*Authors: Michele Gazzetti*

OFMF server and Fabric Agent interactions via Events

WIP

*Contact: michele.gazzetti1@ibm.com*

# 

DRAFT

This document describes the following scenarios:

1. Fabric Agent setup phase.
2. The registration and synchronization mechanisms carried out between OFMF and a Fabric Agent.
3. The handling of events generated by Agents and processed by the OFMF.

The OFMF is a highly available service running in the datacenter and maintaining a global view of hardware resources available on various fabrics. It provides a single point of interaction for clients to request changes of such resources. At the same time, the Fabric Agent is a service exposing an abstraction of hardware resources following the Redfish schema. The OFMF, reachable over the network via a Fully Qualified Domain Name, listens to a specific port and starts before any Fabric Agent. Each Fabric Agent needs to contact the OFMF as part of its registration procedure.

## Agent startup procedure

This section describes the steps followed by the Fabric Agent during the setup phase. These steps need to be carried out before the reception on any management or inspection request related to hardware resources.

1. As a first step the Fabric Agent reads an input configuration containing all the information necessary to carry out the setup phase. Example of these are endpoint information to contact other services like the OFMF or the Fabric Manager (if present). This document does not set any constraints on the methods used to load configuration data or Fabric Agent deployment procedures. Suitable approached could be a configuration file or command line input parameters. Following the necessary input information:
   * OFMF Event Destination that will be used by the OFMF during the initial registration and configuration phase (see section related to Agent registration).
   * Credentials to contact the OFMF
2. The Fabric Agent contacts the Fabric Manager to inspect the current state of hardware resources.
3. (Optional) The Fabric Agent, using data retrieved by the Fabric Manager, can create a local representation of the hardware resources following the Redfish specification. This step is optional as the mapping between Fabric Manager representation and Redfish could be done at runtime to keep the Fabric Agent stateless. This document does not set constraints on the implementation.
4. Once the Fabric is ready to handle the hardware resources available on the fabric, it contacts the OFMF to start the registration and Redfish tree synchronization phase (see section related to Agent registration)

Questions:

* Does the Fabric Agent register its existence (registration) before step 3 and 4 and only after 3 and 2 contacts the OFMF again for the synchronization step.
* Do we need to define an error handling procedure if step 2 or 3 do not succeed? (Fail gracefully)



DRAFT

## OFMF’s event handling

This section describes the mechanism used by the OFMF to handle incoming Events. In this section we consider only events coming from Fabric Agents.

1. The OFMF exposes a REST API as part of an internal module called Event Manager. This is a well-known single point of contact for the reception of events that other services. Fabric Agents have knowledge of such endpoint via an Event Destination that gets loaded during the initialization phase (see Agent startup procedure). A Fabric Agent sends events to the OFMF via a POST. The body of the request contains the data related to the Event and it is formatted following the Redfish Standard (ref).
   1. During the registration phase the OFMF returns a UUID to each Fabric Agent. This is added in the field named Context in each event message to identify the sender.
2. The OFMF receives the request and passes it to the Event Manager that is in charge the processing. The Event Manager inspects the content of the Event and based on the MessageId — a unique identifier for each of the events generated — run the appropriate Event Handler.
3. The Event Handler is an internal routine for processing a specific Event. While the implementation details depend on the Event itself and defining the content of each handler is beyond the scope of this document, we provide some examples of actions that could be taken.
   1. The OFMF can use the UUID to fetch the AggregationSource containing the Fabric Agent’s network endpoint information necessary to send REST requests.
   2. Inspect the resource generating the Event by sending a GET request towards the Fabric Agent. This operation is useful the gain more insights on the current state of the resource.
   3. If the event represents a change of state in the resource, the handler needs to perform the appropriate steps to maintain the internal Redfish representation up to date.



DRAFT

## Agent registration steps

This section describes the registration mechanism carried out between OFMF and a Fabric Agent.



1. As part of the setup phase the Fabric Agent loads the OFMF Event Destination. An example is shown below. This provides the endpoint of the OFMF in the Destination entry.
2. The Fabric Agent can now notify its existence to the OFMF using a registration Event (example below)
3. The OFMF stores the Fabric Agent endpoint and related data as an AggregationSource. Each Fabric Agent can be uniquely identified using the Aggregation Source Id generated by the OFMF.

DRAFT

1. As part of the response to the registration event, the OFMF return the Aggregation Source Id that will used as Context in future Events.

Example of OFMF Event Destination loaded by the Fabric Agent in step 1.

{

"@odata.context":"/redfish/v1/$metadata#EventDestination.EventDestination",  
 "@odata.id":"/redfish/v1/EventService/Subscriptions/1",  
 "@odata.type":"#EventDestination.v1\_0\_0.**EventDestination**",  
 "Id":"1",  
 "Name": "EventSubscription 1",

"Destination": https://10.1.1.1:443/EventListener,

"Protocol": "Redfish",

"Context": "”,

**~~”ResourceTypes" :[ “<AGENT\_REGISTRATION\_MESSAGE>”,“Fabric”,…,](OPTIONAL?)~~**

**~~"SubordinateResources ": true~~**

}

Where:

* Destination: endpoint used by the Fabric Manager to send Events via POST requests.
* Context: empty field for the duration of the startup procedure. This will be updated during the registration phase and it will contain the UUID identifier provided by the OFMF.
* ResourceTypes: resources of interest for the OFMF.
* SubordinateResource: see Redfish specification.

Question:

* Which ResourceType should we use for the registration phase?
* Is this EventDestination only for the registration phase? Does the OFMF create a new one once the registration phase is over?

Following an example of registration event sent by the Fabric Agent to the OFMF in step 2. As a note, we are using a single entry in MessageArgs to send all the necessary data to contact the Fabric Agent in the future. Such data could be encrypted. The OFMF will parse the metadata and save them as local Redfish objects.

{

"@odata.type": "#Event.v1\_7\_0.Event",

   "Id": "1",

   "Name": "Fabric Agent Created",

   "Context": "",

   "Events": [ {

"EventType": "Other",

"EventId": "4593",

"Severity": "OK",

"Message": " The resource has been created successfully.",

"MessageId": "**ResourceCreated**",

DRAFT

"MessageArgs": [ “”  ],

  "OriginOfCondition": {

"@odata.id": "/redfish/v1/<TBD>”

},  
 “Extra payload”

 ]

}

**TODO**: Update with following approach:

* use extra payload 🡪 check example in Redfish school
* use for blobs

Where:

* METADATA is a placeholder for a data structure (json) containing all the information required for the OFMF to contact the Fabric Agent at a later time. As a note, this could be encrypted
* OriginOfCondition: depends on how we want to expose the FabricAgent from a Redfish point of view. Can we omit it?

The agent contacts the OFMF to registers it existence using an Event. This event informs the OFMF that a new agent is available on the network and reports all the metadata necessary for the OFMF to reach the Fabric Agent in the future. Example of such information are URL Endpoint and port.

## Agent synchronization steps

After the registration phase, the Fabric Agent initiates a synchronization phase following two possible methods:

1. **Recursive inspection (Crawl-out)**: The OFMF inspects each Fabric Agent’s resource that needs to be tracked via multiple GET requests.
2. **Snapshot synchronization:** the Fabric Agent sends a single Event containing a representation of the resources that it wants to expose to the OFMF.

### Recursive Inspection

DRAFT



Following an example of Event sent by the Fabric Agent to trigger the synchronization phase.

{

"@odata.type": "#Event.v1\_7\_0.Event",

"Id": "2",

"Name": "Fabric Created",

"Context": "",

"Events": [ {

"EventType": "Other",

"EventId": "4595",

"Severity": "Ok",

"Message": "The resource has been created successfully. ",

"MessageId": "ResourceEvent.1.2.1.ResourceCreated",

"MessageArgs": [],

"OriginOfCondition": {

"@odata.id": "/redfish/v1/Fabrics/<FABRIC\_ID>"

}}]}

## Snapshot synchronization



Following an example of Event sent by the Fabric Agent to advertise the internal state.

DRAFT

{

"@odata.type": "#Event.v1\_7\_0.Event",

   "Id": "1",

   "Name": "Fabric Agent State",

   "Context": "",

   "Events": [ {

"EventType": "Other",

"EventId": "4594",

"Severity": "Info",

"Message": "Fabric agent state: <ENCODED\_REDFISH\_TREE>",

"MessageId": "FabricAgent.1.0.FabricStateCreated",

"MessageArgs": [ “<ENCODED\_REDFISH\_TREE>”  ],

  "OriginOfCondition": {

"@odata.id": "/redfish/v1/Fabric/<FABRIC\_ID>”

}

 ]

}