

Service Interface Design

- [General Information](#)
 - [Status](#)
 - [History](#)
 - [Introduction](#)
 - [Scope](#)
 - [Objective](#)
- [Service Design Principles](#)
 - [Keep It Simple](#)
 - [Be Consistent](#)
 - [Be Resource-Centric: Prefer Nouns over Verbs](#)
 - [Be Clear and Intuitive in naming Resources](#)
 - [Keep the Service URLs Minimalist](#)
 - [URL or Headers? Rule of Thumb](#)
- [Service Interface Requirements](#)
 - [Request](#)
 - [Resource Service URLs](#)
 - [Non-Resource Service URLs](#)
 - [Versioning](#)
 - [Headers](#)
 - [Usage of HTTP Verbs](#)
 - [Resource Associations / Traversal](#)
 - [Query Parameters](#)
 - [Selecting Response Format](#)
 - [Controlling Pagination](#)
 - [Selecting Response Verbosity](#)
 - [Single Resource Search](#)
 - [Response](#)
 - [Headers](#)
 - [HTTP Status Codes](#)
 - [Error Handling](#)
 - [Metadata](#)
 - [Pagination](#)
- [Data Types](#)
 - [Date / Time / DateTime / Duration / Intervals / Repeating Intervals](#)
- [References](#)
 - [Links](#)
 - [Glossary](#)
- [Tasks & Discussion Points](#)

General Information

Status

Introduction

This standard intends to capture the main service interface design principles, characteristics, and technical requirements.

Scope

This standard covers all service interfaced implemented as part of the Pangaea scope, including both @Platform and @Services teams.

Objective

This standard seeks to provide a single, unique perspective and mentality on service interface design. This is significant because, while we are a single organization, we are composed of many individual teams. These multitudes of internal teams are delivering functionality that will be consumed by our API developers. We seek to provide a consistent vocabulary, style, mentality to our developer community such that they have an excellent, intuitive, and positive overall experience building on our platform. We seek to minimize the individual personality of any specific team's perspective such that we avoid the effects of [Conway's Law](#) externally.

Service Design Principles

Throughout the design guidelines, the mentality of these standards is to take a pragmatic viewpoint. We are designing APIs to be consumed by our developer community. As such, each decision we make should bias towards ease of consumption and usage of the service interface. In practice this means that as we apply the design principles, we do so with a bias toward the perspective of the service consumer, not the service implementor. If you're reading this, you are very likely the service implementor and should have your customer in mind while designing and implementing your services.

Keep It Simple

The objective of an API is to enable rapid innovation and decoupling of applications from the underlying systems. To the extent that the APIs expose fundamental operations, innovation is enabled. If presentation specific requirements begin to creep into the APIs, then the APIs become rigid to those application specific requirements and the flexibility of the API erodes. This **should** be avoided as a matter of principle.

Be Consistent

One objective of this API standard is to unify the service interface design where feasible such that the learning curve

for application developers, is reduced as they

Be Resource-Centric: Prefer Nouns over Verbs

REST is based on modeling "resources" which are Nouns.

- Nouns **must** appear as the <resource> in the URL per the Service URL pattern.
- Verbs **must not** appear in the service URL for "resource" based services.
 - The resource service interface **must not** mix nouns and verbs.
 - Use HTTP verbs for identifying the desired interaction with the resource.

We acknowledge that not all services are resource-centric. In cases where the service is not, then the service interface **must** be specified using a verb and not include the noun in the base URL. Because these are non-resource services, there is a single URL for each supported verb.

Be Clear and Intuitive in naming Resources

While we strive to achieve high levels of abstraction in our domain models, this can be taken to extremes. When this happens, our APIs become unintuitive and fail to remain simple, clear and intuitive. Prefer fewer exposed resources with more aggregation of backend complexities, than the converse. This means that resource types will generally become domain facades and the business tier service implementation will need to orchestrate the collaboration among resource tier services.

Keep the Service URLs Minimalist

The objective of the base service urls is to provide an endpoint that is consistent across modalities of interaction. As such, the base urls **should** be consistent over time and through various use-cases with the complexities captured in the query parameters. The usage of intelligent default values will make simple usage of the service interface simple, while not limiting the capabilities of the service to deliver more sophisticated capabilities.

URL or Headers? Rule of Thumb

If the business logic to process the response of the service depends on this information, place the information in the URL so that it becomes easily visible to the developer. If the business logic is invariant to the information, then place the information in the header. Here are some examples:

- Auth/Auth - This information does not change the way the response is handled and therefore belongs in the header.
- Response Format - This information does not change the data returned and is often handled by the marshaling framework of the service client. As such it does not change the business logic of the client.
- Response Verbosity - This information innately interacts with the business logic of the client and represents the client specific needs within the domain model. It must be modeled as a URL query parameter, not a header.

Service Interface Requirements

Request

Resource Service URLs

Resource service URLs **must** follow this patterns.

Resource Collection URL

http[s]://<hostname>:<port>/<domain>/<version number>/<resource>?<query parameters>

Resource Instance URL

http[s]://<hostname>:<port>/<domain>/<version number>/<resource>/<primary key>?<query parameters>

where:

| URI Fragment | Requirements | Examples |
|------------------|--|---|
| <hostname> | | api.walmart.com api.qa.walmartlabs.com |
| <port> | Must be 80 for HTTP Must be 443 for HTTPS | 80 |
| <domain> | Must be all lower case. Must come from an approved namespace for the domain. | paas transaction |
| <version number> | Must be "v<x>" where <x> is the MAJOR version number only. <x> must be a non-negative integer. <x> should only ever be zero in pre-production environments or in alpha or beta releases with ARC approval. | v1 v2 |
| <resource> | This portion of the URL accesses the "collection" of resources for this service. Must be the "noun" name of the resource. Must be the plural form of the resource name. Must be in all lower case. | jobs carts customers accounts |
| <primary key> | This portion of the URL accesses a single instance of the "resource" within the collection for this service. Must follow the UUID standard. | |

| | | |
|--------------------|--|--|
| <query parameters> | <p>Must follow the standardized query parameters, where specified.</p> <p>Must follow the guidelines for choosing between placing content in the request payload, headers or query parameters.</p> | |
|--------------------|--|--|

Non-Resource Service URLs

Non-Resource service URLs **must** follow this patterns.

Non-Resource URL

http[s]://<hostname>:<port>/<domain>/<version number>/<verb>?<query parameters>

Examples of verbs / non-resource might include:

- Translate
- Convert
- Calculate

Versioning

Service interfaces **must** be versioned at the major version number only. The implication is that all minor and patch level releases **must** be **backward compatible** within the same major version. The mentality here is that you are representing the version of the interface, not of the deployed implementation of that interface. The implementation version will vary per our general **versioning and release standards**.

Major service interface versions **must** be supported for some period of time to enable clients to adopt and migrate to the new version. When the current version of a service interface is being superseded by the next version (i.e. transitioning from v1 to v2, or from v5 to v6), the deprecated current version **must** be supported for no less than 6 months from the date of availability of the updated service interface in production environments. The service implementor **may** choose to support the deprecated service interface version for a longer period of time if business conditions merit the investment. If the major versions evolve quickly, the service implementor **must** support all prior versions of the service interface that are within the support time window. This means that there may be situations with multiple (more than 2) major versions are presently deployed and supported in our infrastructure. This situation should be discouraged in favor of practices that provide greater levels of stability in our service interfaces.

Headers


... support for Service Invocation Chaining


... support for Auth/Auth Tokens

... support for Response Content Type "Content-Type: application/[xml | json]"

Usage of HTTP Verbs

HTTP verbs are the mechanism of defining the interaction between the request and the resource. The primary operations on a resource are CRUD - **Create, Read, Update, Delete**. These four verbs map to the HTTP verbs Put, Get, Post and Delete. The interaction of verbs on both the resource collection and instance URLs are as follows:

| CRUD | HTTP Verb | Resource Collection URL e.g. /foos? | Resource Instance URL e.g. /foos/<primary key>? | Semantics |
|--------|---------------------|--|--|--|
| Create | PUT | Create a new instance of the resource and return the id. | If exists then error, else create new instance | <p>The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload. A successful PUT of a given representation would suggest that a subsequent GET on that same target resource will result in an equivalent representation being returned in a 200 (OK) response.</p> <div> David's Requirements</div> |

| | | | | |
|--------|----------------------|---|--|--|
| Read | GET | Access the full collection of all instances | If exists then return instance, else error | Idempotent |
| Update | POST | Bulk Update | If exists then update the instance, else error | <p>The POST method requests that the origin server accept the representation enclosed in the request as data to be processed by the target resource.</p> <p>The actual function performed by the POST method is determined by the server and is usually dependent on the effective request URI.</p> <div></div> |

| | | | | |
|--------|--------|--------------------------------------|---|--|
| | | | | - - d o c u m e n t s u p p l e m e n t a l v e r b s - |
| Delete | DELETE | Delete all records in the collection | Delete the specified instance in the collection | |

Resource Associations / Traversal

...
.../foos/999/bars?

Query Parameters

Selecting Response Format

If your resource provides for multiple response formats, this must be specified as a "suffix" notation.

| Resource Scope | Example |
|----------------|---|
| Collection | .../foos.json?<query parameters> .../foos.xml?<query parameters> |

| | |
|----------|---|
| Instance | .../foos/999.json?<query parameters> .../foos/999.xml?<query parameters> |
| Verb | .../translate.xml?<query parameters> |

If present, the suffix value **must** override the "Accept:" header.

Valid values for the suffix are:

json - JSON formatted response

xml - XML formatted response

png - PNG formatted response

jpg - JPEG formatted response

JSON **must** be the default response format.

Controlling Pagination

If your resource requires pagination controls when accessing the collection, the following query parameter format **must** be used.

| <query parameter> | Requirement | Examples |
|-------------------|--|--------------------|
| offset | Must be a non-negative integer value. This implies that a zero value is valid and is the first position in the result set. | .../foos?offset=25 |
| limit | <p>Must be a positive integer value. This implies that a negative or zero limit is invalid.</p> <p>A default value of 10 should be assumed if no other considerations apply. The default value should be modified by the domain business logic or response size where appropriate. If the responses are small, then the limit may be raised. If the responses are large, a lower limit may be appropriate.</p> <p>If a service wishes to return <i>all</i> the available resources in a collection, the service must do this by providing a large default value for the limit. This is not recommended.</p> | .../foos?limit=25 |

Selecting Response Verbosity

If your resource allows for controlling the verbosity, specifically the fields in the domain model to be returned, then the following query parameter format **must** be used.

| <query parameter> | Requirement | Examples |
|-------------------|---|----------------------------------|
| fields | | .../foos?fields=id,name,summary |
| field_groups | <p>If a service defines a related collection of fields as a labeled group, that label may be used to filter the response fields for the resources returned by the request.</p> <p>If available, the service must define these labeled groupings in the API documentation.</p> | .../foos?field_groups=base,media |

Single Resource Search

If your resource allows for searching then the following query parameter format **must** be used. This feature, when present **should** be used in conjunction with the pagination query parameters.

| <query parameter> | Requirement | Examples |
|-------------------|---|---|
| q | The free text search terms are to be entered following the 'q' parameter. | .../foos?q=free+text+to+search |
| <field> | <p>For each field that allows filtering, the name of the field should be used to express the constraint. Depending on the data type, the expression of the constraint will vary. Here is a summary:</p> <p>Range query on Numeric Fields:</p> <p><field>=999 <field>=[x TO y] <field>=[* TO 999] <field>=[999 TO *] <field>=EXISTS</p> <p>Range queries on Date, Time, or Date Time fields follows the format above, but replaces the 'x', 'y' and '999' with the format specified in the Data Types section.</p> | .../foos?createdAt=2012-12-25T06:53:85.351Z |

| | | |
|-----|--|--|
| ... | What else to consider here? <ul style="list-style-type: none"> <input type="checkbox"/> Brian Johnson - filtering <input checked="" type="checkbox"/> Brian Johnson - 1-dimensional faceting <input type="checkbox"/> Brian Johnson - n-dimensional faceting <input type="checkbox"/> Brian Johnson - sort field(s) & direction(s) | |
|-----|--|--|

Response

Headers

...

HTTP Status Codes

... are defined in [RFC 2616](#).

| Use-Case | Status Code | Which HTTP Verbs | Whose Error | Description |
|----------|-----------------------|------------------|---------------------|--|
| 200 | OK | <ALL> | Nobody - It worked! | |
| 201 | Created | PUT | Nobody - It worked! | |
| 304 | Not Modified | POST | Nobody - It worked! | |
| 400 | Bad Request | <ALL> | Client | |
| 401 | Unauthorized | <ALL> | Client | |
| 403 | Forbidden | <ALL> | Client | |
| 404 | Not Found | GET, POST | Client | |
| 500 | Internal Server Error | <ALL> | Server | |
| 501 | Not Implemented | <ALL> | Server | When the request method is not supported. e.g. GET, PUT, POST, DELETE, OPTIONS, TRACE, ... |
| 503 | Service Unavailable | <ALL> | Server | |
| | | | | |

Error Handling

When the service implementation returns a 4xx or 5xx HTTP status code, the following data model **must** be returned in the response payload.

Error Message Payload

```
_error : {  
  "customerMessage" : "...",  
  "developerMessage" : "...",  
  "errorCode" : <functional_area>-<error_code>,  
  "documentationURL" : "http://<developer portal  
server>/api/documentation/errors/<errorCode>/"  
}
```

The service developer **should** provide a verbose plain english developerMessage which provides hints about how to fix the problem if it is a client error. If it is a server error and a response payload can be generated successfully, it **should** include a detailed explanation of the issue encountered.

Metadata

If the service response provides aggregate or summary information as a portion of the response, then the response data model must place those data elements in a `_metadata` field in the root scope of the response.

Metadata Data Element

```
_metadata : { /* metadata fields go here */ }
```

Pagination

If the response supports pagination, then the response data model **must** provide the `totalCount` in the response metadata such that the client can calculate possible valid pagination input parameters. The service implementation **should not** provide more detailed pagination metadata.

Pagination Metadata

```
_metadata : { totalCount : 98765 }
```

Data Types

Date / Time / DateTime / Duration / Intervals / Repeating Intervals

Elements of the data model representing dates, times, or durations **must** follow [RFC 3339](#) and [ISO 8601](#) with the following clarifications with respect out usage in our systems.

- The timezone **must** be present if the precision of the value includes time element. This clarification is required because ISO 8601 allows for the timezone to be optionally included and if absent local time is assumed.
- Truncated representations must not be used. This clarification is required because ISO 8601:2000 allows for

truncation, though this provision was removed in the subsequent ISO8601:2004 version.

References

i Remove this info macro once you've completed writing the documentation for this section.

Link to any documentation or attachments, adding an explanation of their purpose in this context, where necessary.

Links

REST - http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

RESTafarian - <http://mikeschinkel.com/blog/whatisarestafarian/>, Example: <http://tech.groups.yahoo.com/group/rest-discuss/message/6845>

Glossary

REST - **RE**presentational **S**tate **T**ransfer - An architectural style for distributed hypermedia systems.

Tasks & Discussion Points

- ☐ _____ - Do we want to allow for <domain> within the Service URL? It is documented for now, but my recommendation is to remove this from the specification and disallow it. Each resource should be unique within the Walmart API ecosystem. If it is not, then we either have duplicate functionality, or we have a confusing API.
- ☐ _____ - Tim had requested that the service interface be described through the use of the OPTIONS HTTP verb. Requirements need to be collected and described here.
- ☐ _____ - Any clarifications between PUT/POST and create/update semantics over what's in the spec now? Reference: <http://jcalcote.wordpress.com/2008/10/16/put-or-post-the-rest-of-the-story/> - specifically Create use PUT for full content, POST for creating a subordinate resource using server side algo; Update use PUT to REPLACE existing full content, POST to update provided fields / subordinate resources
- ☐ _____ - What HTTP Response Codes do we want to specify / recommend?
- ☐ _____ - Error response payload data model... Do we want error codes? What's the design / support model around them? How do we codify error codes to give domains control to create and administer them in isolation? Seems like a foundation capability tied to automation and publishing. I have a design in mind.
- ☐ _____ - The terms in RED require further definition and supporting documentation.
- ☐ _____ - How far do we want to go with response formatting options? (JPG, PNG, CVS, ...)