## OFMF Architecture and Software Stack Structure

The OFMF architecture is depicted in the following figure. Red lines indicate Redfish API formatted requests. Green lines indicate fabric specific formats:



Figure 1: General Architecture of an example Gen-Z fabric managed using a typical OFMF Service

### Clients

Clients are any of the applications, application libraries (such as libfabric, OpenFAM, or OpenSHMEM), resource managers (such as FAM pool managers, storage pool managers), orchestration managers, workload managers, and the admin GUI and tools that call into the OFMF’s Redfish Services.

Client software queries the OFMF Redfish Service (using HTTP GET, for example) to determine what resources are available on the fabric. Clients issue updates (using HTTP PUT, POST, PATCH, DELETE) to objects to make modifications to the Redfish fabric model.

Clients identify the fabric resources by the URIs given them by the OFMF’s Redfish Service. When clients need to exchange information about fabric resources, they use the OFMF’s URIs to unambiguously refer to specific objects. Clients accept the Redfish object description as the common truth. The URIs in use within the OFMF Redfish Service are unique only to that Service, and make up that OFMF’s Redfish URI namespace.

### Composability Manager Services

The element that sits between clients and the OFMF central block is collectively called the Composability Manager.  The Composability Manager can be seen as a collection of resource managers, policy stores, and monitoring elements for tracking the current state of the entire system.  The Composability Manager is in charge of composing hardware resources according to requirements coming from clients and by applying specific policies. Technically the members of the Composability Manager are all clients of the OFMF Services and any user or admin application can pursue taking responsibility for these tasks. However, within the OFMF Architecture we anticipate releasing a *coordinated set* of reference code utilities to offer random clients commonly needed ‘composability’ features. Examples of such features (not an exhaustive list):

* Composition/Deletion of Computer Systems and Clusters
* Dynamic update of Computer Systems resources
* Application driven dynamic resources configuration and provisioning such as:
  + Creation and binding of memory and storage resources
  + Creation and binding of shared memory region across nodes
  + On-demand provisioning of accelerators
  + Virtual network creation
* Management of client permissions/resource ownership

*Coordinating these various clients enforces a cohesive interpretation of the Redfish objects and schema which are used to describe the fabric resources being managed.* In addition, permissions and access controls for shared fabric resources need to be managed in this layer, which is above the OFMF which enforces access controls and below the general Clients upon which the restrictions are imposed.

### OFMF Services

The OFMF Services are presented to all clients via Redfish API calls. The OFMF maintains the aggregate Redfish model of all fabrics it controls and all resources on those fabrics. When clients request data or request changes to model objects, the OFMF determines which Redfish objects in the model are impacted, makes the required changes to those Redfish objects in the model. Any relevant actions or requests that affect state or configuration of the fabric manager or actual fabric hardware are relayed by the OFMF to the fabric-specific Agent.

The OFMF also subscribes to events from the various Agents and offers its own Events Subscription Service to OFMF Clients. Clients and the Composability Manager can subscribe to the various Redfish events defined for the Redfish objects, and when the OFMF receives associated events from the Agents these events are forwarded to those subscribed to receive them.

Not all client requests will require the OFMF to interact with the fabric Agent. Many requests (for example, a request to register to receive certain events) simply affect the OFMF fabric model and some bookkeeping properties therein. Some simple client requests (for example, to DELETE an object) may translate to multiple changes to multiple model objects and potentially require multiple exchanges with the fabric Agent. Maintaining the integrity and consistency of the aggregate Redfish model of all fabric resources is one of the primary duties and major values of the OFMF.

Finally, the OFMF is responsible for tracking and enforcing Authentication and Access Control policies for both Clients and Agents.

### Agents

Fabric-specific Agents act as the translators between the OFMF’s Redfish API syntax and schema and the fabric-specific versions used by the given fabric manager software. Agents thus ‘speak Redfish’ to the OFMF, and speak ‘(potentially) fabric-specific protocols’ to an actual fabric manager. For example, the fabric manager may have a RESTful interface called ‘bind resource’ which allows an admin to enable Host A to access Memory B. The OFMF’s Redfish equivalent is ‘POST Connection’ between Host A and Memory B.

Fabric-specific Agents also act as the translator between the OFMF’s Redfish URI namespace and the fabric specific component and resource namespaces. For example, the open source Gen-Z fabric manager for Linux (called Zephyr) assigns its own 128-bit UUID-style ID to a Gen-Z fabric memory module. Redfish models a complicated fabric resident memory module as several related Redfish objects (Fabric Adapters, fabric Ports, fabric Endpoints, Memory Domains, etc). The Gen-Z Agent is responsible for keeping the mappings between the various Redfish IDs (URIs) assigned by the OFMF and the associated IDs (UUIDs) assigned by the Zephyr Fabric Manager. Clients use the Redfish IDs, and the fabric manager uses the FM IDs. The Agent is possibly the only entity that knows both namespaces.

Like the OFMF, an Agent needs to parse requests coming from its client (the OFMF), modify its internal representation of its fabric view, update any internal state it is required to track, and send any appropriate request or requests on to the fabric manager.

Another very important role of the Agent is to aggregate the inventory and events from multiple hardware Fabric Manager instances used to configure and control larger fabrics as Figure 1 illustrates. Since a large fabric may have multiple hardware Fabric Managers controlling components on the fabric which must share a common fabric address namespace, the task of reconciling multiple namespaces and ‘subnets’ into a single Redfish Fabric representation falls to the Agent.

If the fabric components include self-contained, self-managed resource pools such as a storage appliance that have a Fabric Manager instance that implements a high level management API, the Agent is responsible for communicating with this hardware manager and translating any resources and management API commands to be made visible to the OFMF Redfish Service into appropriate Redfish equivalents. A storage appliance might export its own Swordfish/Redfish API via an in-band interface and the fabric Agent must translate this into a form visible to the OFMF so that Clients can have one Redfish Service to contact. (The previous statement assumes storage volumes on the storage appliance are to be represented as fabric resident volumes that can be bound to one or more hosts also resident on the same fabric using standard Redfish API calls. There are other arrangements that can be made, but our reference version of a Composability Manager / OFMF Services / Agent will the above assumption.)

Finally, the Agent must pass on events, alerts, and performance monitoring updates in support of the OFMF’s Redfish events service. The Agent – Fabric Manager interface is fabric manager specific. The OFMF – Agent interface is architected to be a Redfish API using Redfish schema so the Agent must translate events and logs emitted by or retrieved from the FM into the appropriate Redfish formats.

### Fabric Managers

The terms ‘fabric manager’ and ‘fabric management’ carry many different interpretations throughout the industry. Certainly, the large numbers of functions and features required to manage even a modest ‘fabric’ may require many different blocks of code to execute in many different ‘layers’ of a ‘fabric management software stack’.

For the purposes of the OFMF Architecture, Fabric Managers (FMs) are those entities with physical access to the control space of the fabric resources and the authority to modify those settings. The FM is responsible for performing a fabric crawl, taking inventory of fabric resources, and the initial configuration of such resources as required by the FM’s initial configuration policies. The FM’s ‘management domain’ is those components for which it has primary access rights to the associated control surfaces. (‘Fabric Manager’ as used herein is Gen-Z terminology, but the more popular ‘subnet manager’ term is inconsistent across various other fabrics, so we will use Fabric Manager herein.)

#### Number of FMs for a single fabric type

Another major assumption of the OFMF Architecture is that the Fabric Manager (per OFMF definition) need not be a single entity or single thread of execution. Even a modest sized fabric may have multiple Fabric Manager instances running, each in control of specific components. Each such FM may converse with the same Agent, or each may have its own Agent. The Agent instance is responsible for aggregating the resources from all FMs it is servicing, tracking the associated states and managing the associated resource IDs and Redfish namespace mappings.

*It is the number of Agents* of a fabric type that register with the OFMF that determines how many ‘Fabric Instances’ appear in the OFMF’s Redfish collection for that fabric type. If one Agent is presenting one fabric-specific API to the OFMF Services, that one Agent is responsible for aggregating the namespaces and resources for all FMs reporting through it. If multiple Agents are communicating with the OFMF, there will be one Redfish Fabric Instance for each Agent. Thus, Clients may see multiple ‘fabrics’ of the same type in the Redfish Service.

## The Agent Software Architecture

The Agent is organized similarly to the OFMF, but with an emphasis on its roles of translator between OFMF Redfish and Fabric Manager proprietary APIs and keeper of fabric specific state.



Figure 2: General Features Architecture of an Agent

The following figure describes the general code block diagram of the Gen-Z Agent:



Figure 3: Agent Code Block Diagram

The Agent SW Code Block Diagram, *Figure 3* shows the major code functions that make up the Agent Services. The orange colored blocks are heavily leveraged from the OFMF Services repository. The green colored blocks (on the right side) are the specialized Agent blocks. As in the OFMF Services, files created by the Agent Service are depicted in grey. As in the OFMF Services, the Gen-Z Agent Services are launched and coordinated by the main emulator module (./emulator.py).

### The Orange Blocks

Since these blocks are mostly in common with the OFMF Services, the reader is encouraged to consult the OFMF Services Theory of Operation for detailed explanations of each. The following is a brief overview of the set of these blocks to set the context of their purpose in the Agent Services.

Since the Agent has a Redfish (HTTP) network service as the interface to clients of the Agent Services (currently just the OFMF), the **HTTP API** is based on flask just like the OFMF. Likewise, the Agent fields Redfish API requests for manipulations of Redfish objects which the Agent manages by sorting the incoming requests via the **resource\_manager** module (resource\_manager.py) which matches the specific Redfish object named in the request to a **Redfish object processing module** (resourceObject\_api.py). These specific modules have functions to process individual GET, PUT, POST, DELETE, and actions for each Redfish object. The specific module codes, when they need to create an actual Redfish object’s JSON description may use an **object-specific template** (resourceName#.py) to do so. As in the OFMF Services stack, there are many **utility routines** (utils.py) used to perform manipulations on objects and the Redfish resource tree that are common to most or all Redfish objects.

The Agent versions of the resourceObject\_api.py modules (for example, the f\_connections\_api.py) are where the most differences exist between the Agent versions and the OFMF versions of the Orange Blocks. The OFMF versions are targeted at manipulating the relatively abstract Redfish fabric model (POST a new Memory Chunk within a given Memory Domain.) Many OFMF Client requests (GETs for example) do not impact the Agent’s or the Fabric Managers’ managed state of the actual fabric resources. Thus, only those OFMF Client requests that do impact the Agent’s and Fabric Managers’ managed state of actual resources need to be supported at the Agent’s OFMF interface.

The Agent’s versions of the resourceName\_api.py modules are tasked with mapping the changes requested to the abstract Redfish description of fabric resources to specific changes to the actual hardware resources and determining which, if any Fabric Manager API calls need to be made. The section on Specific Redfish Fabric Objects of Particular Interest to the Fabric Management Problem lists and explains which Redfish Objects a particular Agent will be handled by the Agent’s Redfish API interface to the OFMF (and other Agent Clients.)

Gen-Z POC Note: The Gen-Z Agent directory of resourceName\_api.py modules (./api\_emulator/redfish/) still contains the majority of the OFMF Services api.py modules, even though many of these objects are not relevant to the Gen-Z Agent. A reference implementation will eliminate the un-necessary files.

### The Green (Agent Unique) Blocks

The Agent’s primary role is aggregating and translating the fabric specific Fabric Managers’ representations of fabric resources and fabric state into Redfish schema and translating the OFMF’s Redfish object requests into specific Fabric Managers’ commands. The Gen-Z Agent Architecture is described below in the context of the Gen-Z Agent for organization and terminology (until the OFMFWG solidifies the Agent Architecture and Terminology), but many Agent architectures will be similar in functionality.

#### Events and Logs

The Agent is responsible for processing events from the FM to maintain the Agent’s model of the fabric, issuing appropriate Events to the OFMF so the OFMF can maintain the OFMF’s Redfish model of the fabric, maintaining an EventRegistry for the Agent’s Clients (the OFMF), and forwarding all Events to which a Client (the OFMF) has subscribed.

When the Agent receives an Event from an FM, it parses it, translates it from FM namespace to Redfish namespace, maps it to one or more Redfish events and assesses its impact. If an event clearly denotes a change to the Agent’s internal fabric representation and/or resource description (for example, a hot-add of a new resource), the Agent shall make such changes as appropriate. If an event clearly denotes a change to the Redfish fabric representation and/or resource description maintained by the OFMF, the Agent shall send the appropriate event to the OFMF.

In the Gen-Z Agent Reference Implementation, Events and Logs are located primarily in the TBD.tbd code module.

#### Fabric Inventory and Topology Processing

The Agent obtains a Fabric Manager-specific representation of the inventory and topology of the fabric. This representation has to convey the components (resources) accessible via the fabric and the physical links among these components. The Gen-Z Agent implementation is designed to interface with the open source Gen-Z Linux Fabric Manager (Zephyr…. Give github reference here).

The Zephyr FM responds to requests for the Gen-Z fabric components and topology by returning a a topology graph in JSON format. The Agent code module (./agent\_utils/zParser.py) zParser.py has numerous utilities that parse the JSON topology description, assign Redfish IDs to appropriate components, extract the Redfish Port to Port linkages (the physical connectivity), and build an Agent specific data base of all components, linkages, Redfish IDs and associated Fabric Manager IDs.

Dynamic changes to the existing topology are processed in a fabric-dependant manner using modified versions of the utilities that parse the initial topology description.

When the Agent creates an initial fabric database or updates an existing fabric topology it must alert the OFMF Service of the need to create or update the OFMF’s view of the fabric. When the OFMF Service queries the Agent (via Redfish GETs), the Agent creates actual Redfish JSON descriptions of all the objects in response.

#### Existing Resource Allocations and Bindings

An Agent may be attaching to a managed fabric that has some resources already allocated (Memory Chunks carved from a component’s Memory Domain ) and possibly bound to hosts or other consumers. Such allocations and bindings might have been made by the FM as part of its initial fabric configuration operations, or may have been made by a previous invocation of an Agent. The modules that parse the topology generally must run first to create the raw resources from which allocations can be made and among which bindings might be made.

The Gen-Z Agent obtains the current list of resource allocations and bindings from the Zephyr Fabric Manager in the form of a JSON description of producer-consumer bindings. The code that parses this JSON ‘Resource’ description is found in the (./agent\_utils/zParseResource.py) zParseResource.py module. This module locates the referenced ‘resource producer’ and ‘resource consumer’ in the Agent’s database of the fabric inventory, creates any required ‘resource block’ (eg, a Memory Chunk) described by the JSON ‘Resource’ description, creates the stated binding (eg, a Connection), updates the Agent’s database accordingly, and alerts the OFMF Services of the fabric changes. As in the case of the basic fabric inventory and topology, the Agent must create actual Redfish JSON object descriptions for any resources (IE, Memory Chunks) and bindings (IE, Connections) and return them in response to the queries from the OFMF Service.

### Agent File Stores

#### Agent Data Store

Most Agents will need to have a database in which to store the important details of the fabric inventory and topology and the logical resource allocations and bindings. This database will also have to contain the corresponding Redfish IDs and FM IDs, as well as meta-data such as cross reference structures, OFMF URL’s, FM URL’s, timestamps, resiliency details, operational modes and policies.

The Gen-Z Agent has one large database kept as a general Python Dictionary and written to a JSON formatted file (./agentDB.json).

#### Redfish Objects Resource Files

The OFMF Service architecture creates its Redfish resource tree (Redfish data base) as a file system hierarchy that contains a Redfish object file in JSON format. The Agent Service creates its fabric resource tree in the agentDB.json dictionary file. However, whenever the Agent creates a Redfish JSON description of a Redfish object which it sends to the OFMF Service in response to a OFMF request, the Agent may write that object as a JSON file to the ./agent\_POSTs directory.

PoC Note: In the OFMF-Agent-POC implementation the Gen-Z Agent creates these Redfish object files at the time it parses the initial inventory or updates inventory in response to dynamic changes. Since Events are not supported in the POC, the Agent pushes Redfish objects and updates to the OFMF Service by issuing POST or PATCH commands directly to the OFMF Service’s Client API using these object files as the JSON body of such commands. An Agent reference implementation may have no need to save the actual Redfish JSON objects in a file, except maybe as a debugging option, or as a performance enhancement (since re-creating Redfish objects from the Agent’s database might be tedious for certain objects.)

#### Fabric Manager Command Files

Whenever the Gen-Z Agent sends commands to the Fabric Manager interfaces, these commands are created as JSON descriptions for the FM’s RESTful interfaces. These JSON structures are written to a directory (./zephyr\_cmds in the case of the Gen-Z Agent reference implementation). The Gen-Z Agent reference implementation may have no need to save actual command JSON contents in a file except as a debugging option, but since Zephyr DELETE resource bindings take exactly the same arguments as the Zephyr ADD resource bindings it is highly efficient to save the Zephyr ADD commands.

### The Agent Major Responsibilities

The general nature of Agent duties are explained in the overview of *Agents.* The following is a partial list of key responsibilities of the fabric Agent Service vs the OFMF Service.

* The Agent is responsible for locating the OFMF’s Event Service and issuing a form of New Fabric Event Notification to initiate OFMF and Agent communications.
* The Agent is responsible for declaring the Redfish IDs of all fabric physical components because the Agent presents these objects to the OFMF. The Namespace translation between the OFMF’s Redfish namespace and the Fabric Manager’s (potentially) proprietary namespace is managed by the Agent.
* Logical objects created by allocating or binding resources of the physical components are given their Redfish IDs by the Service that defines them. For example, when Clients of the OFMF Service choose to POST a Memory Chunk, the Client decides what Redfish ID to use in the POST request. If the FM had pre-configured Memory Chunks bound to specific hosts on the fabric, the Agent will assign their Redfish IDs before it makes them known to the OFMF Service.
* The Agent is responsible to aggregate multiple FM inventories into one fabric namespace since Gen-z fabrics have a flat fabric address space that must be disambiguated before the fabric can carry general data traffic. Doing this aggregation makes the Agent responsible for creating a unique Redfish ID for each unique fabric component in the aggregated fabric.
* The Agent shall provide an HTTP Redfish API to the OFMF
* The Agent shall track the status of the OFMF representation of the fabric in its Agent – specific internal fabric database for all the fabric objects that it has presented to the OFMF Service, and for all the logical fabric objects sent to it by the OFMF Service.
  + The OFMF can use GETs to the Agent’s API Service to get caught up if the OFMF drops
  + The Agent cannot use GETs to the OFMF Service to discover what was made visible to the OFMF should the Agent drop. The Agent must alert the OFMF that they are potentially out of sync.
* The Agent shall issue an Agent Restart Event to the OFMF Service if the Agent drops and recovers.
  + *OFMF and Agent actions in response to an Agent Restart are TBD.*
* The Agent parses Redfish requests from the OFMF and
  + Evaluates the impact on the Agent’s model of the fabric
  + Evaluates the impact on the FM’s state of the fabric, and creates an appropriate FM specific request to cause the FM to update the state of the fabric accordingly
  + Returns success or failure of the OFMF request along with an appropriate Redfish object that results from a successful request
* The Agent parses any Agent specific HTTP Requests. These are requests made through the Agent’s client interface that target Agent internal configuration, and are not meant to directly manipulate Redfish objects. Think of these as Agent administration requests.
  + Examples: Restart the Agent, sync with a Redundant Agent
  + PoC Note: there are no Agent specific HTTP Requests defined
* The Agent registers for events from the FM to receive notice of dynamic changes in fabric hardware and FM or Host software

### Gen-Z Agent Supported Redfish Fabric Objects

#### Fabric Adapters and associated Ports

Redfish Fabric Adapter objects connect a resource supplier such as a storage controller to the fabric, or enable a consumer of resources such as compute entities to place requests on the fabric. Resources available through the fabric adapter (memory or storage for example) or compute entities that access the fabric through a fabric adapter may be modelled in other objects that are linked to the adapter.

Fabric Adapters which connect directly to the fabric have associated Redfish Port objects that describe physical and logical interconnect details. Adapters with multiple ports to the fabric have multiple subordinate Port objects. Redfish models Ports as separate objects to allow different links to have different properties and controls, and to keep the basic Fabric Adapter model succinct even for very large port count devices.

#### Switches and associated Ports

Switch objects relay fabric traffic (packets) from fabric ingress links to fabric egress links. Similar to the Fabric Adapter, the switch features, functions, and actions associated with the switch overall functionality are modelled in the switch object, while link and port details are modelled in the subordinate Port objects. Ingress and Egress port filtering and routing tables are examples of such port details, and these are often very fabric specific as well. The Gen-Z Agent may not populate these details in Redfish objects sent to the OFMF Services.

The physical topology/connectivity of the fabric can be discerned by tracing the Ports connectivity data. OFMF Clients generally have no permissions to modify Gen-Z fabric switches directly, but such clients may need to see the physical topology to understand which fabric resources are most appropriate to bind to each other.

#### Endpoints (associated with initiators or targets of fabric traffic)

Redfish Endpoints model the actual addressable entity of a fabric that either sources (initiatior) or consumes (target) fabric packets. For some fabrics (ex. PCIe), every initiator or target port is uniquely addressable and a multi-port device will have multiple fabric addresses. Yet, the one component might contain just one target or initiator functionality that needs to be modelled. Conversely, a single port component might contain multiple entities that need unique Redfish models to convey and control their functionality even though fabric packets access them at a single fabric target address.

Switches are not Endpoints on a fabric unless there are resources within the switch which are the target or initiator of fabric packets. Gen-Z switches contain management structures that are accessible in-band, so all Gen-Z switches have a corresponding management Endpoint modelled.

#### Memory Domains

Memory Domains are associated with entities that supply memory resources via an Endpoint on the fabric. Memory Domains describe the total memory resources available via the entity. This total capacity is carved into one or more Memory Chunks.

#### Memory Chunks

Memory Chunks are associated with Memory Domains, and describe a subset of the Memory Domain capacity. Memory Chunks are the ‘allocated memory space’ within a Memory Domain that can be bound to a fabric initiator.

#### Zones (and Zones of Zones)

Zones are a list of Endpoints on a fabric which have been granted routing privileges amongst the Endpoints in the list. Naming two Endpoints to the same Zone enables at least one valid path for a packet sent from one Endpoint to follow to reach the other Endpoint. Routing choices enabled by the Gen-Z Fabric Managers must adhere to the restrictions implied by Redfish Zones.

The Gen-Z PoC Agent does NOT support Zones. The support of Zones in the Gen-Z reference implementation is TBD.

#### Connections

Connections establish access permissions between an initiator Endpoint (or group of Endpoints) and a resource provided by a target Endpoint (or group of Endpoints). Zones enable the fabric to route packets between endpoints. Connections are required to enable the initiator to launch the packet and the contents of the packet to be executed by the target Endpoint(s). (Connections might be thought of as enabling packets to pass through a source or destination firewall.)

The Gen-Z Agent supports Connections. The PoC version only supports a Redfish Connection between one Target Endpoint and one Initiator Endpoint; it does not support Endpoint Groups or lists of multiple Target or Initiator Endpoints.

#### Connection Policies

Connection Policies serve to limit permissions that may be granted by those Connections to which they are applied. The Gen-Z PoC implementation does not support Connection Policies.

#### Address Pools

Address Pool objects contain constraints on the ranges of fabric addresses which may be assigned to specific Endpoints or specific Zones. The Gen-Z PoC implementation does not support Address Pools.

#### Additional resources

Additional resources that can be accessed through the fabric adapter at a given endpoint can be modelled in the OFMF tree if the details of such resources are POSTed by a knowledgeable (and authorized) entity such as a Storage Array Manager Agent. Gen-Z FMs which explore their fabric and take inventory of the fabric-touching resources may not have the ability to query about what is behind a Gen-Z Fabric Adapter. If it isn’t visible to the FM, it isn’t managed by the FM, and the OFMF representation can only be a ‘placeholder’ object that anchors such a Fabric Adapter into the standard Redfish resource tree.

## The OFMF – Agent Interface

The normal operating model for communications between the OFMF Services and the Agent Services is expected to be:

* The Agent sends the OFMF Event Service a form of ‘new fabric’ Event to alert the OFMF of the existence of a new fabric and associated Agent. The Event message will inform the OFMF of the location of the Agent’s Redfish API Service.
* The OFMF sends Redfish requests, event subscriptions, and any specialized commands to the Agent’s Redfish API Service.
* The OFMF does not send Redfish Events to the Agent
* The Agent sends responses to OFMF requests to the requesting URL, as is the normal behavior for an HTTP API Service.
* The Agent does not send any form of request to the OFMF’s Redfish API Service.
* Instead, if the Agent needs the OFMF to update the OFMF’s Redfish models of the Agent’s fabric resources, the Agent sends the OFMF’s Event Services an appropriate Redfish Event, much like it did to announce the presence of the Agent and the fabric.
* The OFMF’s Event handlers will issue corresponding Redfish requests (GETs mainly) to retrieve the objects or logs indicated in the Event it has received.

The Gen-Z PoC Agent used the OFMF API Client Services to POST the fabric inventory because Event processing did not exist in the OFMF Services emulator. *Normally, this is not allowed* as the Gen-Z Agent may be consulted in the processing of Gen-Z requests and a deadlock loop would be formed if the Agent initiated a request on Gen-Z fabric objects. However, for the Gen-Z PoC, the OFMF object-specific processing routines were modified to not consult with the Gen-Z Agent if the Gen-Z Agent was the source of the request.

Such special case code is not allowed and not included in any OFMF Services or Agent Services reference implementation. Thus, Agents do not manipulate their own fabric resource representations in the OFMF resource tree by call the OFMF Services Redfish APIs.

### Extraction of New Fabric Details

When the OFMF Event handler for a new fabric is triggered, it does the following:

The OFMF issues one or more GET commands to the Agent’s Client API Services. The targets of these GET commands are:

* The Redfish Fabric instance which was included in the new fabric notification Event
* Subsequent sub-ordinate resources of the Fabric instance; examples:
  + the Redfish Manager object (describing the Agent details) linked to the Fabric instance in the ‘ManagedBy’ property (does this property exist yet?)
  + the Fabric’s Endpoints instances
  + the Fabric’s Connections instances
  + the Fabric’s Switches instances
* Subsequent objects listed in the navigation links for the Endpoints, Connections, and Switches
  + Fabric Adapters, Memory Domains, Memory Chunks, Systems, Chassis, etc

As each of the new fabric objects, resource allocations and bindings are retrieved from the Agent, the OFMF shall register for change Events for each object. Any dynamic changes to these objects, or dynamic additions to the appropriate collection shall be signalled via an appropriate Redfish Event from the Agent to the OFMF Event Services URL listed in the subscription. Queries from the OFMF Event handler will retrieve the appropriate objects and logs.

The details of the Redfish EventServices are part of the Redfish specification (link here).

### OFMF Client Requests Requiring Agent Processing

The OFMF Clients requests against objects on the Gen-Z fabric may require the OFMF communicate with the Gen-Z Agent before it can satisfy the Client’s original request.

Simple GET requests may require the OFMF to send a copy of the GET request to the fabric Agent, depending upon policies and processes established for a given OFMF – Agent combination. In general, OFMF Architecture encourages a GET be satisfied by the OFMF Redfish resource tree unless there is a pending notification of change from the Agent or the OFMF tree has a potential to be out of date.

Client requests that modify fabric Redfish objects are inspected by the object-specific processing module (the objectName\_api.py code in the OFMFWG repositories) to determine fabric changes required and determine the associated requests (if any) to send to the Agent(s).

The section titled Gen-Z Agent Supported Redfish Fabric Objects defines a minimum required set of Redfish Objects that the reference implementation needs to support so that a general Gen-Z fabric can be modeled and managed via the OFMF Services. In an OFMF – Agent implementation that supports Events, the fabric Agent is expected to respond to GET requests on all the objects associated with fabric components that are initially described by the FM / Agent. Most of these components, once visible in the OFMF resource tree have Redfish details that can be modified by OFMF Clients (via POST, PATCH, or DELETE requests), so most Redfish operations may need to be relayed from the OFMF Service to the Agent Service to keep the OFMF state and the Agent state for these objects in sync.

The Gen-Z FM API combines the creation of Memory Chunks with their assignment as a target of a Connection. If a Memory Chunk is defined on a Memory Domain by an OFMF Client or the Composability Manager, the Agent’s database should be updated accordingly. However, the Gen-Z Agent has no mechanism to inform the Fabric Manager that such an allocation has been made. The Agent should track such logical subdivisions of component resources in case the FM sends Events for the source component that impact the state of the logical resources (eg, Memory Chunks) that might be derived from that source.

When an actual Gen-Z consumer (IE, a Host System) of an allocated resource (IE, Memory Chunk) is bound to it (IE, a Connection made), the Redfish POST Connection request made by an OFMF Client will need to be passed to the Gen-Z Agent’s Redfish API, where it will be translated into the FM’s call to add\_resource(). The add\_resource() call describes both the Memory Chunk (the ‘resource’ to be mapped) and the Connection (the producer of the resource and the host consuming the resource) to the Gen-Z Fabric Manager.

Anytime the OFMF relays a Redfish request to the Agent, it should not update the OFMF Redfish resource models until and unless the Agent returns success. The OFMF request to the Agent should contain the JSON request body that came with the OFMF’s Client request. Often, the Agent will fill in specific properties in the Redfish object for POST and PATCH operations, and these updates shall be placed into the OFMF’s copy of the object.

The OFMF is responsible for serializing different request streams sent to multiple Agents if the OFMF originates these multiple streams. For example, if a client requests a connection between Redfish objects on different subnets modelled as different ‘fabrics’ and thus involving two different Agents, the OFMF is responsible to coordinate the sequencing of those two request streams.

A complete list of the Redfish API requests which can be sent by the OFMF to the Gen-Z Agent is contained in the release documents of each Agent release. (This documentation is not yet available for the Gen-Z Agent reference code implementation)

PoC Note: This documentation is also not available for the Gen-Z PoC Agents.

## The Agent – Fabric Manager Interface

### Agent – FM Link up

How the Gen-Z Agent and Zephyr Fabric Manager find each other is still being architected. Currently the Gen-Z Agent PoC implementation hard codes a Zephyr FM URL which supports a simple RESTful HTTP interface. This simple RESTful interface is defined on the Zephyr Open Source Linux Gen-Z Fabric Manager github site, (need URL to github repo), but two currently used by the Gen-Z PoC Agent implement are described herein. The Zephyr github documentation should be consulted for the most accurate interface options and details. The following is a snap shot in time from the Gen-Z PoC implementation.

### FM Queries

#### Topology

The Gen-Z Agent can retrieve the current Gen-Z fabric inventory of components and their interconnections from the Zephyr Gen-Z fabric manager (for Linux) by issuing an HTTP GET request to the Zephyr interface URL with path=/fabric/topology. The response will return a JSON formatted topology graph containing JSON structures which describe the components found (the nodes = vertexes in the graph) and the physical links (the edges = the links between nodes). Examples of simple topology graph JSON files can be found on the Gen-Z PoC github site (need URL). In general, the Zephyr topology graph JSON description contains three major dictionaries that inform the Agent of the critical details about the discovered Gen-Z resources:

* “graph” details: Really details about the fabric as a whole, such as an FM assigned fabric UUID, a list of manager UUIDs, and the Gen-Z hardware-defined device ID which hosts the Zephyr manager Endpoint on the Gen-Z fabric.
* “node details: A list of each Gen-Z components on the fabric. Each node dictionary contains important information such as an instance UUID, the Gen-Z component class of the component, the UUID of the component’s primary manager, the Gen-Z component ID used to route to the component, other component IDs, and the maximum memory this component can present to the fabric, plus various status and capabilities.
* “links” details: A list of ‘link’ objects. Each link object describes the components at each end of the link: the instance UUID of the component, the port number this link connects, and the port state and PHY status.

#### Resources

The Gen-Z Agent can retrieve the current Gen-Z fabric inventory of fabric attached Memory Chunks (memory resources) provider from ‘producer nodes’ and which are mapped to hosts (compute node ‘consumers’). The Agent does so by issuing an HTTP GET request to the Zephyr interface URL with path=/fabric/resources. The response will return a JSON formatted ‘resources’ object containing JSON structures which describe both the Memory Chunk resource and the Connection between the memory source Endpoint and the host which consumers the memory. Examples of simple ‘resource’ JSON files can be found on the Gen-Z PoC github site (need URL). In general, the Zephyr resource JSON description contains essential object descriptions that inform the Agent of the critical details about the mapped resource producers and consumers:

* Fabric Details: Mainly just the same fabric ID as returned with the topology description, in case multiple fabrics are managed by this FM or multiple FMs are associated with one Agent
* ‘resource’ object Details: Each resource object has
  + A list of ‘consumers’, which are the node IDs (from the topology node graph) of the hosts that will use the memory resources
  + A single ‘producer’, which is the node ID of the component that sources the memory resource to the fabric
  + A ‘resources’ list, which is a list of Memory Chunks from the single producer

Note that the ‘resources’ list (unfortunate Key name) describes one or more Memory Chunks, and the ‘producer’ node ID dictates the ‘target Endpoint’, and the ‘consumer’ node IDs dictate the ‘initiator Endpoints’ of a Connection.

#### Timestamps and Changes

The Resources and the Topology JSON descriptions returned by Zephyr when the Agent queries them also contain at least two timestamps:

* A ‘current timestamp’ that denotes when the object description was compiled into the response
* A ‘modification timestamp’ that denotes when the object was last modified

The current timestamp can be stored by the Agent and compared to that returned on the next query. If the two current timestamps are the same, nothing has changed in the entire response. The Agent need not look further for differences in the two descriptions.

Should something have changed, the Agent needs to compare the modification timestamps for each object with that which it stored from the previous query. If the modification timestamp is more recent (higher numerical value) then that object has been updated, and the Agent needs to update the Agent database with the new object details. If the changes impact related objects also known to the Fabric Manager, the FM should have modified those objects as well. For example, if one end of a link goes down, the other end of the link also goes down, and the FM should updated both components’ Ports states accordingly. The Agent should not have to make assumptions about such hardware dictated interactions.

However, if the Agent detects a change also impacts a logical object which the FM does not track, the Agent may need to update related Agent data structures accordingly.

If the OFMF has subscribed to change Events on any updated objects, the Agent shall issue such an Event to the OFMF EventService.

#### Dynamic Update Process Flow for Agent – FM Interface

The PoC implementation used simply polling by the Agent to detect dynamic updates to the Gen-Z fabric nodes, links, or resources allocations and bindings. A proper reference implementation will use a simple Event driven flow, wherein the Agent will register an Agent URL to which the FM may send change notification Events. The Agent then can query the FM’s topology and resource APIs.

#### Dynamic Update Actions and Guidelines

The aforementioned timestamps enable the Agent to discern several different ‘changed’ scenarios. These are listed below, along with any restrictions on the FM reporting and with the agreed upon corrective actions to be taken by the Agent.

* New node present in the Gen-Z fabric topology
  + The Agent adds the new node to the Agent database, creates a new Redfish object, and sends an Event on the OFMF collection to which the new object will POST
* Existing node no-longer present in the Gen-Z fabric topology
  + The Agent flags the node as disabled, but does not delete it from the Agent DB
  + The Agent sends the appropriate Event to the OFMF
* New resource binding present in the Gen-Z ‘resources’ list (new Memory Chunk, a new Connection, or both)
  + The Agent adds any new Connections or Memory Chunks to the Agent database, creates any new Redfish objects, and sends an Event on the OFMF collection(s) to which the new object(s) will POST
  + Health monitor apps may subscribe to new Connections or Memory Chunks created by the Agent in response to an FM Change Event. In the OFMF environment, new Connections and Memory Chunks are expected to be created by the OFMF Clients, not as surprise additions from the FM.
* Existing resource binding no-longer present in the Gen-Z ‘resources’ list
  + The Agent flags the Connection, Memory Chunks, or Endpoints as disabled, but does not delete any objects or data from the Agent DB
  + The Agent sends appropriate Events to the OFMF per subscriptions
* Existing node parameters have changed: (modified, added, missing)
  + The Agent updates the Agent DB with the union of the new parameter set and the existing set in the Agent DB
  + The Agent sends appropriate Events to the OFMF per subscriptions
* New link present in the Gen-Z fabric topology
  + The Agent updates the associated endpoint nodes in the Agent DB
  + The Agent sends appropriate Events to the OFMF per subscriptions
* Existing link no-longer present in the Gen-Z fabric topology
  + The Agent flags the associated Ports and PHYs of the endpoint components as disabled, but does not delete any objects or data from the Agent DB
  + The Agent sends appropriate Events to the OFMF per subscriptions
* Existing link parameters have changed: (modified, added, missing)
  + The Agent updates the Agent DB with the union of the new parameter set and the existing set in the Agent DB
  + The Agent sends appropriate Events to the OFMF per subscriptions
* Node or link modification timestamp updated, but no other difference detected in Agent data
  + The Agent updates the stored modification timestamp
  + The Agent does not issue any object updated Events
  + The FM is allowed to issue such modification timestamps

## Example Management Use Cases and Associated Bounce Diagrams

These may be out-of-date; TBD

Here we will walk through this bounce diagram that briefly describes a FAM Manager client creating a new Memory Chunk and then binding it to some specific host. The following diagram applies to two use case descriptions.



### FAM Mgr Creates Memory Chunk

|  |  |
| --- | --- |
| Use Case Description | * FAM Mgr creates Memory Chunk |
| Actors | * Memory Resource Manager (FAM Mgr), OFMF & Agent |
| Description | * FAM Mgr client requests allocation of a region of Fabric Attached Memory * FAM Mgr collects proper inventory, creates the logical memory region from physical resources, and returns the URI of the resource, in this case a Memory Chunk |
| Comments | * FAM request is satisfied by creation of one Memory Chunk |
| Input Data | * Size of memory region * Type of memory * Attributes   + Access Types, addressing granularity   + Read and Write Permission capabilities   + Sharing / isolation requirements   + Self-encryption requirements   + QoS, HA, RAS requirements * Affinity requirements? |
| Preconditions | * FAM Mgr is in-service, has sufficient free pool of physical FAM resources * Fabric is up and OFMF is actively managing it |
| Postconditions | * OFMF Redfish tree contains the description of the Memory Chunk, and any other objects required to implement the Memory Chunk * Fabric Managers’ (Agents) fabric models contain the appropriate subset of resources * Actual fabric hardware state matches the OFMF Redfish description and desired fabric functionality is enabled |
| Trigger | * FAM Mgr is passed a request to create a logical memory region from its client, a Composition Manager (CM) |
| Normal Flow  (single chunk allocated from free list) | FAM Mgr Allocates the requested logical memory region   * Parse logical memory requirements * FAM Mgr locates sufficient, unused memory capacity on a single FAM device * FAM Mgr chooses to create a new, single Memory Chunk from a multi-chunk, multi-tenant endpoint already modeled as multiple memory chunks * FAM Mgr POSTs Memory Chunk to OFMF   + OFMF POSTs Memory Chunk to OFMF’s Redfish resource tree     - Linked to appropriate FAM Mgr event collection / management agents     - Linked to appropriate fabric Endpoints   + OFMF creates and sends an appropriate Redfish Memory Chunk request to Agent     - Agent updates its fabric-specific description of the impacted fabric resources (specific media controller and media resources-in-use descriptions)     - Agent does not alert Zephyr Fabric Manager since memory allocation (in this case) doesn’t require changes to fabric state, just the bookkeeping at the Agent     - Agent returns success and the original Redfish request from OFMF updated with any fabric specific details the Agent is programmed to pass up to OFMF   + OFMF returns all objects created (just one Memory Chunk in this case) and any errors or other changes * FAM Mgr: POST/PATCH resulting Memory Chunk to its own FAM Mgr Redfish data base, if appropriate (not shown on diagram) * FAM Mgr: Return success and URI of allocated logical memory region to caller (CM) (not shown on diagram) |
|  |  |

### FAM Mgr Creates a Connection Between a Host and a Memory Chunk

|  |  |
| --- | --- |
| Use Case Description | * FAM Mgr Binds logical memory region by creating a Connection |
| Actors | * Memory Resource Manager (FAM Mgr), OFMF & Agent, Fabric Manager (Zephyr), Host OS fabric management layer (Llamas) |
| Description | * FAM Mgr requests OFMF create a Connection from server to a logical memory region it created as a Memory Chunk * OFMF updates internal Redfish fabric model, and requests Agent make the appropriate changes to the fabric |
| Comments | * Normally the FAM Mgr would not trigger a Connection, but in this example it simplifies the bounce diagrams |
| Input Data | * User provided handle to predefined logical memory region (the Memory Chunk) * Attributes of the binding   + Access Types, addressing granularity   + Read and Write Permission capabilities   + Sharing / isolation requirements   + Self-encryption requirements   + QoS, HA, RAS requirements * IDs of initiator(s) (host) for connection |
| Preconditions | * Memory Chunk is previously defined * Fabric is up and OFMF is actively managing it * Cluster is up and running * Initiators (servers and accelerators) are booted and local node services are in communication with the Fabric Manager (FM) |
| Postconditions | * OFMF Redfish tree contains the description of the logical memory region, and all Redfish connections that result from binding a logical memory region to the designated initiators * Fabric Managers’ (Agents) fabric models contain the appropriate subset of resources * Actual fabric hardware state matches the OFMF Redfish description and desired fabric functionality is enabled |
| Trigger | * FAM Mgr is passed a request to bind a logical memory region to initiators within a cluster |
| Normal Flow | FAM Mgr receives the access request to a logical memory region (not shown in diagram)   * FAM Mgr Parses the request (not shown in diagram)   + validates requesting client has sufficient permissions to make the request   + validates requesting initiator is member of an appropriate OFMF cluster (zone)   + validates the requested initiator has appropriate access permissions within the zone * FAM Mgr POSTs relevant objects to OFMF   + Evaluate existing connection policies for match to requested connection details     - POST new connection policy object if required (not shown)   + Evaluate existing initiator endpoint groups     - POST new initiator endpoint group if required (not shown)   + POST connection to Memory Chunk to OFMF     - OFMF reviews changes required to OFMF Redfish resource tree     - OFMF calls fabric specific Agent to POST Connection     - Agent updates its internal fabric data, looks up FM IDs for the Host and the Memory Chunk     - Agent calls Zephyr Fabric Manager with proprietary request (add\_request) that binds the Memory Chunk to the given Host     - FM updates internal tables and configures the fabric to allow Host to access the given memory range on the given device     - FM returns success when fabric changes are complete,     - FM does not wait for Host to successfully incorporate the new memory capacity     - Agent waits for successful response from FM before returning success response to OFMF along with any fabric specific details it wants added to the Connection object   + OFMF finalizes update of Connection into the OFMF’s Redfish resource tree     - OFMF will update impacted Redfish objects as needed       * New Connections are tracked in related Redfish components   + OFMF returns success to FAM Mgr along with URI and Connection object * FAM Mgr may POST/PATCH resulting Connection(s) to FAM Mgr Redfish data base, if appropriate (not shown in diagram) * FAM Mgr may Return success and URI of Connection to (not shown in diagram) |