Fusion Management REST API Specification

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Revision | Author | Description |
| July 27, 2010 | 0.10 | Terry Martin | Initial revision (mainly URI & query concepts) |
| Nov 4, 2010 | 0.20 | Terry Martin | Updates reflecting the latest federation and resource manager architectures |
| Nov 9, 2010 | 0.30 | Terry Martin | Rolled-in some team review comments, and other minor changes/cleanup |
| Dec 7, 2010 | 0.40 | Terry Martin | Rolled-in several changes based on comments from general i7 review |
| Dec 21, 2010 | 0.50 | Terry Martin | Changes based on feedback from general i7 review, round 2 |
| Nov 17, 2011 | 0.60 | Geoff Schunicht | Updates based on ZAO meetings |
| Dec 16, 2011 | 0.61 | Geoff Schunicht | Additional updates for section 3. |
| Dec 20, 2011 | 0.70 | Keith Kelley | Updates for sections 3.4 and 3.5 |
| Feb 1, 2012 | 0.75 | Geoff Schunicht | Multiple updates on section 3, additional updates for messaging. |
| Feb 2, 2012 | 0.76 | Keith Kelley | Replaced section 3.5 (HTTP Status Codes) |
| Feb 3, 2012 | 0.77 | Keith Kelley | Replaced section 3.5 (HTTP Status Codes) |
| Feb 6, 2012 | 0.78 | Terry Martin | General cleanup & re-org of section 2 |
| Feb 8, 2012 | 0.80 | Terry Martin | Multiple updates & clarifications, plus re-organization of the overall doc structure |
| Feb 9, 2012 | 0.81 | Terry Martin | More updates to (new) section 3 |
| Feb 10, 2012 | 0.82 | Terry Martin | Updates to section 4: Rest Operations |
| Feb 14, 2012 | 0.83 | Terry Martin | Added details on Activities and Resource Lists. Cleanup after initial ZAO review |

Table of Contents

Revision History 1

1 Introduction 7

1.1 Objectives 7

1.2 Approach 7

1.3 Document Conventions 8

2 General Concepts 8

2.1 REST Web Service Interfaces 8

2.2 Fusion Components and Interfaces 9

2.3 REST Request Routing in Fusion 11

2.3.1 Routing Within an Appliance 11

2.3.2 Request Routing in a Federation (FUTURE – not V1!) 11

2.3.3 Routing and URIs 12

2.4 Fusion Resource Identification and URIs 13

2.4.1 Principles of Fusion URIs 13

2.4.2 Fusion URI Format 14

2.4.3 URIs as Resource Identifiers 15

2.4.4 Additional Resource URIs 15

2.4.4.1 Resource Schema/Meta-Data URIs 15

2.4.4.2 Resource Attribute Validation URIs 16

2.5 Fusion Interface Versioning 16

2.6 Fusion Resource Versioning and ETags 17

2.7 Fusion Authentication and Authorization 18

2.7.1 User Authorization 18

2.7.2 Trusted Component Authorization 19

2.8 Localization 20

3 Fusion Resource Models 20

3.1 Definitions 20

3.1.1 Resource 20

3.1.2 Entity 20

3.1.3 Data Transfer Object (DTO) 20

3.1.4 Data Access Object (DAO) 20

3.2 Common Resource Model 20

3.2.1 Base Resource Attributes 20

3.2.2 Managed Resource Attributes 21

3.3 Resource List Model 22

3.4 Common Collection Model (post-V1) 22

~~3.5~~ ~~Property Model~~ 24

~~3.5.1~~ ~~Property Types~~ 24

~~3.5.2~~ ~~Property Qualifiers~~ 25

3.6 Error Message Model 26

3.7 Activity Resource Model 27

3.7.1 Task State Model 28

3.7.2 Task Error Model 29

3.8 Resource Schemas/Meta-Data Model 29

3.9 Data Format Consistency 32

3.9.1 Standardized Units (e.g., for bandwidth) 32

3.9.2 Standard Timestamp Format 32

3.10 Attribute Name Consistency 32

4 Fusion REST Operations 32

4.1 Operations Overview 32

4.1.1 Create 33

4.1.1.1 Synchronous Create of a URI-Addressable Resource 34

4.1.1.2 Synchronous Create of a Non URI-Addressable Resource 34

4.1.1.3 Asynchronous Create 34

4.1.2 Read 35

4.1.2.1 Retrieve a List of Resources 35

4.1.2.2 Retrieve a Single Resource 36

4.1.2.3 Retrieve a Single Resource Attribute 36

4.1.3 Update 37

4.1.3.1 Synchronous Update 37

4.1.3.2 Asynchronous Update 37

4.1.3.3 Update of an Individual Resource Attribute 38

4.1.3.4 Updates via POST 39

4.1.4 Delete 39

4.1.4.1 Synchronous Delete 39

4.1.4.2 Asynchronous Delete 39

4.1.5 Custom Actions 40

4.2 Query String Format 40

4.2.1 Filter Parameter 40

4.2.2 Query Parameter 41

4.2.3 Sort Parameter 42

4.2.4 Count Parameter 42

4.2.5 Start Parameter 42

4.2.6 Fields Parameter (TBD: Not in V1?) 42

4.2.7 View Parameter (TBD: Not in V1?) 43

4.3 Document/Data Formats 43

4.3.1 General Data Format 43

4.3.2 Views (TBD: Not in V1?) 43

4.3.2.1 Default View 43

4.3.2.2 Summary View 43

4.3.2.3 Deep View 44

4.3.2.4 Documentation View 44

4.3.3 JSON Specifics 44

4.4 HTTP Headers 44

4.4.1 Required Request Headers 44

4.4.1.1 Accept:<media-type>[;version=<api-version>] 44

4.4.1.2 Content-Type:<media-type>[;version=<api-version>] 45

4.4.1.3 auth:<session-token> 45

4.4.2 Optional Request Headers 45

4.4.2.1 Accept-Charset:<charset> 45

4.4.2.2 Accept-Language:<language(s)> 46

4.4.2.3 If-Match:<ETag> 46

4.4.2.4 If-None-Match:<ETag> 46

4.4.3 Required Response Headers 46

4.4.3.1 ETag:<entity-tag> 46

4.4.3.2 Vary:Accept 46

4.4.4 Optional Response Headers 47

4.4.4.1 Location:<URI> 47

4.4.4.2 WWW-Authenticate:<challenge> 47

4.4.5 Entity Headers (both request and response entities) 47

4.4.5.1 Required Entity Headers 47

4.4.5.2 Optional Entity Headers 47

4.5 HTTP Status Codes 48

5 Fusion Messaging 50

5.1 State Change Messaging 51

5.2 Client Support 51

5.3 State Change Message Object 52

5.3.1 Change Type (enum) 52

5.4 Subscription Cleanup 53

6 Appendix A: Fusion Resource Types 53

7 Appendix B: Common Services & APIs 55

7.1 Appliance Setup Services 56

7.1.1 Install 56

7.1.2 Setup/Config 56

7.1.3 Update 56

7.2 High Availability Services 56

7.2.1 Cluster Configuration 56

7.2.2 Application Management 56

7.2.3 Backup/Restore 56

7.3 Security Services 56

7.3.1 Authentication Service 56

7.3.2 Authorization Service 56

7.3.3 Trusted Component 56

7.3.4 Session Management 56

7.3.5 Device Credential Manager 56

7.4 Logging Services 56

7.4.1 Audit Logs Service 56

7.4.2 Product Logs Service 56

7.4.3 Activity Tracker 56

7.5 UI Helpers 56

7.5.1 Index Service 56

7.5.1.1 Search/Lookup 56

7.5.1.2 Associations 56

7.5.2 Global User/Settings Service 56

8 Appendix C: Resource Managers & APIs 56

9 Appendix D: Examples 57

10 Appendix E: References 57

# Introduction

## Objectives

The Fusion REST APIs provide a consistent mechanism for interfacing with Fusion services and resource managers. This APIs is used both internally within the CIC appliance and exported for external use. It is expected the API will be used by customers and partners alike for creating command line interfaces (CLIs), custom applications, scripting and integration with other applications. Given the broad scope, it is important that the APIs provided by various Fusion services and resource managers are as consistent, so customers and partners (and internal developers) see this as a single, cohesive API. This not only presents our product as a well-designed cohesive unit, but it will greatly reduce the learning curve required for scripting and integrating with our environment.

The goals of this document are:

* Provide a common understanding of Fusion REST API goals and requirements across participating teams
* Ensure a consistent API across resource managers and common services in the Fusion environment, by providing a:
  + common reference specification for Fusion REST APIs which teams may reference
  + central reference for verifying the consistency of Fusion APIs in design and code reviews
* Provide an understanding of how Fusion REST interfaces will behave that customers and partners will use to integrate with the Fusion environment, post-release
  + Learning Products will likely use this specification as a reference for creating an official/external document in the future

## Approach

In order to meet the above goals, this document is broken into the following sections:

Section 2 of this document (General Concepts) provides some background into the general Fusion architecture and some of the context around which this API specification was formed.

Section 3 (Fusion Resource Models) describes the common resource models used in Fusion. This includes the models for common resource types, such as Error Messages and Activities. The models for other, RM-specific resource types are NOT defined here – they are defined in the API documentation for the component/RM that owns each resource type.

Section 4 (Fusion REST Operations) describes the basic REST API model that all Fusion services and resource managers are expected to follow. The intent is to specify a minimum set of requirements in order to meet the objective of providing a consistent & cohesive API across Fusion components, and to allow each component as much latitude as possible in defining the details of the APIs they provide.

Section 5 (Fusion Messaging) covers the common Fusion messaging model, including resources state-change messaging, which all RMs are expected to use and conform to.

Each Fusion resource manager and service that exposes a RESTful API will have separate SDK documents written to address individual service details. This will ensure the SDK remains as current as possible. This will provide a single document for Fusion developers, customers and partners to reference when integrating with the Fusion environment.

## Document Conventions

The following formatting is used to highlight what is not currently implemented and what is not planned for Fusion V1, etc.

* Important notes to developers
* Support not yet available in the infrastructure, but planned for V1
* NOT planned for V1 (documented here only as a reminder of future plans)
* ~~No longer planned in Fusion (only in to be clear that this has been removed)~~
* TODO’s & reminders to authors of additions/changes needed to doc

# General Concepts

## REST Web Service Interfaces

Fusion has a resource-oriented architecture and conforms to the REST architecture style with HTTP 1.1 providing the uniform interface, in other words Fusion implements a specific style of a RESTful web service. The resource has a central role in our model; all domain information, devices and control/management operations are defined in terms of a resource and its defined states. Every resource has at least one Uniform Resource Identifier (URI), or conversely every URI in a CIC domain identifies exactly one resource. Resources may be queried or manipulated using the four operations defined by the HTTP methods POST, PUT, DELETE and GET, see Table 1. Consequently every resource is addressable, represents a physical device or logical construct and may be manipulated using a uniform interface. A CIC API does not conform to an RPC style architecture and does not expose application defined operations or methods (for example it is not object-oriented with object methods nor does it support SOAP-based RPC). . In summary, the Fusion API is specifically designed around the REST architectural style, and SHALL be exposed through a web service endpoint implemented using secure Hypertext Transfer Protocols, version 1.1 (RFC 2616) technologies.

|  |  |  |
| --- | --- | --- |
| Method | Properties | Description |
| GET | Safe, Idempotent | Returns representation of resource |
| PUT | Idempotent | Update resource, Create specific resource (client supplied URI) |
| DELETE | Idempotent | Delete resource (no longer accessible using URI) |
| POST |  | Create new subordinate resource (server defines URI), append to an existing resource |

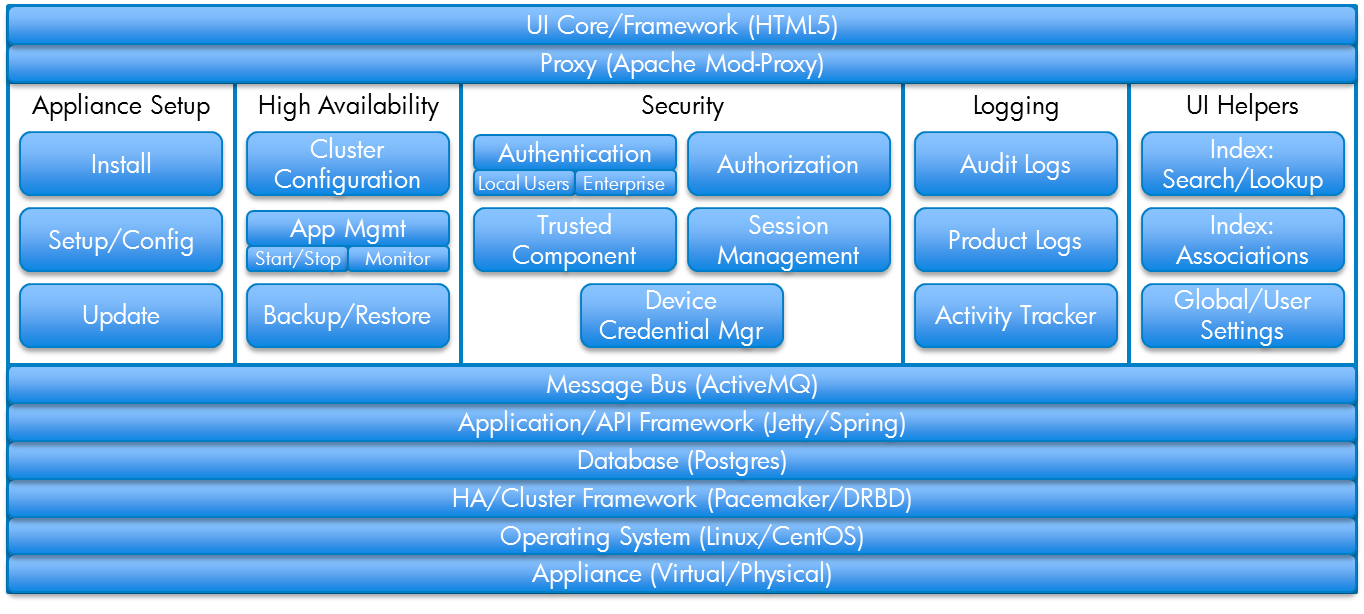
Table 1: Primary HTTP Operations

## Fusion Components and Interfaces

In general, the Fusion Management environment can be thought of as one or more management “appliances”, where each appliance is responsible for managing some set of resources (servers, storage and network resources, etc.). In other words, each of these resources is “owned” by a given appliance, and management of that resource is *ultimately* through that appliance (independent of the appliance that the client may be currently connected to).

Figure 1: Fusion Appliances and Resources

Each appliance runs a set of Fusion resource managers and foundation services. Each of these services has very specific responsibilities within the Fusion management environment: Resource Managers are responsible for managing a specific “class” of resources (physical servers, server profiles, storage, networking, etc.), and Foundation Services provide the framework and common functionality needed to create an integrated and consistent user experience across a diverse set of Resource Managers.

Figure 2: CI Foundation Services: The horizontal bars are the truly “foundational” components (tools & technologies that all Fusion components are built on), and the boxes in the middle section are the “common services” provided by the framework (typically exposing their own REST interfaces)

The primary interface exposed by Fusion components are REST APIs, however components also communicate via a messaging infrastructure called the State Change Message Bus: Components publish and subscribe to resource state-change messages on this bus. While this message bus is largely for internal communication between components, external clients (partner components and client scripts) may also tap into this bus in order to be notified asynchronously of Fusion resource state changes.

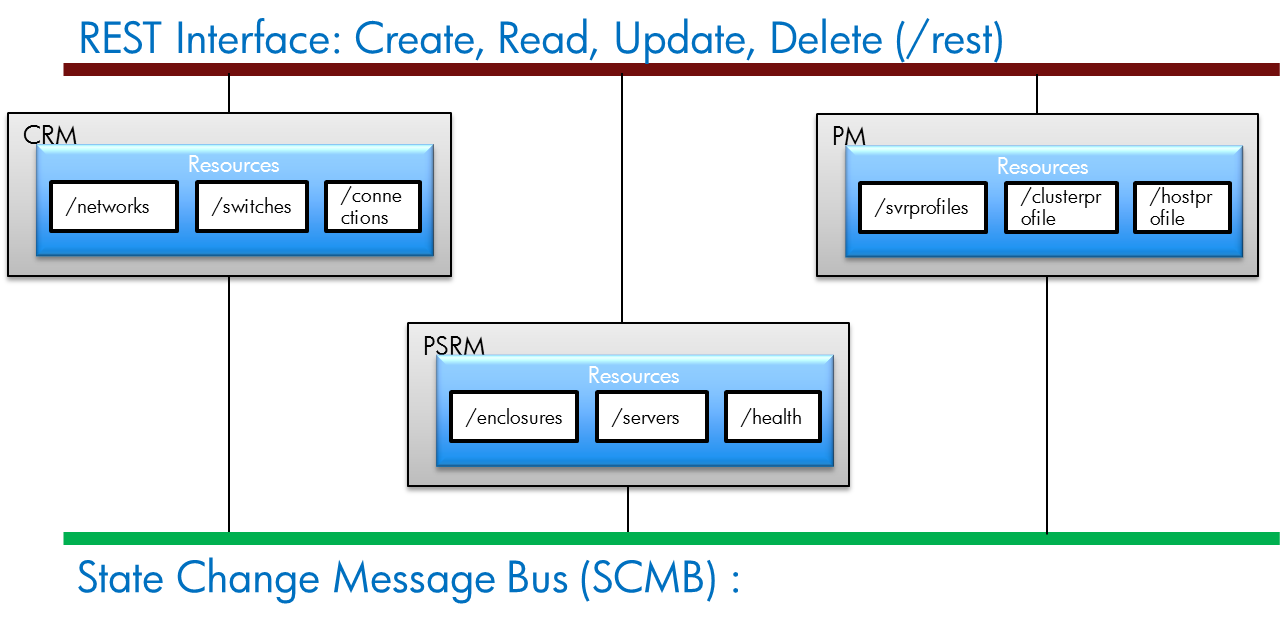


Figure 3: A sampling of the main resource managers in Fusion. Notice that each RM surfaces its APIs via REST and posts state change messages on the State Change Message Bus

## REST Request Routing in Fusion

In the resource-oriented model, clients interact directly with various Fusion “resources”, and the details of which service/component handles each resource is completely hidden from the client. In other words, Fusion URIs point to the resources and NOT the services that manage those resources. This is important because it allows us to change the underlying structure (the set of components/web applications that manage particular resource types) without affecting the external APIs.

### Routing Within an Appliance

Within a Fusion appliance, each Fusion component (RM or Foundation Service) runs as a particular web application, within a particular web container, listening on a particular HTTP port. From an external perspective, however, Fusion appears as a single, cohesive web application on the appliance, servicing requests for various URIs, corresponding to a set of Fusion resources.

This is accomplished via the Fusion Proxy: The Proxy handles all requests to an appliance (on https port 443) and re-directs those requests to the particular web application/port on the appliance, based on “resource type” specified on the incoming URI.

Figure 4: Example of request routing via the Fusion Proxy. In this case “perm” represents the name of the web application that handles “enclosures” resources on the appliance, and “port” represents the port# that perm’s web container is listening on.

**Note**: Fusion components configure the external->internal URI mappings for their component, along with which web container they run in, in the *CICWebContainers.xml* file (installed under */ci/etc/services.d* on the appliance)

**Note**: Last portion of the URI (after …:port/webapp) could and should be the same for external & internal.

### Request Routing in a Federation (FUTURE – not V1!)

All client interfaces to the Fusion Management appliance involve interactions with particular Resource Managers and Foundation Services running on particular management appliances. However, the client APIs hide the details of which particular service instance (Resource Manager or Common Service) on which particular appliance needs to be contacted, in order to perform actions and/or gather information on a particular resource. In the end, the client is able to manage all resources through a single connection to a single appliance, even if some/all of those resources are actually “owned” by services/resource managers running on a different appliance.

This is again accomplished via the Fusion Proxy, and resource URIs containing the information needed by the Proxy to route requests to the appropriate service or resource manager on the appropriate appliance. All client connections are really to the Proxy running on the appliance that the client is connected to. The Proxy then forwards the request to the appropriate service on the appropriate appliance, based on the URI of the request.

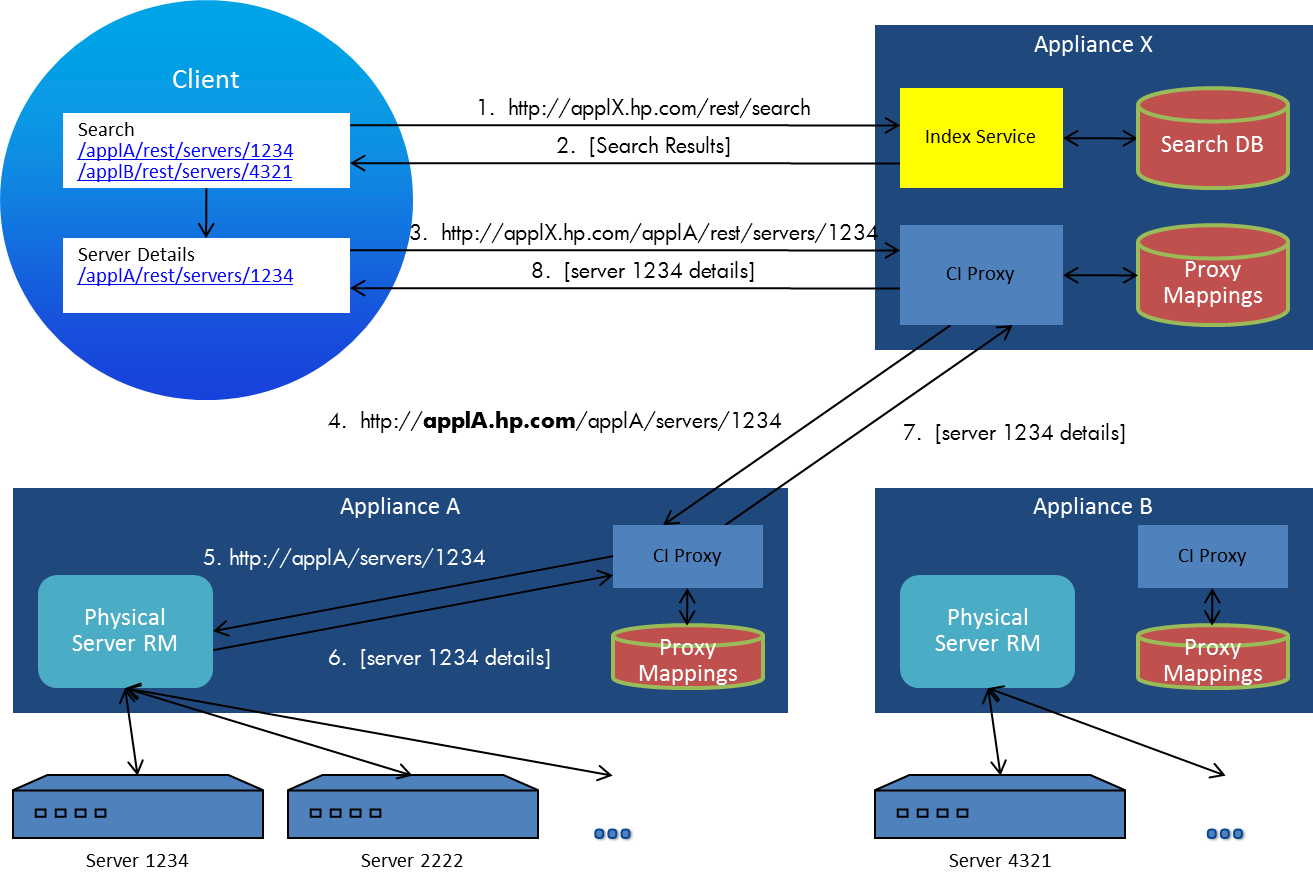


Figure 5: Request Routing in a Federation (proposed for future – NOT in V1!)

### Routing and URIs

What is important (at least for Fusion developers) to understand from the routing discussion, above, is that the format of a resource’s URI is critical, in order for this routing of requests by the Proxy to work. Basically, you can think of the URI as being made up of 2 parts. For example, in the following URI:

**https://appliancex.hp.com/rest/physical-servers/9876**

* The first portion of the URI (in **gray**, called the “connection”) represents the client’s connection to “some” appliance (really the Proxy on that appliance), and this connection is independent of the appliance that actually “owns” the resource.
* The second portion of the URI (in **green**, called the “resource-path”) identifies the actual resource being accessed, and is mapped by the Proxy to the internal port/web container & web application name of the component that handles the particular resource type indicated in the URI. These will be named using the resource type. The resource type should be plural, assuming there is more than one resource managed by the resource manager. Some examples are switches, physical-servers, users. See Appendix A: Fusion Resource Types, for the full list.

**Note**: In the future, with federation (post-V1), the resource path will likely include the ID of the appliance that manages the particular resource (e.g., “…/applA/…”), allowing the proxy to re-direct to that appliance. Details of this *future* format are omitted from this document, pending further investigation of federation, post-V1.

See the Fusion Resource Identification and URIs section, below, for more details and requirements on Fusion URIs.

## Fusion Resource Identification and URIs

From an external/client perspective, all Fusion resources are identified and addressed by URIs (and URIs only). The fact that many URIs will embed an *internal* resource ID (UUID, etc.) is immaterial from a client’s perspective. Resource URIs should never be parsed or constructed, and must be treated as opaque strings by clients.

### Principles of Fusion URIs

* All Fusion URIs point to “resources” (either a Fusion service, a resource or a collection of resources – all “nouns”)
* Any URI returned from a Fusion API should be considered an **opaque** identifier of a resource. The client should not need to modify or parse these URIs in any way, and there should be no need for a client to build a URI from scratch (other than adding query parameters, as needed).
* Fusion services and resource managers are ultimately responsible for defining the URIs to the resource’s they own. These URIs must adhere to certain requirements (as described in the [General URI Format](#_General_URI_Format_1) section, below), but it is up the service/resource manager to ensure that:
  + Each URI is unique (and any individual resource identifiers within the URI are “globally” unique to that resource type, in the event that resources from multiple environments are merged, etc.).
  + The URI for a given resource should never change: This is important because the URI is the resource’s “primary identifier” for that resource, and will be stored in multiple places, so changes to URIs can be expensive.

**Important**: The resource URI must be treated as an opaque string to the client, thus the client must never parse or create a resource URI based for a given resource instance (other than adding query parameters, as needed). For example, if a physical server has a resource path of /rest/physical-servers/A8US291325 the client should not need to understand what the string A8US291325 is or need to build the URI from other data.

### Fusion URI Format

The general format of a Fusion URI is:

**https://<appliance>:<port>/rest/<resource-type>/[<resource-id>]**

**/[<attribute> |<?query-string>]**

Where

|  |  |
| --- | --- |
| **https:** | Fusion will only use secure HTTP for communications |
| **<appliance>** | Is the cluster IP or hostname of the appliance that the client is connecting to |
| **<port>** | Is the configured Fusion Interface port (may be omitted for default port 443) |
| **“rest”** | Root of all REST APIs (appliance root URI points to the UI) |
| **<resource-type>** | The resource or service type name (plural for resources). See [Appendix A](#_Appendix_A:_) for a list of current Fusion resource types. |
| **<resource-id>** | The unique identifier of individual resource, omitting the resource-id on a GET will return all resources of the given resource type. |
| **<attribute>** | Optionally used to specify the name of an individual attribute or “meta” attribute for that resource |
| **<query-string>** | Optionally used to refine the list of resources to be returned and/or specify how data should be returned (e.g., sort order, etc.). |

The portion of the URI in **gray**, above, represents the client’s *connection* to the appliance. Since this connection is considered independent of the resource being managed, this portion of the URI is *not* part of the *unique identifier* of the resource, and is *not stored or returned* in Fusion APIs.

The portion of the URI in **green**, above, is the *resource-path*, identifying the specific resource or collection that the URI refers to. The format of this section may be defined by each Fusion service or resource manager, and could be considered an opaque string. For consistency, however, all Fusion components must use the format defined (resource-type ID followed by some sort of resource ID). As a MINIMUM, resource paths must *uniquely identify* a resource across all *instances* of the particular service (in the event the resource is migrated from one instance of the service to another) and these identifiers must be *stable* for the lifetime of the resource (avoiding problems due to broken links/stale URIs, etc.).

The **<attribute>** portion represents optional attributes or meta-attributes that may be supported by a given resource.

The**<query-string>** portion of the URI is optionally used to refine the list of resources to be returned and/or specify how data should be returned (e.g., sort order, etc.). See [*Query String Format*](#_Object_Paths), below, for more information. Note that the query string is NOT part of the unique identifier of a resource.

### URIs as Resource Identifiers

The URI is the primary unique identifier of Fusion resources. Note, however, that the “connection” portion of the URI (in gray) will vary, depending on the client’s connection, therefore this portion of the URI is *not* part of the *unique identifier* of the resource, and is *not stored or returned* in Fusion APIs. The remainder of the URI (the resource path, in green) is what *uniquely identifies* the resource, and this is what is returned in all Fusion APIs. The client must pre-pend the “connection” portion of the URI (protocol://hostname:port), based on the appliance the client is connecting to, in order to make direct API calls on these resources.

**Note**: Since in the future, Fusion resources may be accessed through multiple appliances, and routed by the Fusion Proxy to the appropriate appliance that owns/manages the resource; the connection portion of the URI is not part of the unique identifier of a resource. The unique identifier of a resource is the *resource-path* portion of the URI (in **green**, above)

For example, a POST may return the following URI in the Location header of the response (indicating the new resource created by the POST):

/rest/server-profiles/1234

Assuming the client is connecting through an appliance named “c-mgmt-a.hp.com”, the full URI needed to access this new resource directly would be:

https://ci-mgmt-a.hp.com:443/rest/server-profiles/1234

**Note**: For APIs that take a *resource URI* as an attribute, the client would simply use the original URI (without the connection portion) -- again, because the connection is not part of the *identifier* of the resource.

### Additional Resource URIs

To assist the clients with additional programmatic functionality each resource manager may support additional URIs.

#### Resource Schema/Meta-Data URIs

Resource managers should include resource meta-data that describes the resource itself. The URI for meta-data is well is a defined URI off the main path for a given resource manager. The URI is defined as:

https://<appliance>:<port>/rest/<resource-type>/**schema**

Where ***schema*** is the identifier used for meta-data for all resource managers with exposed APIs.

Information on the standard meta-data format used in Fusion can be found in the [Resource Schemas/Meta-Data](#_Resource_Schemas/Meta-Data) section, below.

#### Resource Attribute Validation URIs

In many cases the UI or client may want to validate proposed changes to a resource, prior to an update or create of the resource. One way this can be done is by querying for resource meta-data (see above) and validating attributes on the client side. In some cases, however, server-side validation is needed in order to absolutely ensure the proposed changes are valid (i.e., there’s a limit to what can be validated via meta-data).

This server-side validation of resource attributes may be done via a well known URI on a given resource manager. While this same validation should be done during create/update of the resource, this URI is used to validate without modifying the state of the resource in any way.

The standard URI for validation is.

POST https://<appliance>:<port>/rest/<resource-type>/**validator**

…where the request body includes the set of proposed changes to the resource: Either a single attribute, set of attributes or the full resource. Full resource validation is discouraged as a normal POST/PUT will have the same effect and will reduce the number of calls made.

There also may be cases where server-side validation of changes to a *specific instance* of a resource is needed (e.g., changes may be valid for one instance of a resource, but not another). In this case, the following URI may be *optionally* supported by certain components, for validation of a given resource instance:

POST https://<appliance>:<port>/rest/<resource-type>/**<resource-id>/validator**

The above validation requests will return the following on success:

* HTTP Status 200 (OK)
* Response Body: Includes the same set of attributes sent in the request (as a minimum)
  + Note: The representation of some attributes may be modified by the server, as part of the validation

…and the following on failure:

* HTTP Status 400 (BAD REQUEST)
* Response Body: [Error Message](#_Error_Message_Model) indicating which attribute(s) failed validation and why

## Fusion Interface Versioning

In order to support rolling upgrade of appliances, and to not require customer and partner integration scripts to be updated with each new version of the CI management tools, CI services MUST be capable of handling requests from clients that were written for an *older* version of the interface (the previous *major* version, at least). In addition, CI services SHOULD do their best to handle requests from clients written for a *newer* version of the interface – at the very least, services will fail gracefully when they are unable to handle a request from a newer client.

Interface versions are of the format ***<major-version>.<minor-version>***. The interface version need not change with each revision of the CI management tools (although it is likely that the interface version will be kept in sync with the tools version for clarity). The reasons for updating the interface version are:

* Attributes added or removed from one or more resource types
  + A “minor” revision, if this is the only change
* The semantics of modifying an attribute on one or more resources have changed
  + This results in a “major” change to the interface version

For the Fusion REST APIs, interface versions are specified in the ***version*** parameter of ***Accept*** and ***Content-Type*** HTTP headers (see the section for details on the format of the version parameter in these headers).

* Clients may specify the requested interface version# in the ***Accept*** header for GET requests
  + If accepted, the **Content-Type** header on the response will include the same version#
  + If the service is unable to handle the requested version (e.g., newer version than the service), the response will be 406 (NOT ACCEPTABLE).
* For PUTs and POSTs, the client specifies the interface version# in the ***Content-Type*** header
  + If the service is unable to handle the requested version (e.g., newer version than the service), the response will be 415 (UNSUPPORTED MEDIA TYPE).
* By default, if no version# is specified in the Accept/Content-Type headers, the most recent version supported by the service is assumed

## Fusion Resource Versioning and ETags

Fusion will leverage ETags to support resource versioning and optimistic locking of resources. An ETag is an opaque value that will enable a client to know that a resource has changed. Specifically the ETag for a given resource must be different if the information returned by a GET request is different.

Options for creation of an ETag include, a monotonically increasing integer value, for example a SQL Sequence is a good example. Another option would be to use the BaseResources modified timestamp, assuming millisecond resolution.

A GET request for a single resource will include the ETag of that resource in the response body and in the *ETag* response header. In addition, a synchronous POST or PUT operation must also include the ETag in the response header in order to avoid a subsequent GET just to obtain the current ETag. A GET of multiple resources will return the ETag in the response body for each of the resources returned, the ETag header should not be present.

To update a single resource, the PUT request header MUST include the If-Match header set to the ETag value for that resource. The following behavior of the resource manager should be followed:

* Return HTTP Status 403 (FORBIDDEN) – if the Request Header does not include the ‘If-Match’ field
* Return HTTP Status 412 (PRECONDITION FAILED) – if the provided ETag does not match the current resource value
* If the provided ETag matches the current resource value RM attempts to update the specified resource
* If the provided ETag is ‘\*’ RM attempts to update the specified resource. **Use of ‘\*’ is strongly discouraged**
* ETags in the Request Body are ignored

Client caching is not yet required for resource managers, however if a resource manager is to support client caching it must use ETags. Client caching is supported with the If-Non-Match HTTP header in a GET request. A resource manager can compare the value of the If-Non-Match header with the current resource, if there is no change the resource manager must return a 304 (NOT MODIFIED) status with an empty response body.

**Note**: Full and partial GETs on the same resource must return the same ETag

## Fusion Authentication and Authorization

In order to prevent rogue processes from accessing/modifying Fusion resources, all APIs **must** be authorized. The authorization mechanism employed in Fusion APIs involves passing a valid session token in the ***auth*** header of each request (see the section, below). This token is obtained in one of 2 ways:

1. Users “log-in” via the Authentication Service, obtaining a session token in the response
2. Components may obtain a Trusted Component token for use when making internal calls to other components (possibly on behalf of a logged-in user)

In both cases, the session token is validated by the server, either by direct REST calls to the Authorization Service, or by taking advantage of built-in Spring Security mechanisms (which ultimately authorizes via the Authorization Service).

### User Authorization

In Fusion, user authorization is based on:

* The ***type***/category of the resource being accessed/modified
  + For example ***enclosures*** or ***networks***, etc.
* The type of ***operation*** requested
  + For example ***create***, ***read***, ***update*** or ***delete*** of the resource
* The current user’s ***role***(s)
  + For example, ***Server Admin***, ***Network Admin***, etc.

For example a ***Server Admin*** may have permissions for ALL operations (***create***, ***read***, ***update*** and ***delete***) on ***enclosures***, but may only have ***read*** permissions on ***networks***.

**Note**: In the future there will be a 3rd construct: ***organizations***, which will be used to sub-divide the specific resources that a given user has permissions on. For example a ***Server Admin*** in the ***Atlanta*** organization may have read/write access to all servers in the ***Atlanta*** organization, and read-only access to servers in the ***Austin*** organization.

This may or may not be part of Fusion v1.0, and in any case, this functionality will be encapsulated within the Authorization Service (with the possible exception of needing to add an ***organization*** field to the [Base Resource](#_Base_Resource_Model_1) model).

### Trusted Component Authorization

Independent of user authorization, there needs to be a mechanism for internal components to be authorized for API calls to other components. There are 2 cases where this is needed:

1. As part of servicing a user request, components may need to make internal calls to other components – and the current user may/may not be authorized for those calls.
   * An example may be applying a server profile (a typical ***Server Admin*** permission), that in turn requires the creation of network connections (possibly not a ***Server Admin*** permission)
   * In order to keep our authorization model simple, we entrust the internal component to perform whatever tasks are necessary to satisfy a user request, even if the user doesn’t have explicit permissions to perform those tasks themselves
2. Fusion components often need to make calls to other components as part of on-going operations/background processing, which are not associated with any logged-in user
   * An example here would be when an RM needs to update the Index Service due to a change that resulted from an internal even (e.g., Health notification)

Components obtain a Trusted Component token via a Java (and soon C/C++[[1]](#footnote-1)) API that relies on a private key, known only to internal components on the appliance.

**Note**: In cases where a component is making internal requests o behalf of a user request, the user/login token must be “wrapped” in the Trusted Component token, in order to keep track of the user that the request is being made “on behalf of”. This wrapping is also accomplished via a Java (and soon C/C++) API call.

**Note**: A CLI has also been developed (for testing only) in order to obtain a Trusted Component token from the test head.

## Localization

TODO: Describe here how localization works in APIs: Basically that all messages returned from APIs will be pre-localized, based on the ***accept-language*** header of the request. Also talk here about the (future) common message repositories, etc.

# Fusion Resource Models

## Definitions

TODO: Add these definitions to glossary as well!

### Resource

### Entity

### Data Transfer Object (DTO)

### Data Access Object (DAO)

## Common Resource Model

In general, all Fusion resources are modeled as a list of attributes (name-value pairs). The following describes the attributes that are common across Fusion resources and collections.

Each specific resource type defines a set of additional attributes that are returned from a GET and expected for POST/PUT operations. Refer to the resource model documentation for the specific resource type for details on those additional attributes.

**Future**: Not yet implemented, and may/may not be in V1:

Note that the actual list of resource attributes returned from a GET request depends on a number of factors:

The requested “***view***” (see description of [Views](#_Default_Representation), below)

Any “***fields***” parameter sent on the request (see description of the [fields](#_Fields_Parameter) parameter, below)

### Base Resource Attributes

The following attributes are guaranteed to be returned for *all* Fusion resources, including things like *events* and *notifications* (which are not considered first-class/managed resources):

|  |  |  |
| --- | --- | --- |
| Attribute Name | Data Type | Description |
| uri | String | The canonical URI of the resource. As described above, this is the resource path which primary identifier of the resource in the Fusion Management environment |
| category | String | The resource category, used for authorizations (see [Authentication & Authorization](#_Fusion_Authentication_and), above) and resource type groupings (e.g., in the UI). For example “enclosures”, “server-profiles”, etc. See [Appendix A](#_Appendix_A:_) for a list of current Fusion resource categories & types |
| type | String | The resource type represented by the entity. For example, “blade” or “rackmount” (as specific types of the “physical-servers” category). This field can be used to determine the exact set of attributes expected/included in the entity/DTO (based on the documented resource models for the given resource type). See [Appendix A](#_Appendix_A:_) for a list of current Fusion resource categories & types |
| created | String | Date and time (in UTC, [ISO 8601](http://en.wikipedia.org/wiki/ISO_8601) format) when the resource was created |
| modified | String | Date and time (in UTC, [ISO 8601](http://en.wikipedia.org/wiki/ISO_8601) format) when the resource was last modified |

### Managed Resource Attributes

In addition to the Base Resource attributes, above, the following attributes are included for “managed” resources (essentially resources where state & status needs to be tracked in a common way). Attributes with a gray background are inherited from the [Base Resource model](#_Base_Resource_Attributes)):

|  |  |  |
| --- | --- | --- |
| Attribute Name | Data Type | Description |
| uri | **String** | The canonical URI of the resource. As described above, this is the resource path which primary identifier of the resource in the Fusion Management environment |
| category | String | The resource category, used for authorizations (see [Authentication & Authorization](#_Fusion_Authentication_and), above) and resource type groupings (e.g., in the UI). For example “enclosures”, “server-profiles”, etc. See [Appendix A](#_Appendix_A:_) for a list of current Fusion resource categories & types |
| type | **String** | The resource type represented by the entity. For example, “blade” or “rackmount” (as specific types of the “physical-servers” category). This field can be used to determine the exact set of attributes expected/included in the entity/DTO (based on the documented resource models for the given resource type). See [Appendix A](#_Appendix_A:_) for a list of current Fusion resource categories & types |
| created | String | Date and time (in UTC, [ISO 8601](http://en.wikipedia.org/wiki/ISO_8601) format) when the resource was created |
| modified | **String** | Date and time (in UTC, [ISO 8601](http://en.wikipedia.org/wiki/ISO_8601) format) when the resource was last modified |
| name | String | Display name for the resource (UTF-8 for internationalization) |
| description | String | Brief description of the resource (UTF-8 for internationalization) |
| status | String | Overall health status of the resource (for follow-the-red). Current possible values are OK, Major, Critical. |
| state | String | The current state of the resource. Each resource type may have different states to meet the needs of that resource. Some examples could be importing, configured, assigned. |
| ~~properties~~ | ~~List<~~[~~Property~~](#_Property_Model)~~>~~ | ~~List of properties specific to the particular resource/type~~ |
| ~~actions~~ | ~~List<~~[~~Action~~](#_Action_Model)~~>~~ | ~~List of actions available on the particular resource/type. Note that this attribute is only returned when the~~ ***~~listActions=true~~*** ~~request parameter is sent on the request~~ |

## Resource List Model

In order to support a common model for pagination of multi-resource lists, the following entity has been defined to be returned on all [multi-resource GET](#_Retrieve_a_List) queries (dubbed “wrapped-array” in some contexts):

|  |  |  |
| --- | --- | --- |
| Attribute Name | Data Type | Description |
| total | Integer | The total number of resources that *would be returned* from the query, if there was no pagination (or enforced resource limits) employed. |
| count | Integer | The *actual* number of resources returned in this *page*. Unless this is the last available page, this would equate to the value of the [count parameter](#_Count_Parameter) sent in the request. |
| start | Integer | The row/record number of the first item returned in the resources list, below. This typically equates to the [start parameter](#_Start_Parameter) sent in the request. |
| sort | String | The sort order used for this list of resources. Equates to the [sort parameter](#_Sort_Parameter) sent in the request, but included here to allow clients to store complete page information (including sort order & filters used to create the page) for future use. |
| filters | List<String> | Filters used (if any) to create the list of resources. Equates to the [filter parameter(s)](#_Filter_Parameter) sent in the request, but included here to allow clients to store complete page information (including sort order & filters used to create the page) for future use. |
| resources | Array<BaseResource> | Ordered list of resources returned from the query. All resources extend the [Base Resource](#_Required_Attributes) model (adding attributes, specific to the resource type) |

## Common Collection Model (post-V1)

NOTE: This needs to be better defined! Also needs to be reconciled with CHRP.

Collections are basically lists of individual resources that are *defined and addressable by a URI*. Fusion Collections are homogeneous, meaning that all elements are of the same “type” (“enclosures”, “server-profiles”, etc.). Collections may be pre-defined or user-defined. For example, the following URI:

…/rest/physical-servers

…represents the pre-defined/implicit collection of “all physical servers managed on the appliance”. Whereas the following URI:

…/rest/physical-servers/collections/my-servers

…represents a user-defined collection of blades.

The model for collections is an extension of the base resource model, described above (attributes with a gray background are inherited from the [Base](#_Required_Attributes) and [Managed](#_Managed_Resource_Attributes) Resource models):

|  |  |  |
| --- | --- | --- |
| Attribute Name | Data Type | Description |
| uri | **String** | The canonical URI of the collection. |
| category | String | The category of the collection, used for authorizations (see [Authentication & Authorization](#_Fusion_Authentication_and), above) and resource type groupings (e.g., in the UI). For example “enclosures”, “server-profiles”, etc. See [Appendix A](#_Appendix_A:_) for a list of current Fusion resource categories & types |
| type | **String** | The type of resources in the collection. For example, “blade” or “rackmount” (as specific types of the “physical-servers” category). This field can be used to determine the exact set of attributes expected/included in the entity/DTO (based on the documented resource models for the given resource type). See [Appendix A](#_Appendix_A:_) for a list of current Fusion resource categories & types |
| created | String | Date and time (in UTC, [ISO 8601](http://en.wikipedia.org/wiki/ISO_8601) format) when the collection was created |
| modified | **String** | Date and time (in UTC, [ISO 8601](http://en.wikipedia.org/wiki/ISO_8601) format) when the collection was last modified |
| name | String | Display name for the collection (UTF-8 for internationalization) |
| description | String | Brief description of the collection (UTF-8 for internationalization) |
| status | String | Aggregate status of the collection (for follow-the-red). Current possible values are OK, Minor, Major, Critical, Disabled and Test. [TODO: get complete definition of Status values from the Health Team when available] |
| state | String | TODO: Not sure if this is needed: Could be used for keeping track of whether a collection is visible, or updated (for smart collections)? |
| ~~properties (optional)~~ | ~~List<~~[~~Property~~](#_Property_Model)~~>~~ | ~~List of properties specific to the particular collection/type~~ |
| ~~actions (optional)~~ | ~~List<~~[~~Action~~](#_Action_Model)~~>~~ | ~~List of actions available on the particular collection/type. Note that this attribute is only returned when the~~ ***~~listActions=true~~*** ~~request parameter is sent on the request~~ |
| ~~organization~~ | ~~String~~ | ~~For RBAC/authorizations, this is the “organization” that owns/is allowed access to this resource~~ |
| total | Integer | The total number of elements that can be safely assumed to be in the *members* list |
| members | List<[Resource](#_Base_Resource_Model)> | The list of resources in this collection |

## ~~Property Model~~

~~The properties list is used to extend the basic resource model with additional properties/attributes that are specific to a particular resource instance/type.~~

~~The Property model is similar to the Property class defined in OVF 1.1, and can be used directly when constructing an OVF Profile or instance (see OVF 1.1, section 9.5, for more details on OVF Property use).~~

|  |  |  |
| --- | --- | --- |
| ~~Attribute Name~~ | ~~Data Type~~ | ~~Description~~ |
| ~~Name~~ | ~~String~~ | ~~Name of the property (or parameter, in the case of Action properties)~~ |
| ~~description~~ | ~~String~~ | ~~Brief description of the property (UTF-8 for internationalization)~~ |
| ~~timestamp~~ | ~~String~~ | ~~Date and time (in UTC,~~ [~~ISO 8601~~](http://en.wikipedia.org/wiki/ISO_8601) ~~format) when the resource was created or last modified~~ |
| ~~Type~~ | ~~String~~ | ~~This is the data type of this property. Available property types are listed in table (x), below~~ |
| ~~Value~~ | ~~String~~ | ~~The value of the property in string form (the actual value is interpreted based on the~~ *~~type~~* ~~field, above)~~ |
| ~~userDefineable~~ | ~~Boolean~~ | ~~Indicates whether this property may be defined by the user or not~~ |
| ~~qualifierList~~ | ~~List<String>~~ | ~~List of qualifiers indicating the acceptable values for properties~~ |

### ~~Property Types~~

~~The following Types can be used for the Types field of Properties. This is a superset of what is in the OVF 1.1 Specification section 9.5, Table 6.~~

|  |  |  |
| --- | --- | --- |
| ~~Type~~ | ~~Description~~ | ~~Value field content~~ |
| ~~Collection<entity>~~ | ~~Collection of type “Entity”~~ | ~~An unordered collection of URIs that are the opaque identifiers of instances of the class (or subclasses of) Entity~~ |
| ~~Enum~~ | ~~Enumeration~~ | ~~The numerical value of a well known ordered list which is associated with a definition for each value~~ |
| ~~DateTime~~ | ~~A string representing the date and time~~ | ~~Date and time in ISO 8601 format.~~ |
| ~~Uint8~~ | ~~Unsigned 8 bit integer~~ |  |
| ~~Sint8~~ | ~~Signed 8 bit integer~~ |  |
| ~~Uint16~~ | ~~Unsigned 16 bit integer~~ |  |
| ~~Sint16~~ | ~~Signed 16 bit integer~~ |  |
| ~~Uint32~~ | ~~Unsigned 32 bit integer~~ |  |
| ~~Sint32~~ | ~~Signed 32 bit integer~~ |  |
| ~~Uint64~~ | ~~Unsigned 64 bit integer~~ |  |
| ~~Sint64~~ | ~~Signed 64 bit integer~~ |  |
| ~~String~~ | ~~String~~ | ~~A contiguous set of UNICODE characters~~ |
| ~~Boolean~~ | ~~Boolean~~ | ~~Value of either 0 or 1~~ |
| ~~Real32~~ | ~~IEEE 4-byte floating point~~ |  |
| ~~Real64~~ | ~~IEEE 8-byte floating point~~ |  |

### ~~Property Qualifiers~~

~~The following qualifiers can be used for entries in a QualifierList. The purpose of the Qualifiers are to inform the interface of acceptable values and ranges for values which the implementation will be used to validate against. This is a superset of the values defined in Table 7 of the OVF 1.1 Specification.~~

|  |  |  |
| --- | --- | --- |
| ~~Qualifier~~ | ~~Description~~ | ~~Example~~ |
| ~~Min~~ | ~~The minimum numeric value acceptable~~ | ~~“Min(0)” would indicate the minimum value is zero.~~ |
| ~~Max~~ | ~~The maximum numeric value acceptable~~ | ~~“Max(16)” would indicate the minimum value is sixteen.~~ |
| ~~MinLen~~ | ~~The minimum string length acceptable~~ | ~~“MinLen(8)” would indicate that the minimum acceptable length string for this value would be 8 UNICODE characters.~~ |
| ~~MaxLen~~ | ~~The maximum string length acceptable~~ | ~~“MaxLen(256)” would indicate that the minimum acceptable length string for this value would be 256 UNICODE characters.~~ |
| ~~Enum~~ | ~~A regular expression as defined in The~~ [~~IEEE~~](http://en.wikipedia.org/wiki/Institute_of_Electrical_and_Electronics_Engineers)[~~POSIX~~](http://en.wikipedia.org/wiki/POSIX) ~~Basic Regular Expressions (BRE) standard of a set of values which represent an enumerated list~~ | ~~“[1-3]” would indicate the values one through three are acceptable.~~ |
| ~~Regex~~ | ~~A regular expression as defined in The~~ [~~IEEE~~](http://en.wikipedia.org/wiki/Institute_of_Electrical_and_Electronics_Engineers)[~~POSIX~~](http://en.wikipedia.org/wiki/POSIX) ~~Basic Regular Expressions (BRE) standard~~ |  |

## 

## Error Message Model

In all cases where an API has failed (i.e., a non-200 status code is returned), the response body will include an Error Message entity. The fields of this entity are described below.

Note that the structure of the Error Message entity is recursive: The top-level Error Message may contain one or more sub-errors (in the ***errors*** array), and each of these sub-errors may contain their own set of sub-errors. In these cases, clients should treat top-level errors as the summary and sub-errors as the specifics (where each sub-error is typically associated with a specific attribute or resource that failed).

|  |  |  |
| --- | --- | --- |
| Attribute Name | Data Type | Description |
| errorCode | String | A string code (not localized – basically an enum) that the client can switch on, for identifying specific errors/error groups. Examples are “VALIDATION”, “AUTHORIZATION”, etc. |
| sourceId | String | A reference to the resource/attribute that the error applies to. This is typically only needed when returning multiple errors: In this case, each ErrorMessage in the errors array, below, should include a sourceId indicating the resource/attribute that the sub-error applies to.  One example is form-validation errors, where this field indicates the attribute that failed validation. Another example is a task that involves multiple resources – in this case, this field would indicate the URI of the resource that failed. |
| errorMessage | String | A **clear and concise** description of the error condition (UTF-8, localized). The description must be meaningful to an end-user (so no internal code references or jargon – just what the user needs to know about the error condition). Messages should include the “what” and “why” of the error condition.  **Note**: Internal/debug info (stack traces, etc.) will be logged in the debug log, not returned in these messages. |
| resolution | String | Describes what the ***user*** could/should do to rectify the problem (localized, UTF-8). For example, “Correct the format of <some attribute> and retry”; “Retry import after resetting the OA”; etc.  **Note**: Avoid over-use of a generic “Contact HP Support” resolution – only use this as a last resort, when there is truly no other appropriate resolution. |
| errors | List<ErrorMessage> | An array of Error Messages (sub-errors), typically used when there are multiple errors, or to provide additional detail on the error condition. In cases where the errors array is present, the ***sourceId*** field of each sub-error will reference the specific resource or attribute to which the sub-error applies.  For example, in a form-validation error, each sub-error will indicate the specific errors on each attribute, and the ***sourceId*** will identify the specific attribute in error. In the case of an operation involving multiple resources, sub-errors will indicate specific errors on individual resources, and the ***sourceId*** field will identify the specific resource in error. |
| rootCause | String | Class name of the exception that resulted in the error  ISSUE: Deprecate this? Should not be exposing internals like this! |
| nestedCause | String | Another error message when there is a rootCause  ISSUE: Can this be removed, along with the above? |

## Activity Resource Model

The following resource is created (via POST to …/rest/activities) and returned by RMs from asynchronous APIs. Clients may then either poll or subscribe to messages on the returned activity URI, in order to monitor the progress and completion status of the asynchronous API.

During asynchronous API execution, the owning RM will update the activity resource (via PUT to …/rest/activities/<activityID>), modifying state, stepsCompleted, progressText, etc.

Fields inherited from the [Base Resource](#_Required_Attributes) model are in gray background, below:

|  |  |  |
| --- | --- | --- |
| Attribute Name | Data Type | Description |
| uri | String | The canonical URI of the Activity Resource (inherited from Base Resource). |
| category | String | The resource category (inherited from Base Resource), used for authorizing access to the Activity Resource.  **Note**: For Activity Resources, this is the category of the ***associatedResource*** (allowing for authorization of activity queries, based on the ***associatedResource*** type) |
| type | String | The specific resource type represented by the entity (inherited from Base Resource). This will always be ***ActivityResource*** for this entity. |
| created | String | Date and time (in UTC, [ISO 8601](http://en.wikipedia.org/wiki/ISO_8601) format) when the Activity Resource was created (inherited from Base Resource) |
| modified | String | Date and time (in UTC, [ISO 8601](http://en.wikipedia.org/wiki/ISO_8601) format) when the Activity Resource was last modified (inherited from Base Resource) |
| name | String | Display name for the asynchronous task/activity |
| userName | String | The name of the user under whose authority the task is running. Required. If null, set from the session token received by the service. |
| state | [TaskState](#_Task_State_Model) | Current state of the asynchronous activity (enum). See [Task State Model](#_Task_State_Model), below, for possible values. |
| associatedResource | String | URI of the resource associated with this activity.  **Note**: For activities that create resources, this field may be null until the point that the resource has been created (must be set by the time the activity has successfully completed) |
| parentTaskUri | String | For sub-tasks, this is the URI of the parent Activity Resource (null if no parent) |
| totalSteps | Integer | Total number of “steps” to be completed for this asynchronous activity. Defaults to 1. |
| stepsComplete | Integer | Total number of steps currently completed. Initialized to ‘0’. Set to ***totalSteps*** when the task is COMPLETE. |
| progressText | Array<String> | Ordered list of strings indicating the progress and eventual completion of the asynchronous activity |
| progressTimestamps | Array<String> | Ordered list of timestamps (in UTC, [ISO 8601](http://en.wikipedia.org/wiki/ISO_8601) format) corresponding to the progress string updates, above |
| taskErrors | List<[TaskError](#_Task_Error_Model)> | Optional list of errors associated with the asynchronous activity. See [Task Error Model](#_Task_Error_Model), below, for the format of these Task Errors. |

### Task State Model

Enumeration of possible task states:

|  |  |
| --- | --- |
| Task State Value | Description |
| CANCELED | Task execution was canceled by the client/user |
| COMPLETE | Task execution completed successfully |
| FAILED | Task execution failed (see taskErrors for details) |
| PAUSED | Task execution has been paused by the client/user |
| PENDING | Task is not yet scheduled for execution |
| ROLLED\_BACK | Task has been rolled-back by the client/user |
| RUNNING | Task is running |
| UNKNOWN | Task is in an unknown/un-defined state |
| WARNING | Task execution completed with warning |

### Task Error Model

If/when errors have occurred, the Activity Resource may contain one or more of the following in the taskErrors field:

|  |  |  |
| --- | --- | --- |
| Attribute Name | Data Type | Description |
| id | Integer | Error code/error ID that the client can use to programmatically identify error types |
| message | String | Error description/overview |
| details | String | Error details string |
| resolution | String | Description of what the user could/should do to rectify the problem |

**Note**: In the future, this should change to use the common [Error Message Model](#_Resource_Schemas/Meta-Data), described above (for consistency & compatibility with other errors).

## Resource Schemas/Meta-Data Model

Resource schemas/meta-data allow users/clients to query for information on how resources and their attributes are meant to be used, and also provides validation rules/constraints that clients can use to pre-validate attributes before attempting to create/modify resources.

See [Resource Schema/Meta-Data URIs](#_Resource_Schema/Meta-Data_URIs), above, for the URI used by clients to query for resource meta-data.

The format used for Fusion meta-data is based on the [JSON Schema](http://json-schema.org/) proposed standard. The minimum JSON Schema attributes supported for Fusion are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Schema  Attribute | Attribute  Type | Required For | Description |
| description |  | All resources & attributes | Description of the resource/attribute (what its purpose/usage is, etc.) |
| type | string | All resource attributes | Describes the type of attribute. Based on JSON types. Clients may use this to determine what is valid to set for the attribute. Possible values are:   * ***string*** * ***number*** (numeric, includes floating point) * ***integer*** (numeric, non-floating point) * ***boolean*** (“true” and “false” values only) * ***object*** * ***array*** * ***any*** (any type values allowed, including null) |
| required | boolean | All resource attributes | If ***true***, the attribute must have a value, and cannot be un-defined (null) |
| default | any | Applicable resource attributes | Defines the default value of an attribute, when the attribute is undefined |
| readonly | boolean | Applicable resources & attributes | If ***true***, indicates that any attempt to modify the resource/attribute (via PUT) will fail. Default is ***false***. |
| minimum | number | Applicable ***number*** and ***integer*** attributes | Minimum value of the numeric attribute |
| maximum | number | Applicable ***number*** and ***integer*** attributes | Maximum value of the numeric attribute |
| minLength | integer | Applicable ***string*** attributes | Minimum length of the string |
| maxLength | integer | Applicable ***string*** attributes | Maximum length of the string |
| items | object | All ***array*** attributes | JSON object defining the expected elements in an array |
| minItems | integer | Applicable ***array*** attributes | Minimum number of values in the array |
| maxItems | integer | Applicable ***array*** attributes | Maximum number of values in the array |
| enum | array | Attributes with a fixed set of possible values | Provides an enumeration of all possible values that are valid for the attribute |
| pattern | string | Applicable ***string*** attributes | Regular expression that the string attribute MUST match to be valid |
| properties | object | All resources and ***object*** attributes | JSON object that defines the individual properties within a given resource or object attribute of a resource |
| format | string | Applicable ***string***  attributes | Defines the format of the string expected in the attribute. The following formats are pre-defined:   * ***date-time***   This SHOULD be a date in ISO 8601 format of YYYY-MM-DDThh:mm:ssZ in UTC time.   * ***date***   This SHOULD be a date in the format of YYYY-MM-DD.   * ***time***   This SHOULD be a time in the format of hh:mm:ss.   * ***regex***   A regular expression, following the regular expression specification from ECMA 262/Perl 5.   * ***color***   This is a CSS color (like "#FF0000" or "red"), based on CSS 2.1 [[W3C.CR-CSS21-20070719](http://tools.ietf.org/html/draft-zyp-json-schema-03#ref-W3C.CR-CSS21-20070719)].   * ***phone***   This SHOULD be a phone number (format MAY follow E.123).   * ***uri***   This value SHOULD be a URI.. |
| searchable | boolean | All resources an attributes | If true, indicates that the resource/attribute is searchable (& should be indexed by the Index service).  Note: This is a Fusion-specific extension to the core JSON Schema definition! |

The JSON Schema format is best described through an example (from the JSON Schema draft proposal):

An example JSON Schema that describes products might look like:

{

"name":"Product",

"properties":{

"id":{

"type":"number",

"description":"Product identifier",

"required":true

},

"name":{

"description":"Name of the product",

"type":"string",

"required":true

},

"price":{

"required":true,

"type": "number",

"minimum":0,

"required":true

},

"tags":{

"type":"array",

"items":{

"type":"string"

}

}

}

}

## Data Format Consistency

### Standardized Units (e.g., for bandwidth)

### Standard Timestamp Format

## Attribute Name Consistency

To ensure overall consistency for Fusion APIs, the attribute names used in the resource model should have a consistent look and feel, and in some cases same attribute name. For Fusion, attribute names should adhere to the following:

* Use camelCase names, first letter is lower case, capitalize the first letter of following each word, and remove the spaces between words. No underscores are used.
* If units are to be returned, that should be included in the attribute name. For example take portSpeed, if returned in GigaBits, it should be portSpeedGbs. Standard notations are expected for units, e.g., Kb, Mb, Gb, etc.
* For common attribute names, such as name, model, serialNumber, all resource managers should be consistent with the names TBD: Need list!

# Fusion REST Operations

## Operations Overview

Basic Create, Read, Update and Delete (CRUD) operations are performed on Fusion resources via the standard HTTP POST, GET, PUT and DELETE methods. RESTful interfaces are based on the world wide web standards, thus most modern web servers can support these operations without modification. The mapping of operations and their behavior is best described in this table:

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | HTTP Verb | Property | Description |
| Create | POST URI  <Payload = Resource data> |  | New resources are created using the POST operation and including relevant data in the payload. On Success the Resource URI is returned. |
| Read | GET URI?query-parameters | Safe,  Idempotent | Returns the requested resource representation(s) |
| Update | PUT URI  <Payload = Update data> | Idempotent | Update an existing resource using the update data. |
| Delete | DELETE URI | Idempotent | Delete the addressed resource |

Restful APIs are, by definition, stateless. The resource state is maintained by the resource manager and is reported as the resource representation. Any application state must be maintained by the client, it may manipulate the resource locally but until a PUT or POST is made, the resource as known by the resource manager is not changed.

**Note**: Support for Create or Update of multiple resources is still under definition and currently not supported.

A given Fusion API will be either synchronous or asynchronous, usually depending on the expected amount of time required to process the request (all APIs are required to return within 200 milliseconds). In general (assuming no errors), synchronous APIs will return with either an HTTP status 200 (OK) or 201 (CREATED) and the associated resource in the response body; and asynchronous APIs will return with 202 (ACCEPTED) and an [Activity Resource](#_Activity_Resource_Model) in the response body, allowing the client to monitor the progress of the asynchronous request.

**Note**: In the future, there may be a way for APIs to be called *either* synchronously or asynchronously, as requested by the client (likely via a request parameter). For now, though, all APIs are one or the other: either synchronous or asynchronous, and the client can determine this based on the HTTP status code returned (202 = asynchronous; all other responses are synchronous)

The following sections describe *all* supported cases for create, read, update and delete of resource (both synchronous and asynchronous cases). The responses described in the following are always for the success case. For *error cases*, the appropriate (non-200-level) [HTTP Status Code](#_HTTP_Status_Codes_1) is returned, and an [Error Message](#_Resource_Schemas/Meta-Data) entity is returned in the response body.

### Create

HTTP POST is *always[[2]](#footnote-2)* used to create new Fusion resources. The POST operation is targeted at a URI representing a *collection* of resources of the type being created. This may either be the URI for all resources of that type, or a custom/user-defined collection of resources of that type. For example:

* POST https://.../rest/physical-servers
  + This would create a new *physical*-*server* resource and add it to the list of all servers managed by this particular resource manager instance

#### Synchronous Create of a URI-Addressable Resource

In most cases, POST is used to create a new, URI-addressable resource, and in these cases a successful POST will return an HTTP status 201 (CREATED) with the URI of the newly created resource in the ***Location*** header of the response, this is known as a synchronous API call.

**Request**:

|  |  |
| --- | --- |
| HTTP Method | POST |
| Request URI | **https://…/rest/<resource-type>**  (e.g., https://…/rest/enclosures) |
| Request Body | ***Client-side representation* of the resource being added**  (e.g., ClaimEnclosure is OK) |

**Response**:

|  |  |
| --- | --- |
| HTTP Status | 201 (Created) |
| Response Body | **A representation of the resource that was created**  (e.g., Enclosure resource) |
| Location Header | **URI to the resource that was created** |

#### Synchronous Create of a Non URI-Addressable Resource

In cases where the resource created by the POST is not URI-addressable (e.g., when posting log messages), the response will be 200 (OK) with no ***Location*** header.

**Request**:

|  |  |
| --- | --- |
| HTTP Method | POST |
| Request URI | **https://…/rest/<resource-type>** |
| Request Body | **Client-side representation of the resource being added** |

**Response**:

|  |  |
| --- | --- |
| HTTP Status | 200 (OK) |
| Response Body | **A representation of the resource that was created** |
| Location Header | **None** |

#### Asynchronous Create

For the asynchronous case, which will typically be used for longer running operations such as an import, The POST request would return a status of 202 (Accepted) and the response body will be an [Activity Resource](#_Activity_Resource_Model). In most all cases, the resource manager should create and persist a partially populated resource prior to returning, the URI for that resource will be returned as the associated resource URI in the Activity tracking resource with the Activity state set to running.

**Request**:

|  |  |
| --- | --- |
| HTTP Method | POST |
| Request URI | **https://…/rest/<resource-type>**  (e.g., https://…/rest/enclosures) |
| Request Body | ***Client-side representation* of the resource being added**  (e.g., ClaimEnclosure is OK) |

**Note**:  The request body may include a **parent task URI**, used by the service to register their sub-tasks in Activity Tracker

**Response**:

|  |  |
| --- | --- |
| HTTP Status | 202 (Accepted) |
| Response Body | **An** [**Activity Resource**](#_Activity_Resource_Model), which the client can use to monitor the progress of the asynch POST/create operation |
| Location Header | N/A |

**Note**:  The associatedResource field of the returned [Activity Resource](#_Activity_Resource_Model) may/may not be filled-in at this point. If not, it will be updated by the service when the new entity has been created, no later than setting the activity status to COMPLETE.

### Read

HTTP GET is always used to retrieve the current representation of a single resource, single resource attribute, or a list of resources identified by the URI. GET is *always* *synchronous*.

On success, the GET request will return an HTTP status 200 (OK) with a representation of the requested resource(s) in the response body. For GET requests that return a list of resources, additional parameters can optionally specify the [start](#_Start_Parameter) index position and the [count](#_Count_Parameter) (number of) resources to be returned, for “pagination” of the results. See the Query String Format section for a more detailed account of these optional parameters.

Note that for all GET/read requests that return multiple resources, a special [Resource List](#_Resource_List_Model) entity is returned, allowing the client to keep track of where they are (essentially which *page* they have received & how many pages are remaining). This entity is returned regardless of whether the client specified a [start](#_Start_Parameter) and/or [count](#_Count_Parameter) parameter was sent in the request, in order to allow services to potentially *limit* the number of resources a request might return for scalability and also to prevent denial-of-service attacks. Because of this, clients should always check whether the ***count*** and ***total*** values of the returned [Resource List](#_Resource_List_Model) match, in order to determine whether additional resources are available from the service.

#### Retrieve a List of Resources

**Request**:

|  |  |
| --- | --- |
| HTTP Method | GET |
| Request URI | **https://.../rest/<resource-type> [?<query-string>]**  e.g., https://.../rest/physical-servers?filter=”status=failed” |
| Request Body | **None** |

**Note:**  The optional ***<query-string>*** is used to filter and paginate the results of a multi-resource query. See the [Query String Format](#_Object_Paths) section for details on supported query string parameters.

**Response**:

|  |  |
| --- | --- |
| HTTP Status | 200 (OK) |
| Response Body | **A** [**Resource List**](#_Resource_List_Model) **entity, including a list of (possibly paginated) array of the requested resources** |
| Location Header | N/A |

**Note**: The control header format for pagination is not yet defined or implemented. It’s also possible that this information will be returned in the form of a standardized Resource List entity, rather than a header.

#### Retrieve a Single Resource

**Request**:

|  |  |
| --- | --- |
| HTTP Method | GET |
| Request URI | **https://.../rest/<resource-type>/<resource-ID>** |
| Request Body | **None** |

**Note:**  In the future, an optional query-string (possibly including a “view” and/or “fields” parameter) may be supported, in order for clients to specify a subset of the resource representation to be returned, however this is not yet implemented.

**Response**:

|  |  |
| --- | --- |
| HTTP Status | 200 (OK) |
| Response Body | **Representation of the requested resource** |
| Location Header | N/A |

#### Retrieve a Single Resource Attribute

**Request**:

|  |  |
| --- | --- |
| HTTP Method | GET |
| Request URI | **https://.../rest/<resource-type>/<resource-ID>/<attribute>** |
| Request Body | **None** |

**Response**:

|  |  |
| --- | --- |
| HTTP Status | 200 (OK) |
| Response Body | **Requested attribute name & current value** |
| Location Header | N/A |

### Update

HTTP PUT can be used to update one or more attributes (or states) for a resource. PUT is used only for updating the resource when the URI for that resource is known. The PUT operation can be made for a partial update, such as a single attribute or possibly a “meta-attribute”. A PUT operation can be either synchronous or asynchronous.

Note: For any read-only attributes sent in an update request: If the value of one or more of these attributes sent in the request is different from the current server state, the server will return an error (403: Forbidden). However, if these attributes match the current server state, it will not be an error. This allows clients to perform resource updates in a GET-modify-PUT fashion, without bumping into read-only errors on attributes that they did not modify.

#### Synchronous Update

In the case of a synchronous PUT operation, the response code will be 200 (OK) and the response body will be a representation of the resource that was updated.

**Request**:

|  |  |
| --- | --- |
| HTTP Method | PUT |
| Request URI | **https://.../rest/<resource-type>/<resource-ID>** |
| Request Body | **Representation of the resource, including one or more attributes on the resource to be updated** |

**Response**:

|  |  |
| --- | --- |
| HTTP Status | 200 (OK) |
| Response Body | ***Full* representation of the resource that was updated** |
| Location Header | N/A |

**Note**: Even for partial resource updates, the full representation of the resource is returned in the response body. This is because modifying one resource attribute may result in changes to other attributes, and by returning the full resource, the client is ensured to have the full, latest resource state.

#### Asynchronous Update

An asynchronous PUT may contain the parent activity resource URI as part of the message body and will return a 202 (Accepted) with the response body set to an [Activity Resource](#_Activity_Resource_Model) with the associated resource URI set to that of the resource the PUT operation was made on.

**Request**:

|  |  |
| --- | --- |
| HTTP Method | PUT |
| Request URI | **https://.../rest/<resource-type>/<resource-ID>** |
| Request Body | **Representation of the resource, including one or more attributes on the resource to be updated** |

**Response**:

|  |  |
| --- | --- |
| HTTP Status | 202 (Accepted) |
| Response Body | **An** [**Activity Resource**](#_Activity_Resource_Model), which the client can use to monitor the progress of the asynch PUT/update operation |
| Location Header | N/A |

#### Update of an Individual Resource Attribute

Some Fusion components will support the update of individual resource attributes, by specifying the attribute to be updated on the URI. In this case, the request body will contain only the name & value of the attribute being updated.

Note that these attributes addressed on the URI may also be “meta-attributes”, and may represent a set of attributes. For example:

* PUT https://.../rest/physical-servers/1234/serversettings
  + This is an example of a meta-attribute, where the body for server settings is a set of attributes, possibly named in a structured fashion.

**Note**: At this point, support for addressing of individual attributes on the URI is purely optional, at the discretion of the resource manager/component. In the future, this may become a requirement for consistency, but it is purely optional at this time.

**Request**:

|  |  |
| --- | --- |
| HTTP Method | PUT |
| Request URI | **https://.../rest/<resource-type>/<resource-ID>/<attribute>**  (e.g., https://.../rest/physical-servers/1234/powerstate) |
| Request Body | **Name & value of the attribute being updated**  (e.g., {powerstate=on}) |

**Note:** It may seem redundant to include the attribute name both on the URI and in the request body, however most data formats (JSON, at least) require a name:value pair as a minimum, and sending just the value in the body may not work (ISSUE: Need to verify this)

**Response**:

|  |  |
| --- | --- |
| HTTP Status | 200 (OK) |
| Response Body | ***Full* representation of the resource that was updated** |
| Location Header | N/A |

**Note**: Even for partial resource updates, the full representation of the resource is returned in the response body. This is because modifying one resource attribute may result in changes to other attributes, and by returning the full resource, the client is ensured to have the full, latest resource state.

#### Updates via POST

While not required, certain Fusion components *may* “update” resources on a POST, if they are able to determine that the resource being “created” in the POST already exists. Again, it depends on the particular RM/resource, whether this behavior is supported or not.

### Delete

HTTP DELETE is always used to delete Fusion resources. When applied to a URI representing an individual resource, the resource is deleted. Once the delete returns, any subsequent requests for that resource would fail as it no longer exists.

#### Synchronous Delete

A synchronous DELETE operation will return no response body and an HTTP status of 204 (No content).

**Request**:

|  |  |
| --- | --- |
| HTTP Method | DELETE |
| Request URI | **https://.../rest/<resource-type>[/<resource-ID>]** |
| Request Body | **None** |

**Response**:

|  |  |
| --- | --- |
| HTTP Status | 204 (No Content) |
| Response Body | **None** |
| Location Header | N/A |

#### Asynchronous Delete

For asynchronous DELETE the request body must be empty (some web servers/libraries do not support a request body), however a parameter may contain a parent Activity URI if needed.

The asynchronous DELETE operation will return HTTP status of 202 with the response body set to an activity resource. If the call returns additional data, for example resource state, then the HTTP status would be 200.

**Request**:

|  |  |
| --- | --- |
| HTTP Method | DELETE |
| Request URI | **https://.../rest/<resource-type>[/<resource-ID>]** |
| Request Body | **None** |

**Response**:

|  |  |
| --- | --- |
| HTTP Status | 202 (Accepted) |
| Response Body | **An** [**Activity Resource**](#_Activity_Resource_Model), which the client can use to monitor the progress of the asynch DELETE operation |
| Location Header | N/A |

### Custom Actions

Custom actions (eg POST /rest/rm/{id}&action=update) are not supported by Fusion, instead a PUT or POST should be made to change the desired attributes or state of the resource. One example may be when a user applies a profile. The profile manager may call the physical-servers resource manager with a PUT and the server profile data, for example:

Instead of POST https://.../rest/physical-servers/1234&action=setProfile

Body = profile data.

Use PUT <https://.../rest/physical-servers/1234/serverpersonality>

The body would ideally be the *serverpersonality* model portion of the server resource, since this is a long running operation, the operation is an asynchronous PUT which will allow for the resource manager to return 202 and an activity resource in an asynchronous fashion.

## Query String Format

The following is a list of commonly used query parameters in the Fusion REST API. Use of these parameters will vary by the specific service or resource manager API, but this list ensures a consistent naming scheme for common parameters.

Note: Passing a query parameter that is unsupported results in a 400 (BAD REQUEST) response. For information on other status codes, see the section of this document.

### Filter Parameter

The ***filter*** parameter is used to specify a general filter/query string. This query string is used to narrow the list of resources returned from a [multi-resource GET/read query](#_Retrieve_a_List). Note that the filter parameter would not be used on URIs representing an individual resource, only on collection URIs (including the base “type” collections). The filter may not contain double quotes, however single quotes may be used when spaces are necessary.

The format of the filter parameter is:

***filter***=’[***not***] <***attribute***> <***operator***> <***value***>’

Where

* ***attribute*** = the resource attribute being filtered (e.g., model, platform, etc.)
* ***operator*** = one of {=, <>, >, >=, <, <=, matches}
* ***value*** = the value of the attribute being filtered. For the “matches” operation, this is a regular expression string.

Note:

* Multiple filter statements represent an “AND” predicate
* “OR” is accomplished via multiple GET requests
* The operands {>, >=, <, <=} only apply to numeric attributes

For example, the following would return the list of Integrity servers with hostnames that begin with “esx”:

https://…/ physical-servers?filter=’platform = Integrity’&filter=’HostName matches ^esx’

Note that the format of the filter string is compatible with standard SQL, allowing these filters to be used directly in SQL queries on the back end.

In other words, the *path* portion of the URI (<https://.../servers>) indicates the "select" part of the query, and the ***filter*** parameter represents the "where" clause. Sorting & counts are specified by separate query parameters (see below).

Note that there are plans in the near-future to change the syntax of the filter parameter, in order to align with the [ESSN-Wide Standard Query Language](http://rndwiki.atlanta.hp.com/confluence/display/BOTO/BTO+Query+Language). This largely amounts to changing the operands from symbols (=, <>, >, >=, <, <=, etc.) to alpha (EQ, NE, GT, GE, LT, LE, etc.), in order to prevent issues with tools blocking these symbols for cross-site scripting concerns, etc.

**Note**: When referencing attributes that are members of objects contained within the primary resource object, dot-notation is used. For example, the following filter would be on the ***networkURI*** attribute of the ***connections*** attribute of the resource:

filter=”**connections.networkURI**=’/crm-core/rest/connectionType/9d266036’”

### Query Parameter

The ***query*** parameter can be thought of as an alternative to the ***filter*** parameter (both are for filtering the results of a [multi-resource GET/read query](#_Retrieve_a_List)), where complex queries are needed (including support for ORs and nested queries – more of a full SQL-like language). The ***filter*** parameter has a much simpler syntax, and should be usable in a large percentage of cases, but for those cases when the filter parameter does not provide the query capabilities needed, the ***query*** parameter may be used.

The format of the ***query*** parameter is based on the [ESSN-Wide Standard Query Language](http://rndwiki.atlanta.hp.com/confluence/display/BOTO/BTO+Query+Language).

### Sort Parameter

The optional ***sort*** parameter is used to specify the sorting order of the returned data set. The format is similar to the “order by” clause in SQL:

***sort***=<***attribute***> ***ascending***|***descending***

Multiple sort parameters result in multiple sort orders (sort by the first ***sort*** parameter first, then by the second…).

For example, the following requests a list of servers sorted by the “name” field in ascending order:

https://.../rest/ physical-servers?sort=name+ascending

### Count Parameter

The ***count*** parameter is used to specify the number of rows to return from a [multi-resource GET/read query](#_Retrieve_a_List).

Specifying *count=-1* indicates that all resources matching the query should be returned. By default, if count is not specified then all resources will be returned (same as *count=-1*).

For example, the following requests a list of the first 100 servers, sorted by “name”:

https://.../rest/ physical-servers?sort=name+ascending&count=100

### Start Parameter

The ***start*** parameter is used to specify the beginning row/record to be returned from a [multi-resource GET/read query](#_Retrieve_a_List), where ‘0’ represents the first available resource (i.e., zero-based index).

If start is not specified, the beginning row will be 0 by default.

For example, the following requests a list of the first 100 servers, beginning with server index #200, sorted by “name”:

https://.../rest/ physical-servers?sort=name+ascending&count=100&start=200

### Fields Parameter (TBD: Not in V1?)

The ***fields*** parameter is used to specify which fields/columns should be returned in the result set.

For example, the following would only return the “hostname” and “model” fields for all ComputeServers:

https://.../rest/ physical-servers?fields=hostname,model

### View Parameter (TBD: Not in V1?)

The ***view*** parameter is used to request a specific “view” of the resource or collection to be returned. Views are pre-defined subsets of attributes to be returned for a resource. Many views are specific to the resource type. Some views that are common across many resource types (and their parameter values) are described in the [Views](#_Views) section, below.

Note that primary way of requesting a particular view is through the Accept header (see [Media Types](#_Media_Types), below). This request parameter is provided only as an alternative for clients (e.g., Flex) that are unable to set the Accept header in their requests.

For example, the following would request a summary view of the ***servers*** collection:

https://.../rest/physical-servers?view=summary

## Document/Data Formats

### General Data Format

In general, the data passed in the bodies of POSTs and returned in the bodies of GETs are some combination of named values, arrays of named values, and simple objects composed of named values. For all of these, a “value” could be a simple value (string, number, etc.) or could be an array or an object.

All of this data could be represented in many formats, including XML, JSON, protobuf, etc. For Fusion (v1) JSON, as specified in RFC4627, is the only format *that will* and must be supported by all Fusion services and resource managers. Other formats (XML, protobuf, etc.) *may* be supported in the future.

### Views (TBD: Not in V1?)

Views are pre-defined subsets of attributes to be returned for a resource. Many views are specific to the resource type. Some views that are common across many/all resource types are described below.

#### Default View

The ***default*** view includes *all* client-visible attributes of the resource. As a minimum, this includes everything listed in the [Base Resource Model](#_Base_Resource_Model), including the [Required Attributes](#_Required_Attributes), any [Optional Attributes](#_Collection_Model) that are *supported by the particular resource*, along with any additional, *resource-specific* attributes defined on that particular resource type.

In the case of collections (including collections that are members of the requested resource), the full [Collection Model](#_Collection_Model) is returned, including the individual elements of the collection.

#### Summary View

For individual resources, the ***summary*** view includes only the ***uri***, ***type***, ***name***, ***description*** (if available), ***timestamp*** and ***status*** (if available) attributes of the resource are to be returned. In other words, everything from the [Base Resource Model](#_Base_Resource_Model), except the ***properties***, ***actions*** and ***organization*** attributes are returned.

In the case of collection URIs (or for any collections that are members of the requested resource), only the collection’s ***uri***, ***type***, ***name***, ***description*** (if available), ***timestamp*** and ***total*** (#elements) are returned, and the individual elements of the collection are *not* returned.

#### Deep View

The ***deep*** view, in many cases a given resource may contain references to other resources in the form of an URI. While this is correct, it can require a client to make multiple requests to build a complete picture for the user. For instance, an activity resource contains an associated URI, however the UI needs to show the name of that resource. In this case, a deep view will expand that URI to the actual resource. \* Issue: Need to document how this looks in the response / model \*. Not all RMs are required to support this feature, this available to help with performance goals when needed.

#### Documentation View

There will be HTML based API documentation made available, this may not be part of the REST API, and may instead be available via a separate help and documentation web server page.

### JSON Specifics

Fusion will use the Jackson JSON processor. The default option for of this serializer will include the @classid in the body. Fusion resource managers and data transfer objects should disable this feature of the serialization.

## HTTP Headers

The following headers are required in Fusion REST requests/responses, in addition to any headers required by HTTP in general (e.g., the Host header).

### Required Request Headers

#### Accept:<media-type>[;version=<api-version>]

…where ***<media-type>*** is the requested data format (e.g., ***application/json***) and ***<version>*** is the Fusion management interface version requested (e.g., “1.0” for the initial release). If the specified media-type or interface version is not supported by the service, the status code of the response will be 406 (Not Acceptable).

**Note**: that JSON is the only format supported by Fusion services and resource managers. Given that Fusion version 1.0 supports only JSON, the Accept request header must be:

Accept: application/json[;version=1.0]

The Accept header is required for all requests that return a response body, Since most Fusion requests return a response body – synchronous DELETE is the lone exception – it is required that all requests include the Accept header. If version is omitted, the current version of the Fusion resource and API will be used.

#### Content-Type:<media-type>[;version=<api-version>]

Where ***<media-type>*** is the requested data format (e.g., ***application/json***) and ***<version>*** is the Fusion management interface version requested (e.g., “1.0” for the initial release). If the specified media-type or interface version is not supported by the service, the status code of the response will be 406 (Not Acceptable).

**Note**: that JSON is the only format supported by Fusion services and resource managers. Given that Fusion version 1.0 supports only JSON, the Accept request header must be:

Content-Type: application/json[;version=1.0]

The Accept header is required for all requests that return a response body, Since most Fusion requests return a response body – synchronous DELETE is the lone exception – it is required that all requests include the Accept header. If version is omitted, the current version of the Fusion resource and API will be used.

#### auth:<session-token>

...where ***<session-token>*** is either a Fusion session or trusted token. The session token is obtained via a login to the Fusion Authentication Service, while a trusted token is used internally for resource manager to resource manager (private) communications over REST.

This header must be sent on all Fusion API requests. If this header is not present on a request, or if the <session-token> is not accepted, the response status code will be 401 (Unauthorized), and the response will include the WWW-Authenticate header, indicating that authentication is required in order to complete the request.

### Optional Request Headers

#### Accept-Charset:<charset>

This header is used to indicate the character sets that are acceptable in the response. If the requested character set is not supported by the service, the response status code will be 406 (Not Acceptable). Given that Fusion version 1.0 supports only UTF-8, the Accept-Charset header must be:

Accept-Charset: UTF-8 ISSUE: Per F2F, think we said UTF-16

See <http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.4> for more information on character set definitions for this header.

#### Accept-Language:<language(s)>

This header is used to indicate the language(s) requested in the response. If none of the requested languages are not supported by the service, the response status code will be 406 (Not Acceptable). If no Accept-Language header is provided, the Fusion appliance locale will be used. Fusion version 1.0 will support at least en-us.

See <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.4> for more information on the Accept-Language header.

#### If-Match:<ETag>

This header is required when updating a resource type that supports Optimistic Locking. The service will attempt to update the resource only if the current ETag for the resource matches the ETag passed in this header. If the ETag specified in this header does not match the resource’s current ETag, status code 412 (Precondition Failed) will be returned from the PUT. If this header is omitted when attempting to update a resource type that supports Optimistic Locking, the PUT will return status code 403 (Forbidden). Update can be forced by specifying the header “***If-Match: \*”.***

#### If-None-Match:<ETag>

This header is used for conditional retrievals (GET). If this header is present, and the service supports conditional retrieval on the specified resource type, the service will only return the requested resource if the current ETag of that resource does not match the ETag sent in this header. If the ETag specified in this header matches the resource’s current ETag, the status code returned from the GET will be 304 (Not Modified). This header may be left out or sent as “***If-None-Match: \****” to force retrieval of the resource. The service will ignore the header if the service does not support conditional retrieval on the specified resource type.

### Required Response Headers

#### ETag:<entity-tag>

The ETag is used to determine if the resource identified by the URI has been modified between REST calls. (Specifically, if two GETs on the same URI would return different Response Bodies, the <entity-tag> must be different.) This allows clients to perform conditional GETs and PUTs using the ***If-None-Match*** and ***If-Match*** headers (see [Optional Request Headers](#_Optional_Request_Headers), above).

NOTE: a GET response for a list of resources should not include the ETag header, instead each resource returned must include the proper ETag .

#### Vary:Accept

Since the actual content returned from a GET may vary, based on the Accept header of the request (by both media type and interface version), setting this header on the response ensures that caches map responses to the specific Accept header in the request, and do not erroneously return the incorrect content (media type, version, etc.) from the cache.

The Vary header is required for all requests that return a response body. Since most Fusion requests return a response body – synchronous DELETE is the lone exception – it is recommended that all requests include this header.

### Optional Response Headers

#### Location:<URI>

This header is returned for POSTs that create a new resource which is identifiable by a URI. The ***<URI>*** field can be used to access the newly created resource.

#### WWW-Authenticate:<challenge>

This header is sent, along with a status code of 401 (UNAUTHORIZED) to indicate that either the ***Authorization*** header was not present in the request of the ***<session-token***> field sent with the ***Authorization*** header was not accepted, and authorization is required to complete the request.

***ISSUE***: What do we put in the <challenge> field here?

### Entity Headers (both request and response entities)

Whenever a request or response includes a message body ([Entity](http://www.w3.org/Protocols/rfc2616/rfc2616-sec7.html)), the following headers apply:

#### Required Entity Headers

* ***Content-Type:<media-type>;<charset>;version=<api-version>***

…where ***<media-type>*** is the data format of the content body (e.g., ***application/json***), ***<charset>*** is one of: (***ISSUE***: What are the supported charsets in Fusion APIs? UTF-8 only?) and ***<api-version>*** is the Fusion management interface version (e.g., “1.0” for the initial release).

When sent in a request, if the specified media-type, charset or interface version is not supported by the service, the status code of the response will be 415 (Unsupported Media Type).

* ***Content-Length:<length>***

…where ***<length>*** is the size of the response body in bytes.

#### Optional Entity Headers

* ***Content-Language:<language>***

This header indicates the language of the response body. See <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.12> for more information on the Content-Language header.

## HTTP Status Codes

Standard HTTP status codes will be returned as applicable. The following table lists the scenarios and the required Response Body for the status codes that may be returned by Fusion REST APIs:

| Status Code | Methods | Scenario | Response Body |
| --- | --- | --- | --- |
| 200  OK | GET | Successful return from a synchronous read/query operation | An entity (or list of entities) corresponding to the requested resource(s) |
| GET | The URI points to a valid resource or collection, but there is nothing to return based on specified filters | Empty list (“[]” in JSON) |
| PUT | Successful return from a synchronous update (or add) operation | An entity (or list of entities) corresponding to the updated resource(s)[[3]](#footnote-3) |
| POST, DELETE[[4]](#footnote-4) | Successful (synchronous) POST or DELETE returned information that is not identified by a URI | An entity that describes the result of the operation |
| 201  Created | POST | Successful return from a synchronous add operation | An entity (or list of entities) corresponding to the created resource(s). |
| 202  Accepted | POST, PUT, DELETE | Successful return from an asynchronous add, update or delete operation | A TaskResource entity (or list of TaskResource entities) is returned, corresponding to the resource(s) affected by the operation. |
| 204  No Content | POST[[5]](#footnote-5), PUT[[6]](#footnote-6), DELETE | Successful return from a synchronous add, update or delete operation | NO response body returned |
| 304  Not Modified | GET | The ETag specified in the optional “If-None-Match” header of the request matches the current resource ETag | NO response body returned |
| 400  Bad Request | ALL | Invalid request URI, invalid header, or invalid request parameter sent in request | ErrorMessage entity describing the specific problem with the request |
| GET, DELETE | The syntax of a request parameter (filter, query, start, count, sort) is invalid or refers to invalid resource attribute names | ErrorMessage entity indicating the specific request parameter(s) in error, and the specific problem(s) with those parameters |
| POST, PUT | Field validation failed: One or more field values sent in an add/update request are not acceptable, because the format or content[[7]](#footnote-7) of the field is invalid | ErrorMessage entity describing the specific fields that are not acceptable and why |
| 401  Unauthorized | ALL | The user (based on session token passed in the Authorization header) is not allowed to perform the operation on the specified resources | ErrorMessage entity describing the specific operations on the specific resources that the user was not allowed to perform |
| 403  Forbidden | POST, PUT | User attempted to update an attribute that is read-only or not allowed to be updated by that user | ErrorMessage entity describing why the request is forbidden |
| POST, PUT | User attempted to update a resource without providing an If-Match Request Header (required for optimistic concurrency control) | ErrorMessage entity, indicating “If-Match” header and ETag is required for updates |
| POST, PUT | For login request, password change required | ErrorMessage entity describing why the request is forbidden |
| 404  Not Found | GET, PUT, DELETE | The URI path (excluding filters) specifies a non-existent resource or collection | ErrorMessage entity indicating which resource(s) were not found |
| 405  Method Not Allowed | ALL | The requested HTTP method is not valid/supported | ErrorMessage entity describing that the method is either not supported[[8]](#footnote-8) or not applicable for the given resource |
| 406  Not Acceptable | ALL | The user has requested an unsupported representation format in the Accept header | ErrorMessage entity (in JSON format) describing the list of representations supported by the server |
| 409  Conflict | POST, PUT, DELETE | The request could not be completed due to a conflict with the current state of the resource(s)[[9]](#footnote-9) | ErrorMessage entity indicating the specific conflict with which resource(s) |
| 410  Gone | GET, PUT, DELETE | (optional) The requested resource has been deleted (only used if the server has some way of knowing this) | ErrorMessage entity indicating which resource(s) were deleted |
| 412  Precondition Failed | PUT | User attempted to update a resource but provided an unacceptable ETag in the “If-Match” Request Header (this typically occurs when multiple users attempt to update the same resource) | ErrorMessage entity indicating which resource(s) failed this test |
| 415  Unsupported Media Type | POST, PUT | The media type of the request body (as specified in the Content-Type header) is not supported by the server | ErrorMessage entity indicating the list of media types supported by the server |
| 500  Internal Server Error | ALL | An unexpected error has occurred that does not fit into a standard error category (examples include database access errors) | ErrorMessage entity providing any available information on the error (at what point the error occurred, accessing what resources, etc.) |
| 501  Not Implemented | ALL | The requested HTTP method is not “currently” supported, but will be in the future (pre-release ONLY) | ErrorMessage entity describing that the requested method is not available, but will be implemented in the future |

# Fusion Messaging

Fusion will use state change messaging between resources to help keep resource managers, the UI, clients, etc. informed when a resource has been changed. State change messages will use a well defined message (below). The index service, for example, will leverage these state change messages to ensure the index service (cache) is as up to date as possible.

Messaging will be available for clients that are capable of receiving messages, there will be a mechanism allowing them to subscribe to and receive messages via REST API calls. TBD: Depends on the final technology choice.

## State Change Messaging

Resource managers will need to send state change messages when a resource they manage changes its state. A state change should be considered as a higher level “lifecycle” change in a resource, such as imported, applying profile, removed, etc. There may be many “sub-states” that are part of applying, in general these should not fire a state change message unless there is a real need, this is done to avoid flooding the message bus.

State changes will be sent on a topic named for their resource type, such as “physical-servers”, “users”, “enclosures” and “activities”. The state change message body is a well defined structure with several optional fields described below.

State change messages should be sent on the “arc” of the state change. In the example below, the state change would be sent as the resource changes from a created state to the importing state. In this case the message includes the change type of updated, specific change type of importing that corresponds to the change in state and the resource URI.

Created

Importing

ChangeType=Updated

SpecificChangeType=Importing

ResourceUri=<uri>

## Client Support

Clients will be able to subscribe to the Fusion message bus, the exact form is TBD. Clients will not be allowed to publish state change messages, publishing messages are reserved for Fusion resource managers and/or services.

## State Change Message Object

The State Change Message object described below will be sent as a JSON serialized string on the message bus. Thus external clients will be able to properly de-serialize the JSON string in the same fashion used in the REST API calls.

|  |  |  |
| --- | --- | --- |
| Attribute Name | Data Type | Description |
| resourceURI | String | The URI for the resource that had the state change |
| changeType | [ChangeType](#_Change_Type_(enum)) | The state change type. These are high level change types, CREATED, UPDATED, DELETED. See description of the [ChangeType](#_Change_Type_(enum)) enum, below. (TBD: Rename from actionType) |
| specificChangeType | String | Mainly used for the UPDATE change type, would typically be the resource state. Definition is specific to the resource type. (TBD: Rename from specificActionType) |
| eTag | String | The ETag for the resource at the instant the state change occurred (when this message was sent) |
| timestamp | String | Time when the message is sent (may be set by library), this is in the [ISO 8601](http://en.wikipedia.org/wiki/ISO_8601) format. (TBD: Can get this from messaging system, need to make available in payload) |
| subChangeType | String | (Optional) if sub state/secondary state messages are needed, this would be the resource specific sub state. Ideally these are rare. |
| resource | String | (Optional except for delete) Would be the resource, use sparingly as this could cause large messages. For small resources, maybe OK, or in cases where a state change always results in a GET request. |
| sourceActivity | String | (Optional) If available, the related activity URI should be included. |
| changedAttributes | List<String> | (Highly Preferred) A list of top level attribute names that have changed based on the POST or PUT call that resulted in the state change message to be sent. A top level attribute refers to the attribute names of the resource and should not include those attributes related to complex objects. TBD: Should this be a Map of attribute name-values instead? |

### Change Type (enum)

Note: Current name is ActionType. To be renamed, along with other changes to message payload (highlighted in yellow, above)

|  |  |
| --- | --- |
| ChangeType Value | Description |
| CREATED | The resource has been newly created/added to the system |
| UPDATED | The resource has been updated (state and/or attributes on the resource have changed) |
| REMOVED | The resource has been permanently removed from the system |

## Subscription Cleanup

TBD: Pending final technology choice

# Appendix A: Fusion Resource Types

The currently known set of resource types in Fusion by resource manager/area are as follows:

Global

* domains
  + Fusion domains
* appliance
  + Contains cluster/IP config
* global-settings
* user-preferences

Authn/Authz

* users
* organizations (future?)
* roles
* login-sessions
* authentication-providers

Logging/Indexing

* debug-logs
* audit-logs
* activities
* index/resources
* index/associations
* index/search-hits
* index/suggestions

Backup/Restore

* backups
* restores

Health

* events
* alerts
* notifications

Firmware Mgmt

* firmware-drivers

Profile Manager (PM)

* server-profiles
* server-profile-templates

Physical Server Resource manager (PSRM)

* enclosures
* enclosure-groups
* enclosure-types
* physical-servers
* physical-server-types

Connectivity Resource Manager (CRM)

* connections
* connection-templates
* networks
  + ethernet-networks
  + fc-networks
* network-templates
* network-sets
* switches
* switch-groups
* switch-types
* port-groups
* vswitches

Environmental Resource Manager ERM

* datacenters
* racks
* power-devices
* unmanaged-devices

# Appendix B: Common Services & APIs

Figure X: CI Foundation Services: The horizontal bars are the truly “foundational” components (tools & technologies that all Fusion components are built on), and the boxes in the middle section are the “common services” provided by the framework (typically exposing their own REST interfaces)

## Appliance Setup Services

### Install

### Setup/Config

### Update

## High Availability Services

### Cluster Configuration

### Application Management

### Backup/Restore

## Security Services

### Authentication Service

### Authorization Service

### Trusted Component

### Session Management

### Device Credential Manager

## Logging Services

### Audit Logs Service

### Product Logs Service

### Activity Tracker

## UI Helpers

### Index Service

#### Search/Lookup

#### Associations

### Global User/Settings Service

TODO

# Appendix C: Resource Managers & APIs

TODO (or not TODO – likely just reference RM API docs here?)

# Appendix D: Examples

TODO

# Appendix E: References

Publications

1. Richardson, Leonard; Ruby, Sam; David Heinemeier Hansson (2008-12-17). RESTful Web Services . O'Reilly Media. Kindle
2. Robinson, Ian; Jim Webber; Savas Parastatidis (2010-09-15). REST in Practice (Kindle Location 2). O’Reilly Media
3. <http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>

A few interesting posts by Fielding

[**REST APIs must be hypertext-driven**](http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven)

[**Paper tigers and hidden dragons**](http://roy.gbiv.com/untangled/2008/paper-tigers-and-hidden-dragons)

Additional references on the web.

* REST on Wikipedia:
  + <http://en.wikipedia.org/wiki/Representational_State_Transfer>
* REST Principles:
  + <http://www.xfront.com/REST-Web-Services.html>
* Implementing REST: Best Practices & Guidelines:
  + [http://www.xml.com/pub/a/2004/08/11/rest.html](http://www.xml.com/pub/a/2004/08/11/rest.html?page=1)
* REST URL Conventions:
  + <http://microformats.org/wiki/rest/urls>
* RESTful and User Friendly URLs:
  + <http://www.artima.com/weblogs/viewpost.jsp?thread=153170>
* RESTful Services – Ajax Patterns.org:
  + <http://ajaxpatterns.org/RESTful_Service>
* REST worst practices:
  + <http://jacobian.org/writing/rest-worst-practices/>
* Versioning REST Web Services:
  + <http://barelyenough.org/blog/2008/05/versioning-rest-web-services/>
* Versioning REST Web Services (Tricks and Tips):
  + <http://barelyenough.org/blog/2008/05/versioning-rest-web-services-tricks-and-tips/>

List of existing RESTful web services which provide a range of solutions:

1. **Blinksale** API, <http://www.blinksale.com/api/>  
   Simple, good example of ROA. Blinksale is a good example of a simple Restful service that conforms to the **Resource-Oriented Architecture** (ROA).
2. **Openstack**, <http://docs.openstack.org/>  
   Divided into a set of services: Compute, Object Storage, Image, Identity and an initial virtual network service (Quantum). Openstack is targeting a cloud environment and so has some similarities with Fusion.
3. **twitter**, <https://dev.twitter.com/docs/api>  
   REST but not ROA.
4. **NetFlix**, <http://developer.netflix.com/docs/REST_API_Conventions>  
   Looks to be a good example of a REST/ROA service.
5. **Sun Cloud API**, <http://kenai.com/projects/suncloudapis/pages/Home>
6. **Google Data Protocol**, <http://code.google.com/apis/gdata/docs/developers-guide.html>  
   Version 2 is fully compliant with the AtomPub protocol (RFC5023).
7. **Amazon services**, <http://aws.amazon.com/documentation/>  
   Provides a range of APIs from those that are SOAP based to REST and various hybrids (support a GET for query and SOAP for operations).  
   S3 service is REST, <http://aws.amazon.com/documentation/route53/>

1. The Virtual Switch Manager (vSM) team is in the process of developing C APIs for Trusted Component functionality. It’s expected that Foundation Services (P&S team) will ultimately own/maintain this code. [↑](#footnote-ref-1)
2. There are very few, internal-only cases where PUT may be used to create resources. An example would be when adding settings to the settings service. In this case, the request looks like a standard PUT request, and the response conforms to the standard responses for POST for synchronous & asynchronous. [↑](#footnote-ref-2)
3. Returned entities minimally include BaseResource fields (URI, status, state, etc.) [↑](#footnote-ref-3)
4. A successful synchronous DELETE typically does not provide a Response Body (i.e. Status 204 is preferred) [↑](#footnote-ref-4)
5. A successful synchronous POST typically provides a Response Body (i.e. Status 201 is preferred) [↑](#footnote-ref-5)
6. A successful synchronous PUT typically provides a Response Body (i.e. Status 200 is preferred) [↑](#footnote-ref-6)
7. Any field in the Request Body that fails validation because it is a URI that references a non-existent resource will cause the POST or PUT to fail with status 400 (Bad Request), rather than status 404 (Not Found) [↑](#footnote-ref-7)
8. Fusion does not support the TRACE and HEAD methods [↑](#footnote-ref-8)
9. Examples include: the resource is marked for deletion; the resource is locked in a transitory state (e.g., applying a profile); the requested state change does not apply in the resource(s) current state (e.g., server is powered-on or OA already managed by other appliance) [↑](#footnote-ref-9)