1 Container Example

Headers

```
producertoken = b7CNvN36cq
omfversion = 1.1
```

(continues on next page)

(continued from previous page)

```
messagetype = container
action = create
messageformat = json
```

Body

```
[{
        "id": "Tank1Measurements",
        "typeid": "TankMeasurement",
        "typeVersion": "1.0.0.0",
        "indexes": ["Pressure"]
}, {
        "id": "Tank2Measurements",
        "typeid": "TankMeasurement",
        "typeVersion": "1.0.0.0"
}]
```

3.6 Data Messages

Data messages can span multiple Types and Containers. The body of a Data message is composed of an array of objects with the following keywords:

Name	Value
`typeid`	Optional ID of the type. If omitted, container is expected.
`containerid`	Optional ID of the container. If omitted, type is expected.
`typeversion`	Optional version of the Type, if one is specified. The version must be of format x.x.x.x, where x must be an integer greater than or equal to 0. If omitted, version 1.0.0.0 is assumed.
`values`	An array of objects conforming to the type.

For each object, either `typeid` or `containerid` must be specified. If `containerid` is specified, the values must conform to the Type with which the Container is associated. If `typeid` is specified, the values must conform to that Type.

If a Type Property is defined but no property value is provided in the Data message, a default value will be assumed. Default values are specified in *Type Properties and Formats*.

3.6.1 Data Example

Headers

```
producertoken = b7CNvN36cq
omfversion = 1.1
messagetype = data
action = create
messageformat = json
```

Body

```
[{
        "typeid": "Plant",
        "values": [{
```

(continues on next page)

(continued from previous page)

```
                "PlantId": "PlantId1",
                "PlantName": "Plant1",
                "Address": "123 Meridian Ave",
                "Contact": "Bob Ross"
        }]
}, {
        "typeid": "Tank",
        "values": [{
                "Name": "Tank1",
                "Serial": "5236-3523-KKF4",
                "Model": "FN-2187"
        }, {
                "Name": "Tank2",
                "Serial": "2364-4243-FS12",
                "Model": "TK-421"
        }]
}, {
        "typeid": "__Link",
        "values": [{
                "source": {
                        "typeid": "Plant",
                        "index": "PlantId1"
                },
                "target": {
                        "typeid": "Tank",
                        "index": "Tank1"
                }
        }, {

                "source": {
                        "typeid": "Plant",
                        "index": "PlantId1"
                },
                "target": {
                        "typeid": "Tank",
                        "index": "Tank2"
                }
        }, {

                "source": {
                        "typeid": "Tank",
                        "index": "Tank1"
                },
                "target": {
                        "containerid": "Tank1Measurements"
                }
        }, {

                "source": {
                        "typeid": "Tank",
                        "index": "Tank2"
                },
                "target": {
                        "containerid": "Tank2Measurements"
                }
        }]
}, {
```

(continues on next page)

(continued from previous page)

```
        "containerid": "Tank1Measurements",
        "values": [{
                "Time": "2017-01-11T22:23:23.430Z",
                "Pressure": 12.0,
                "Temperature": 100.1
        }, {
                "Time": "2017-01-11T22:24:23.430Z",
                "Pressure": 11.5,
                "Temperature": 101.2
        }]
}, {
        "containerid": "Tank2Measurements",
        "values": [{
                "Time": "2017-01-11T22:23:23.430Z",
                "Pressure": 14.0,
                "Temperature": 90.1
        }, {
                "Time": "2017-01-11T22:24:23.430Z",
                "Pressure": 15.1,
                "Temperature": 91.2
        }]
}]
```

CHAPTER 4

HTTP Behaviors

While the OMF specification does not define a particular binary message protocol, OSIsoft systems accepting OMF over HTTP conform to a set of behaviors described in this section.

4.1 Message Size

The body of an OMF message has a maximum size of 192KB. A request with a body size exceeding the maximum will be rejected with an HTTP response code of `413 Payload Too Large`. Compression is highly recommended for optimal performance.

4.2 HTTP Status Codes

The following status codes are returned by OSIsoft products accepting OMF messages over HTTP.

Status code	Description
`200 OK`	The OMF message was successfully processed. Response body contains additional information.
`202 Accepted`	The OMF message was received and has been successfully queued to be processed.
`204 No Content`	The OMF message was successfully processed, but there is no additional information to return.
`400 Bad Request`	The OMF message was malformed or not understood. The client should not retry sending the message without modifications.
`401 Unauthorized`	Authentication failed.
`403 Forbidden`	Authentication succeeded, but not authorized.
`413 Payload Too Large`	Payload size exceeds OMF body size limit.
`500 Internal Server Error`	The server encountered an unexpected condition.
`503 Service Unavailable`	The server is currently unavailable, retry later.