# General Overview of Open Fabrics Management Framework (OFMF)

## Purpose and Objectives

Today’s HPC clusters and cloud computing environments are running increasingly diverse and dynamic workloads. The industry is seeing more numbers and types of messaging and storage fabrics, and in the near future there will be memory semantic fabrics that enable disaggregated compute and memory resources to be shared across the fabric. The diversity of resources that can be ‘composed’ into a logical platform to execute workloads will rival the best equipped single enclosure server.

Today’s orchestration tools and workload managers do not deal well with multiple fabrics, and few, if any deal with disaggregated memory residing in memory appliances directly accessible from the fabric. HPC clusters have a variety of high speed networks which connect the mainly homogeneous cluster members. Each fabric has a proprietary fabric manager that is largely independent of the workload manager that schedules a few large jobs to run on a few large partitions of the cluster. Cloud computing environments are hosted on mostly homogeneous servers connected by high speed Ethernet. Software defined networking and storage capabilities are available as disaggregated resources to the virtual machines which execute the workloads. These orchestration tools and resource managers are tuned for an Ethernet underlay interconnect, possibly running different flavors of overlays. These tools each manage the resources they control in mostly proprietary manners.

The diagram below depicts the problem facing today’s fabric administrators: There is an explosion of fabrics, resources, and clients, yet no common fabric manager interfaces and fabric models available. <replace with rat’s nest version>



This problem is similar to that of the storage business before SNIA and DMTF offered a common storage fabric model based on Redfish. The industry needs an abstract model of a generic fabric and generic resources on the fabric to use as a unifying API and management methodology. This is the goal of the Open Fabrics Management Framework working group of the Open Fabrics Alliance. <use the matching figure from rat’s nest>



Without the OFMF every tool and every middleware library provider potentially has its own model of the fabric and fabric resources, and needs a unique call to a specific fabric management stack for each different fabric and/or resource supported.

With the OFMF every client calls a common fabric service to manipulate a Redfish model of the fabric resources. When different clients wish to communicate with each other about fabric resources (for example, a memory pool manager informing the orchestration manager of the location of 20 TB of Fabric Attached Memory needed by a workload) they pass details about Redfish objects and / or URI’s to specific Redfish objects. When administration tools (also clients of the OFMF Redfish Services) need to make configuration changes to the fabric such clients also manipulate the Redfish fabric objects to indicate the desired configuration.

OFMF triggers fabric specific providers to make the actual changes in the fabric devices.

## 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:



#### 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 URIsgiven 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.

#### 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 provider.

Not all client requests will require the OFMF to interact with the fabric provider. Many requests (for example, a request to registering 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 provider. 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.

The OFMF Services are also presented to the affected fabric managers through fabric-specific providers as a Redfish Service API.

#### Providers

Fabric-specific providers 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. Providers 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 ‘create connection’ between Host A and Memory B.

Fabric-specific providers 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 provider 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 provider is possibly the only entity that knows both namespaces.

Like the OFMF, a provider 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.

#### 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 fabrics.)

A major assumption of the OFMF Architecture is that any given Fabric Manager can configure the fabric components in a safe manner (safe for data, safe for the hardware, safe for hosts to boot without attacks from the fabric, safe from fabric namespace collisions, etc) without the presence of an OFMF and / or the appropriate Provider. The corollary assumption is that any given FM can be configured with policy flags for a variety of default configurations in the absence of an OFMF or Provider, and that one of the default config policies is to leave the fabric unusable until a Provider gives the FM a configuration to use.

#### 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 Provider, or each may have its own Provider. The Provider 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 Providers* 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 Provider is presenting one fabric-specific API to the OFMF Services, that one Provider is responsible for aggregating the namespaces and resources for all FMs reporting through it. If multiple Providers are communicating with the OFMF, there will be one Fabric Instance for each Provider. Clients will see multiple ‘fabrics’ in the Redfish Service.

In the case of multiple fabrics of the same type being reported to the OFMF, the OFMF will not aggregate the resources into one ‘greater’ fabric. If the administration wishes the OFMF Redfish Service to model multiple fabrics, an administration client must do such aggregation of the associated Redfish objects to create and populate an aggregated Fabric Instance.

Thus, the fabric administration is ultimately responsible for aggregating the resources of multiple FMs into a single ‘fabric instance’ as modelled by the OFMF. This can be accomplished by doing such aggregation in the Provider layer, or through manipulations of the OFMF Redfish models at the client level. Aggregation of resources from multiple fabric Providers into one ‘Fabric Instance’ object is not within the scope of the OFMF’s duties.

## The Role of Redfish

The OFA chose to use Redfish Objects and Schema to meet two major objectives of the OFMF API:

1. Abstraction of detailed fabric components to a set of common fabric ‘resources’
2. Control of abstracted resources translated into control of detailed components

A Redfish Overview

To understand how Redfish enables the OFMF API to meet the above objectives let’s take a very quick tour through the Redfish fabric model. The following figure shows the fabric centric view of a computer system with 2 Fabric Adapters connected to a fabric switch.



In a nutshell, Redfish models a fabric as a set of Redfish objects as follows:

* The fabric itself (the physical connectivity through which packets flow) is modeled as Port objects and Endpoint objects.
* Physical fabric connections (eg. Cables) are always between ‘Ports’
  + Note the two Fabric Adapters have 1 or 2 physical ports, and each has one Port physically connected to the single fabric Switch. (Red links are physical links. The red highlighting is NOT part of the Redfish policy for such diagrams; they were added by the authors.)
* Ports are logically traced back in the Redfish model to the fabric models of the devices that drive the ports via bookkeeping pointers in the Port object.
* Fabric devices trace back to physical or logical infrastructure (what controls the fabric device) AND to the ‘Endpoint’ object associated with the fabric device
  + Note the two Endpoints of the fabric model are linked logically to the two Fabric Adapters, not the Ports of the two Fabric Adapters, and not to the one computer system that controls them both.
  + The fabric shown has two possible initiators of traffic. The fabric has two fabric port addresses from which or to which it must route traffic. Redfish uses two Fabric Adapter objects to model these two initiators.
* Packets are routed by switches within the fabric using some form of the Port addresses, the destination device’s fabric address, or a combination of such.
  + Note that the fabric switch is modelled as a relay component on the fabric, but not as an initiator or target of traffic. Therefore, there is no Endpoint associated with the switch.

Once a fabric resource (is modelled as a related set of Redfish objects, clients can query the OFMF Service for details about these resources and request configuration modifications of those resources. Queries are made using HTTP GET protocols. Changes are made using HTTP PUT, PATCH, POST, or DELETE, or HTTP ACTION operations.

### Specific Redfish Fabric Objects of Particular Interest to the Fabric Management Problem

#### 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. Fabric Adapters have properties that describe specific features, resources, or actions of the actual Fabric Adapter that clients can monitor and control. Most properties are common across many fabrics, however some very fabric specific controls are made available in fabric specific structures. Resources available through the fabric adapter (memory or storage for example) or compute entities that access the fabric through a fabric adapter are 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 switches.

#### 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 physical topology/connectivity of the fabric can be discerned by tracing the Ports connectivity data.

#### 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.

Redfish Endpoints are almost objects of indirection as they contain pointers (Redfish URIs) to the objects that describe the various functionalities connected to the fabric as that one Endpoint. Switches are not Endpoints on a fabric unless there are resources within the switch which are the target or initiator of fabric packets.

* Memory Domains (associated with Fabric Adapters)
* Memory Chunks (associated with Memory Domains)
* Zones (and Zones of Zones) (establish fabric route permissions)
* Connections (between Endpoints and Endpoint Groups) (establish resource and endpoint specific permissions within zones)
* Address Pools (limiting Zone addressing ranges)
* Connection Policies (limiting Connection permissions)

Redfish fabric objects (switches, fabric adapters, memory domains, memory chunks, zones, connections, endpoints, etc) are intended to model resources that may access the fabric, be accessed from the fabric, route traffic through the fabric, or be ‘controlled’ from the fabric. In short, Redfish fabric objects have a representation associated with an Endpoint or a Switch Port on the fabric.

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. Host memory or host storage that is made available to other entities on a fabric through the Host’s Fabric Adapter can be reported to the OFMF by the fabric’s Provider, or by an administrative client (which could be a Provider associated with another Fabric within the OFMF tree).

The focus of the OFMF Redfish Service is to model Redfish fabric objects and the resources they represent. Resources such as the CPUs present on a server which is attached to a fabric via a fabric adapter may not be part of the OFMF Redfish model if the fabric’s FMs do not have visibility of the system behind the Fabric Adapter. FMs which explore their fabric and take inventory of the fabric-touching resources may not have the ability to query about what is behind the adapter. If it isn’t visible to the FM, it isn’t managed by the FM, and the OFMF representation can only be ‘placeholder’ objects that anchor such a Fabric Adapter into the standard Redfish resource tree.

The OFMF will build such placeholder structures as necessary as it retrieves (GETs) fabric inventory information from the fabric Provider. Subsequent details about such objects are PATCHed to the object when they are learned. Providers are encouraged to send any such details they know when the OFMF queries the Provider about them.

## The OFMF Responsibilities

The following block diagram roughly describes the functional blocks required of the OFMF.



Based on the above conceptual block diagram:

### The OFMF is/does

* Manage the Redfish file system, creating files and directories as needed, using standard OS file system APIs
* Manage the runtime Redfish resource tree data structures and keep them in sync with the Redfish file system
  + Typically there will be an internal data base which holds the working data for the Redfish objects. The Redfish file system is the persistent form of this Redfish service tree.
  + The original PoC OFMF simply kept all Redfish data in the filesystem, which required re-loading each object as it was needed for each HTTP request processed.
* Advertise its Redfish Services to clients using SSDP NOTIFY protocols, if enabled
  + Architecture calls for OFMF to advertise its Redfish Services (if it chooses to do so) after it has pre-configured the Redfish Service Root and populated it with known resources.
* Respond to clients seeking its Redfish Service using SSDP
  + Architecture calls for OFMF to respond to SSDP M-search requests with a to-be-named-later-service-type
* Retrieve the Providers’ capabilities structures
* Parse the client’s standard HTTP Redfish Requests, validate the URIs, adjust any non-standard requests as needed to pass the request to Redfish CMD execution
* Parse any OFMF specific HTTP Requests)
  + These are requests made through the client interface that target OFMF internal configuration, and are not meant to directly manipulate the Redfish objects
  + Think of these as OFMF administration requests
  + Examples: Reset Resource tree, Re-scan the Provider list, Re-scan a Provider’s inventory, Reconfigure the Service’s interfaces, Restart the OFMF
* Determine the impacted Redfish objects and process them as needed before responding the HTTP Request is complete.
  + A single PATCH, POST, or DELETE of a single Redfish fabric object may impact many other Redfish objects.
  + OFMF Architecture dictates that the OFMF Redfish Services maintain consistency among related Redfish objects.
    - For example, POSTing a fabric Connection is a request that requires the OFMF to update the ‘Connections’ properties of the initiator and target Endpoint objects.
  + Extracts the target URI(s)
  + Retrieves the specific objects
* Calculate fabric changes required to complete the Request and determine the associated requests to send to Provider(s)
  + Simple GET requests may require the OFMF to send a copy of the GET request to the fabric Provider, depending upon policies and processes established for a given OFMF – Provider combination. In general, OFMF Architecture requires a GET be satisfied by the OFMF Redfish resource tree unless there is a pending notification of change from the Provider or the OFMF tree has a potential to be out of date.
  + Any other requests are evaluated by the OFMF for possible impacts to Provider state or actual fabric component state. If the impact is possibly non-zero, the OFMF will relay the original request to the impacted Providers.
  + If the original client request impacts multiple Redfish objects, the OFMF is responsible for creating HTTP requests for any such additional Redfish objects and sending them to the appropriate Providers as well.
* Track the interactions state for each Provider
  + The OFMF is responsible for serializing any request streams sent to a single Provider, if necessary.
    - It is beyond the scope of OFMF Architecture to establish ordering rules across all the potential Providers
  + The OFMF is responsible for serializing different request streams sent to multiple Providers 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 Providers, the OFMF is responsible to coordinate the sequencing of those two request streams.
* Synchronize Redfish model to Provider state
  + Obviously, any changes that need to be sent to the Providers must be acknowledged by the Providers with an appropriate response before the OFMF can commit the changes to the OFMF Resource tree
  + Since the HTTP interface is asynchronous across multiple clients, the OFMF is responsible (as is any Redfish Service provider) for presenting a consistent Redfish model at all times.
* Locate Providers
  + OFMF Architecture declares the OFMF will listen for Providers searching for it
  + TBD: how to filter and validate Providers
* Query the inventory of all enabled Providers which pass validation
  + OFMF Architecture declares the OFMF sends queries to the Providers to obtain their inventory and current state/configuration of their fabric
  + TBD: the default tree query algorithms
    - Does OFMF search every Redfish path from its default Service Root?
    - Does OFMF use any ‘expanded’ GETs?
      * Doing so implies a corresponding requirement on the Provider to support such options
* Configure the Initial fabric resource tree and populate it with objects retrieved from Providers
  + The Provider supplies the actual Redfish objects of the fabric when queried by the OFMF
  + The OFMF walks a default Provider Service Root issuing GET requests against the default System, Fabrics, Chassis, and other collections which are specified in the OFMF Architecture (and to which the Provider must respond)
  + The OFMF creates any required subordinate collections when an appropriate Redfish object is returned by the Provider
    - example, if the provider returns a ‘System Instance’, the OFMF must create the subordinate Fabric Adapters collection and query the Provider for any instances of such
    - example: if the Provider returns a Fabric Adapter, the OFMF must create the subordinate Ports collection and query the Provider for any instances of such
    - TBD: if OFMF Architecture explicitly lists the subordinate collections that must be created and when they must be queried on the Provider to populate them
    - NOTE: The PoC demo did not use this OFMF-pulls-inventory scheme. The ‘push’ scheme was simpler, so we haven’t defined the algorithms for the ‘pull’ scheme yet.
* Create a single Fabric Instance for each Provider, unless Fabric Aggregation is enabled
* If Fabric Aggregation is enabled, consolidate the Redfish models from multiple Providers under one Fabric Instance (with one URI namespace) and maintain the appropriate URI namespace mappings between the OFMF Service root namespace and the individual Providers’ URI namespaces.
  + ALL clients of the OFMF Redfish Services see only the OFMF URI namespace.
  + Each Provider interfacing with the OFMF has its own URI namespace. This Provider URI namespace may or may not be a proper subset of the OFMF namespace. The OFMF is responsible for maintaining the mapping from OFMF URIs to Provider URIs.
* Manage data base of Provider state and status
  + Client requests which cannot be satisfied due to the current state of the impacted Providers shall be refused, reason <TBD>
  + Example: Reject any client requests referencing objects associated with a fabric if these requests arrive before the associated fabric Provider inventory operation is complete.
* Handle Redfish events from all Providers
  + TBD: The list of Redfish events supported
  + For example: The OFMF shall register for events and support appropriate processing for the following changes in the Redfish fabric objects (not a complete list):
    - Addition or deletion of fabric objects
    - Changes in capacity of storage and memory devices
    - Changes in health of fabric objects or fabric access to objects
  + The OFMF shall query the associated Providers as necessary to determine which properties of which objects must be updated to reflect the changes inferred from such events.
    - IE, the Provider is only going to signal events, the OFMF must query specific object details to determine the latest state / status of any impacted Redfish models.
* Provide Redfish Event services to clients
  + Client registration for supported Redfish events
  + Generate appropriate events to Clients in response to
    - Changes made to the OFMF Redfish models by other Clients
    - Changes made to the OFMF Redfish models as a result of OFMF processing
    - Events received from Providers that translate to Redfish Events for which Clients have registered
  + Registers for Provider-specific events, and handling them according to standard Redfish Event protocols
    - If there are Provider-specific events which do not map to Redfish events, there may be Provider-specific event extensions supported by the OFMF to enable clients to receive them

### The OFMF is-not / does-not

* Broadcast its presence to Providers
  + - I think we decided NO on this one?
* Search for clients

## The Provider Responsibilities

The Provider 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.



### The Provider Is / Does

* Namespace translation between the OFMF’s Redfish namespace and the Fabric Manager’s (potentially) proprietary namespace
  + Redfish IDs and URIs, but FM’s may use something entirely different
  + TBD: how the Provider & OFMF establishes System IDs, Fabric Instance IDs?
  + Provider MUST create or obtain (from the FM?) a fabric-unique FM ID for each unique object it reports to the OFMF
  + Provider MUST maintain all mappings of FM ID to Redfish ID
* Initial Inventory extraction from the FM and posting it to a Provider-specific internal data base
  + Provider / FM format for initial inventory and subsequent updates are not defined by OFMF Architecture
* If enabled, Aggregate multiple FM inventories into one fabric namespace
  + Large fabrics (such as Gen-Z) may have several FM instances that need to be federated, with each FM’s ‘subnet’ integrated into a larger, single fabric namespace
  + A single Provider might be supplied that would do this aggregation before announcing the fabric to the OFMF.
* Provide an HTTP Redfish API to the OFMF
* Track the status of the OFMF representation of the fabric
  + Is it current with FM’s claimed states and status?
  + Are changes pending in FM?
  + Are changes pending in OFMF?
* Parse the Redfish request from the OFMF
  + Evaluate the impact on the Provider’s model of the fabric
  + Evaluate the impact on the FM’s state of the fabric
    - Many OFMF Redfish requests are ‘bookkeeping’ in nature, and may not require any changes in actual fabric resource state or configuration
      * For example: creating a new memory chunk requires the Provider to register the Redfish IDs against a specific memory range of a specific fabric device. This binding of IDs to resource blocks may not require interaction with the FM or changes to the actual device.
* Parse any Provider specific HTTP Requests
  + These are requests made through the client interface that target Provider internal configuration, and are not meant to directly manipulate Redfish objects
  + Think of these as Provider administration requests
  + Examples: Restart the Provider, sync with a Redundant Provider,
* Forward or create FM-specific commands as necessary
  + Example: creating a connection between a memory resource and a host’s Fabric Adapter is likely to require the Provider to alert the FM of the binding so the FM can set up access modes and mapping tables in the fabric hardware.
  + The FM’s command API
* Register for events from the FM
* Process events from the FM
  + Parse them, translate from FM namespace to Redfish namespace, map them to Redfish events, assess their impact
  + If an event clearly denotes a change to the Provider’s internal fabric representation and/or resource description (for example, a hot-add of a new resource), the Provider 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 Provider shall send the appropriate event to the OFMF.
  + Forward the created Redfish events to OFMF as appropriate

### The Provider Is-not / Does-not

* Poll the FM for health status and inventory changes
  + TBD: I think we said NO to polling
  + NOTE: the first PoC was to do this polling for dynamic inventory changes, but we haven’t yet implemented it
* Send requests or events to the Fabric Manager which have no impact on FM bookkeeping or fabric component states
  + Example: Requests from the OFMF for inventory details that are stored in the Provider’s data base may not require the Provider to refresh its view by querying the FM if the Provider/FM interface protocol allows this.

### Event Driven Communications

A brief description covering

* Providers send notifications of changes from the FM to the OFMF (hot plug events, errors, link transitions, OS requests for resource changes <these have to get to resource managers who are clients of the OFMF>
* OFMF sends queries to determine the nature of the ‘changes’
* OFMF sends queries to newly found Providers to extract inventory
* OFMF sends events to clients about changes to fabric objects IF the clients register for them
* Clients don’t send events to OFMF ? (Is this true?)
* Other notes about events?

## Example Management Use Cases and Associated Bounce Diagrams

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 & Provider |
| 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’ (Providers) 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 Provider     - Provider updates its fabric-specific description of the impacted fabric resources (specific media controller and media resources-in-use descriptions)     - Provider does not alert Zephyr Fabric Manager since memory allocation (in this case) doesn’t require changes to fabric state, just the bookkeeping at the Provider     - Provider returns success and the original Redfish request from OFMF updated with any fabric specific details the Provider 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 & Provider, 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 Provider 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’ (providers) 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 Provider to POST Connection     - Provider updates its internal fabric data, looks up FM IDs for the Host and the Memory Chunk     - Provider 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     - Provider 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) |