Guidance to Structuring Data on oneTRANSPORT

Introduction

This practical guidance document introduces you to an approach and options that you should consider when structuring data within the oneTRANSPORT oneM2M browser. Following this guidance will enable your data to be meaningful, discoverable and usable.

How should I organize data using oneM2M?

The way you want your data to be used or discovered should be the starting point for choosing how to organize your dataset. You should consider how data consumers will find it easiest to work with your data and consider the ease of integration for populating the containers.

You should consider grouping containers according to a topic such as geography, context, condition or state. You should also consider any standards that may be used with your data, such as DATEX II, and organize containers according to those standards, to enable consumers that understand the standards to integrate with your data. Most of all the structure of your oneM2M data should be clear and concise.

UTC Example

As an example, if you were to be listing UTC data you may wish to use the System Code Number (**SCN**), which is unique for each item of equipment, to group your data. A **SCN** is made up of a letter indicating the type of equipment and a 5-digit code. The equipment type letters are as follows:

- A - All equipment
- C - Car park
- D - Counting detector
- E - TC12 PC
- F - Special facility / Plan associated sign
- G - Green Wave Route
- H - Computer hardware
- J - Junction controller
- K - NMCS II equipment
- L - Tidal/Tunnel Flow controller
- M - Message Sign
- N - SCOOT Node, Link, Stage or Detector
- P - Pelican controller

- Q - Queue detector
- R - SCOOT region
- S - Car park sign
- T - Terminal (VDU / PC Workstation / Printer)
- U - Diversion
- V - Diversion Sign
- W - Analogue Sensor
- X - Outstation Transmission Unit (OTU)
- Y - Outstation Monitoring Unit (OMU)
- Z - Remote Request.

The 5 digits of the **SCN** are of the form: **SSGOE** where the letters signify: S

- **SS** the sub-3 area associated with the piece of equipment.
- **G** which group in the sub-area is associated with the equipment. In this context "group" can be viewed as a "sub-sub-area"
- **O** which OTU in the group is associated with the piece of equipment
- **E** this digit allows equipment of the same type to be connected to one OTU. Normally this has the value 1; however, for OTUs with more than one junction, count detector etc., each piece of equipment is given a different value.

Equipment Type If you consider the below:

SCN J12345 refers to Junction controller number **5** of OTU **4** in group **3** of sub-area **12**.

You may structure your oneM2M containers in the following format:

```
- J
        - 12
                - 3
                        - 4
                                - 5
```

Now consider that the structure of the oneM2M tree may actually look like this for Junction controller:

```
- J
        - 10
        - 11
        - 12
                - 1
                - 2
```

```
          - 3
                    - 1
                    - 2
                    - 3
                    - 4
                              - 1
                              - 2
                              - 3
                              - 4
                              - 5
                              - 6
```

If data were structured like this, then data consumers would be able to discover containers, and thus data, easily. For example, they could query all Junction controllers of sub-area **12**. This would then return the groups 1-3, which they could further discover.

The next example below will explain further how to structure your data and provide some physical world relevance.

SIRI Example

In the below example we explore how we can organize data for bus stops. This example is based upon SIRI **StopMonitoring** service, that provides information on bus stops. The following define the terms used in the example:

- **MonitoredStopVisit** - A visit to a stop by a VEHICLE as an arrival and /or departure.
- **StopMonitoringDelivery** – The "delivery" record of data for a Stop. This can contain multiple bus visits or cancellations.
- **MonitoredStopVisitCancellation** - Reference to a cancelled Stop Visit which should now be removed from the arrival/departure board for the stop.
- **PublishedLineName** - The route or line for a bus.
- **StopPoint** – A physical bus stop, with StopPointName being the name of the bus stop, for example Church Lane.

Each Content Instance is a **StopMonitoringDelivery** which can be published into multiple containers.

A *StopMonitoringDelivery* can contain multiple *MonitoredStopVisit* or *MonitoredStopCancellation* elements. In this example, the data is structure so that each of a *StopMonitoringDelivery* records elements is separated into its own *StopMonitoringDelivery* content instance.

```
-   <PublishedLineName>
```

```
    -   Status/
        -   Delays
            -   PT
    -   Stops/
        -   <StopPointName>
            -   MonitoredStopVisit
            -   MonitoredStopCancellation
            -   ...
        -   <StopPointName>
            -   MonitoredStopVisit
            -   MonitoredStopCancellation
```

Here we can see how the structure may look using real data:

```
-   Green
    -   Status
        -   Delays
            -   PT
    -   Stops
        -   SciencePark
            -   Visits
            -   Cancellations
            -   ...
        -   StationWay
            -   Visits
            -   Cancellations
```

How do I add meaning and context to my data?

oneM2M allows you store metadata for a container in a field entitled **Labels** [**lbl**]. Metadata provides meaning and context to the data. For example, you could add a label to the Green line container that states the service operator name of the bus line.

Labels can be as complex or simple as you require. **Labels** can be used to ensure that correct content is being processed. For example, you could create some tags in an array format that describes the container in better detail.

You can tag containers with multiple **Labels** when you create a container. Here we describe some meta data for the **SciencePark** container in multiple **Labels**:

```
[
        "ParkNRide:False",
        "Subsidised:True",
        "Operator:OwensBuses",
        "NoOfBuses:2",
        "DigitalSignage:True"
]
```

You will note that the labels are comma separated, with each of these **Labels** being a name value pair stored as a string that can be processed to understand the meta data. The actual content of each of the **Labels** is your own choice.

Storing values that are unique for the container

You may also wish to consider storing values that are unique for the container in the label field. As an example, you may have two bus stops called **SciencePark**, on different routes, and you may wish to construct your label for the **SciencePark** containers such as. For example, **Green** route **SciencePark** stop you could have the following data in its **Label** field

```
[ "Route:Green", "Stop:SciencePark"]
```

For **Red** route **SciencePark** stop you could have the following data in its **Label** field:

```
["Route:Red", "Stop:SciencePark"]
```

This way you would be able to accurately discover the **SciencePark** stop on the **Green** route using the oneM2M discovery queries, which are explained later in this document. To discover the **Green** Route **SciencePark** stop you would search for both the Labels:

```
"Route:Green"
"Stop:SciencePark"
```

If you wanted to return a list of all containers that contain data for both **SciencePark** stops, then you could search for the **Label** such as:

```
"Stop:SciencePark"
```

You would then be presented with both **Red** and **Green** route **SciencePark** containers.

It is important to note that you are free to use the Label field to categorized data in any way you wish. For example, you could use some of the data that you are storing as a label:

```
SomeXmlElementName:SomeXmlElementValue
```

It is also important to note that it is only possible to label containers, not content instances.

If a *MonitoredStopVisit* or *MonitoredStopVisitCancellation* is a content instance, then it is not possible to identify them from the other attributes, such as *VehicleRef*, *OriginRef* or *DestinationRef*. This means that it is not recommended to store different types of data in the same content instance, as you cannot tell the difference between them.

The alternative is to duplicate the same content across containers that represent the types of queries that would be issued to the system. For example:

I want to list the visits to stop SciencePark
Then the content instance for the *StopMonitoringDelivery* would be in

```
/Stops/SciencePark
```

I want to list the visits by buses on the green line to stop SciencePark
Then the content instance for the *StopMonitoringDelivery* would be duplicated in

```
/Line/Green/Stops/SciencePark
```

I want to list the visits for all buses on the green line
Then the content instance for the *StopMonitoringDelivery* would also be duplicated in

```
/Line/Green
```

Likewise, if you wanted to extend this to contain the delays:
I want to list the delays by buses on the green line to stop SciencePark
In your integration, you could define that only the *StopMonitoringDelivery* elements that have a *MonitoredStopVisit.Delay* attribute could have content instances added to

```
/Line/Green/Stops/SciencePark/Delays
```

SIRI data contains both references for lines and stops, and a human-readable name.

The human-readable name does not conform to a valid oneM2M resource name. You cannot name a container with spaces (E.g. "Science Park") so most bus stops will need to be transformed (E.g. "**SciencePark**").

How do I discover data using Labels?

To find out more about discovery in general, please see the oneTRANSPORT oneM2M Developers Portal tutorial document on the oneTRANSPORT.io website.

oneM2M allows you to search the labels for containers that contain your criteria. As an example we shall use curl to carry out a search for Containers (**ty=3**) that contain the **Labels Route:Green** and **Stop:SciencePark**

```
curl -v -H"Authorization: Bearer [Token]" \
     -H "Accept: application/json" \
     -H "X-M2M-RI: cnt-discovery" \
     -H "X-M2M-Origin: [Application AE-ID]" \
     "https://cse-01.onetransport.uk.net/ONETCSE01?fu=1&ty=3&lbl=Route:Green&lbl
=Stop:SciencePark"
```

Please ensure that you use the correct **Authorization** Token and **X-M2M-Origin** values

The response that you would receive back from oneTRANSPORT would contain a path to the containers that having a matching **Label** field.

```
{
  "m2m:uril": [
    "/ONET-CSE-01/ONETCSE01/ChordantCity/SIRI/Green/Stops/SciencePark"
  ]
}
```

If you think of using the **Label** field for **tagging** data in this way, then you may find that the response from your search contains multiple paths to containers that match your criteria. As an example, let's suppose that you all containers on the Green route were labeled as:

```
[ "Route:Green"]
```

This **Label** indicates the line that each stop is on. You may have labelled your Line as "Green" as well. Your search would look like this:

```
curl -v -H"Authorization: Bearer [Token]" \
     -H "Accept: application/json" \
     -H "X-M2M-RI: cnt-discovery" \
     -H "X-M2M-Origin: [Application AE-ID]" \
     "https://cse-01.onetransport.uk.net/ONETCSE01?fu=1&ty=3&lbl=Route:Green"
```

You would expect to receive an outcome similar to:

```
{
  "m2m:uril": [
    "/ONET-CSE-01/ONETCSE01/ChordantCity/SIRI/Green",
    "/ONET-CSE-01/ONETCSE01/ChordantCity/SIRI/Green/Stops",
    "/ONET-CSE-01/ONETCSE01/ChordantCity/SIRI/Green/Stops/SciencePark",
    "/ONET-CSE-01/ONETCSE01/ChordantCity/SIRI/Green/Stops/StationWay",
    "/ONET-CSE-01/ONETCSE01/ChordantCity/SIRI/Green/Stops/OldStreet"
  ]
}
```

Likewise, if you wanted to return all **SciencePark** stops that have a **Label** that indicates the name of the stop, then your search would look like this:

```
curl -v -H"Authorization: Bearer [Token]" \
      -H "Accept: application/json" \
      -H "X-M2M-RI: cnt-discovery" \
      -H "X-M2M-Origin: [Application AE-ID]" \
      "https://cse-01.onetransport.uk.net/ONETCSE01?fu=1&ty=3&lbl=Stop:SciencePar
k"
```

You would expect to receive an outcome that looks like:

```
{
  "m2m:uril": [
    "/ONET-CSE-01/ONETCSE01/ChordantCity/SIRI/Green/Stops/SciencePark",
    "/ONET-CSE-01/ONETCSE01/ChordantCity/SIRI/Red/Stops/SciencePark"
  ]
}
```