Credits

Author: Steffen M. Fohn, ADP (steffen.fohn@adp.com)
Other Contributors:
Isabel Espina, ADP
Michael Figura, OAGi
Boonserm Kulvatunyou, NIST
Scott Nieman, Land O'Lakes
Michael Rowell, Oracle
Nikola Stojanovic, Individual Invited Expert
Jim Wilson, OAGi
## Document History

<table>
<thead>
<tr>
<th>Version Number</th>
<th>Release Date</th>
<th>Author</th>
<th>Version / Revision Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>V 2.0 RC 2.1</td>
<td>December 2017</td>
<td>Steffen Fohn</td>
<td>Version contributed</td>
</tr>
<tr>
<td>V 2.0 RC 2.2</td>
<td>May 2019</td>
<td>Steffen Fohn</td>
<td>Updated per OAGi REST Working Group Review</td>
</tr>
</tbody>
</table>
## Table of Contents

1 Conventions 8  
2 Introduction 9  
  2.1 Purpose 11  
  2.2 Scope and Applicability 11  
  2.3 Goal of This Specification 11  
  2.4 Definitions and Terminology 11  
    2.4.1 Definitions 11  
    2.4.2 Terminology 15  
3 Rationale 16  
4 Versioning 16  
  4.1 Backwards Compatibility 17  
5 Message Architecture 19  
6 Message Headers 21  
  6.1 General Headers 22  
  6.2 Request Headers 24  
  6.3 Response Headers 38  
  6.4 Entity Headers 42  
  6.5 Custom Headers 48  
  6.6 Caching 55  
    6.6.1 Expiration Mechanism 55  
    6.6.2 Validation Mechanism 56  
7 Message Resource Identificaiton 58  
  7.1 Resource Types 58  
  7.2 URI Design and Format 59  
    7.2.1 Service Owner 59  
      7.2.1.1 Design 59  
    7.2.2 API and Developer Domains 60  
      7.2.2.1 API Domain 60  
      7.2.2.2 Developer Domain 61  
    7.2.3 URI Path 61  
      7.2.3.1 Service Domain 61  
      7.2.3.2 API Version 62  
      7.2.3.3 Resource Model 62  
      7.2.3.4 Format 63  
    7.2.4 URI Query 64  
      7.2.4.1 Design 65
7.2.4.2 Format 65
7.3 URI Encoding 65
7.4 URI Template Design and Format 66
8 Message Resource Management 67
  8.1 Query Criteria in the Query Component 68
    8.1.1 Specifying Filter Criterion 70
    8.1.2 “any/all” Lambda Operators 71
  8.2 Query Criteria in the Path Component 72
  8.3 CRUD Operations 72
    8.3.1 Create Operations 76
    8.3.2 Update Operations 77
    8.3.3 Delete Operations 79
    8.3.4 Read Operations 79
      8.3.4.1 Specifying Selection Criterion (for a Partial Response) 81
      8.3.4.2 Specifying Expansion Criterion 81
      8.3.4.3 Specifying Instance Resource Start Sequence Criterion 83
      8.3.4.4 Specifying Instance Resource Maximum Number Criterion 84
      8.3.4.5 Specifying Instance Resource Total Number Criterion 85
      8.3.4.6 Specifying Order Criterion 86
      8.3.4.7 Specifying Search Criterion 87
      8.3.4.8 Specifying Pagination Criteria 88
        8.3.4.8.1 Client Requires Pagination Read Consistency 93
        8.3.4.8.2 Client Does Not Require Pagination Read Consistency 96
      8.3.4.9 Specifying View Criterion 97
    8.3.5 Conditional Operations 98
    8.3.6 A Note on Nulls 98
  8.4 Custom Operations 99
  8.5 Bulk Operations 101
  8.6 A Pattern for Large URIs and Query Components with Sensitive Data 101
9 Hypermedia Controls 106
  9.1 Hypermedia Actions 112
10 Confirmation Management 113
  10.1 HTTP Response Status 114
    10.1.1 1xx Informational 119
    10.1.2 2xx Success 119
    10.1.3 3xx Redirection 121
    10.1.4 4xx Client Error 123
    10.1.5 5xx Server Error 127
  10.2 Confirm Message Request 128
List of Figures

Figure 1: API Specification Versioning and Dependency ........................................ 16
Figure 2: HTTP Message Architecture .................................................................. 20
Figure 3: Positive Approach to Cache Validation ................................................. 57
Figure 4: Resource Management Operation ....................................................... 68
Figure 5: Get Request Message ........................................................................... 89
Figure 6: Get Response Message ....................................................................... 90
Figure 7: Systems Interaction for URIs with Voluminous and Sensitive Data
- Save and Query an Instance Resource Set (Pattern 1) ................................... 102
Figure 8: Systems Interaction for URIs with Voluminous and Sensitive Data
- Query the Instance Resource Set (Pattern 2) .............................................. 104
Figure 9: Link Description Model ....................................................................... 106
Figure 10: Hypermedia Actions Logical Model .................................................. 112
Figure 11: Confirm Message Logical Model ...................................................... 130
Figure 12: Asynchronous Service Provider Push Response Pattern .................. 140
Figure 13: Asynchronous Service Consumer Pull Response Pattern ............... 143
Figure 14: Asynchronous Service Consumer Polling and Pull Response
Pattern ............................................................................................................. 146
Figure 15: Event Notifications Long Polling Pattern ........................................... 150

List of Tables

Table 1: General Headers .................................................................................... 22
Table 2: Request Headers ................................................................................... 27
Table 3: Response Headers .................................................................................. 39
Table 4: Entity Headers ....................................................................................... 43
Table 5: Custom Headers ..................................................................................... 50
Table 6: HTTP Request Methods for Resource Data Management ................. 73
Table 7: HTTP Request Method Usage for CRUD Operations ......................... 75
Table 8: HTTP Request Method for Custom Operations .................................. 100
Table 9: HTTP Response Status Codes ............................................................. 116
Table 10: HTTP Response Status Code Usage ............................................... 118
Table 11: Types of Messages by Request Processing Status ........................... 135
Table 12: Confirm Message Status to HTTP Response Status Map .............. 137
1 Conventions

- **Italicized** Text
  - Used to emphasize text.

- **Bolded** Text
  - Used to indicate words and characters that identify concepts significant to this specification (e.g. defined terms, HTTP header field names)

- **Italicized, Bolded** Text
  - Used to indicate variables that should be replaced by value assignments.

- [option]
  - Content contained in the brackets is optional (e.g. literals, variable value assignments) as optional.

- alternative1 | alternative2
  - The pipe symbol separates alternatives.

- “literal”
  - Quotation marks surround literal text and quoted reference material.
2 Introduction

An Application Programming Interface (API) is a named set of operations (functions and data) that is offered by a provider system and used by consumer systems to enable communication between the provider and consumer systems (i.e. applications). A Web Service (i.e. a Service exposed on the World Wide Web) offers a Web API. A Web API that conforms to the REST architectural style is a RESTful Web API.

The REST (Representational State Transfer or Representational Entity State Transfer) architectural style was named as such and defined by Roy Fielding in 2000 in his Ph.D. dissertation that describes the Web’s architectural style. [Fielding (2000)]

Fielding’s description of the REST architectural style consisted of constraints in six categories:

- Client-Server
- Stateless
- Cache
- Uniform Interface
- Layered System
- Code-On-Demand

The Uniform Interface is a central feature of the REST architectural style that distinguishes it from other network-based styles. The Web’s components (e.g. clients, servers, reverse proxy) depend on the interface uniformity for their interaction and communication. Fielding identifies four constraints of the uniform interface interface [Fielding (2000), Masse (2011)]:

- Identification of Resources:
  where each concept (known as a resource) may be addressed by a unique identifier such as a URI.
- Manipulation of Resources through Representations:
  where the representation is a means to interact with the resource but is not the resource, itself.
- Self-Descriptive Message:
  where a resource’s desired state can be represented within a client’s request message; a resource’s current state may be represented within a server’s response message; metadata may be included to convey addition information on the resource (e.g. resource state, representation format).
- Hypermedia as the Engine of Application State (HATEOAS)
  where a resource’s state representation includes links to related resources.

RESTful Web APIs are designed according to the REST architectural style and leverage the existing Hypertext Transfer Protocol (HTTP) as the application communication protocol. HTTP specifies that resources be retrieved via a unique identifier or Uniform Resource Identifier (URI) that corresponds to their server-side representation [HTTP/1.1 (1999)]. The representation of the resource may be supported by one or more formats (e.g., HTML, XML, CSV, JSON, ATOM and JPEG). HTTP serves as the basis for the uniform interface constraint of the REST architectural style (mentioned above).

Leonard Richardson developed a RESTful Web API Maturity Model consisting of four levels; each level declares an aspect of the Web’s Uniform Interface that a RESTful Web API must be satisfied for a given maturity level. [Richardson (2008), Fowler (2010)]

- Level 0
• Must use the HTTP protocol for communication transport
  
  At this level, HTTP is essentially used as a tunneling mechanism. For example, Web Services using SOAP send messages with the HTTP POST method to the same URL; the operation to be invoked is communicated in the body of the SOAP message.

• Level 1
  • Meets Level 0 requirements
  • Must use Resources
    
    This level establishes resources and their management through the communication of their state representation.

• Level 2
  • Meets Level 1 requirements
  • Must use HTTP Verbs and HTTP Response Codes
    
    This level requires that all Create, Read, Update and Delete (CRUD) data management operations performed on a resource must use the established HTTP methods (e.g. POST, GET) for those operations. All message confirmations (i.e. success or failure) must use the established HTTP response status codes.

• Level 3
  • Meets Level 2 requirements
  • Must use Hypermedia controls
    
    This level is referred to as Hypertext As The Engine of Application State (HATEOAS). A Resource’s current state representation may include hypermedia controls (i.e. links) that provide the requesting system (i.e. Service Consumer) a set of possible next steps (i.e. operations) in the context of systems interacting to realize a use case.

Recall that Fielding [Fielding (2000)] defined Level 3 as pre-requisite to being RESTful. Therefore, this document is written to support a Level 3 maturity level.
2.1 Purpose

The RESTful Web API Design Standard is targeted for both the API designer and the application developer as a specification supporting the design and implementation, respectively, of component or system interfaces exposed on the web according to the REST architectural style.

It describes the techniques, patterns and formats with associated rules that are necessary for the consistent design and use of OAGi's RESTful Web APIs. The majority of the document is intended to be independent of any specific resource representation format; any information that is specific to a given format (e.g. JSON) is documented in a section dedicated to that format.

The document offers guidance to the API designer in developing and maintaining RESTful Web API Specifications, including the following primary activities:

- Determining the resource model for the service being designed and enhanced.
- Designing a URI scheme for identifying the resources that comprise the service.
- Identifying the HTTP methods and any additional parameters that will be used to manage the resources.
- Designing and implementing the resource representations that are supported by each method.

In addition, application developers may use the specification as a general reference guide, as a design point for RESTful Web API implementations, or as a supplement to a given RESTful Web API's specification.

2.2 Scope and Applicability

This specification explains how to design interoperable OAGi RESTful Web APIs.

This specification applies to all systems (internally developed or purchased from third parties) that expose OAGi RESTful Web APIs.

2.3 Goal of This Specification

This specification describes the techniques, patterns, formats and associated rules that are necessary for the consistent design and use of OAGi’s RESTful Web APIs. As such, it forms a common basis upon which OAGi’s RESTful Web API Specifications should be developed and maintained. A given RESTful Web API Specification, designed to meet certain integration requirements, may not address the full breadth or scope of design considerations addressed in this specification.

2.4 Definitions and Terminology

2.4.1 Definitions

**Application Programming Interface (API)**
An Application Programming Interface (API) is a named set of operations (functions and data) that is offered by a service provider (i.e., Server) and used by service consumer (i.e., Client) to enable communication between the components.

**Backwards Compatibility**
Backwards Compatibility means that a new version of an API does not break clients programmed to a previous version of the API. This means that a client, programmed to a
previous version of an API, can continue to communicate with a server, programmed to a
the new version of an API, without any negative impact.

**Bug Fix**  
A bug fix is an internal change that fixes incorrect behavior. [Preston-Werner]

**Business Component**  
Business Component is a component that is an implementation of an autonomous business
concept or process. Responsibility within the business component may be allocated to one
or more logical tiers (i.e., User Interface or User, User Dialog or Workspace, Business Logic
or Enterprise, Business Type or Resource). A business component *may* have one tier of
each kind and each tier may be implemented by one or many distributed components.
[Herzum (2000)]

**Client**  
Client represents either a component or system that acts as a consumer of services.

**Client-Server**  
Client-Server is a hierarchical architectural style for network-based applications. A server
offers a set of services and listens for requests upon those services. A client, desiring a
service to be performed, sends a request to the service. The server either rejects or
performs the request and sends a response back to the client. [Fielding (2000)]

**Component**  
Component is a self-contained piece of software that can be independently deployed and
plugged into an environment with a compatible socket; it has a well-defined and network
addressable runtime interface and can collaborate with other components. The term is used
to represent both Distributed Components and Business Components. [Herzum (2000)]

**Controller**  
A controller resource models a procedural concept. It encapsulates application specific
actions that cannot be logically mapped to one of the standard methods (create, retrieve,
update, and delete, also known as CRUD). [Masse (2011)]

**Collection Resource**  
A collection resource is a server-managed directory of resources. Clients may propose new
resources to be added to a collection. However, it is up to the collection to choose to create
a new resource, or not. A collection resource chooses what it wants to contain and also
decides the URIs of each contained resource. [Masse (2011)]

**Distributed Component**  
Distributed Component is a component that in terms of granularity is the smallest
component that offers interfaces. Tiers of the business component are often implemented as
distributed components. [Herzum (2000)]

**Entity**  
The information transferred as the payload of a request or response. An entity consists of
metainformation in the form of entity-header fields and content in the form of an entity-
body. [Fielding et al (1999)]

**Entity Body**  
An entity-body is only present in a message when a message-body is present. If present,
the entity-body sent with an HTTP request or response is in a format and encoding defined
by the entity-header fields. [Fielding et al (1999)]

**Hypertext Transfer Protocol (HTTP)**
Hypertext Transfer Protocol is an application layer protocol for distributed, collaborative, hypermedia information systems. HTTP has been used by the World Wide Web since 1990. [Fielding et al. (1999)]

**Instance Resource**
An instance resource is a singular concept that is akin to a document instance or database record. An instance resource may have child resources that represent its specific subordinate concepts. [Masse (2011)]

**Instance Resource Set (or Set of Instance Resources)**
An instance resource set is a set of instance resources that is determined by a server (at a point in time) to satisfy the set’s membership criteria (i.e., selection, filter, expansion and search criteria) of a resource management operation (e.g. GET request) upon a collection resource.

**Interaction**
An Interaction identifies the messages exchanged between systems or components in the context of a collaboration; it includes the sequencing of these message send/receive events.

**JavaScript Object Notation (JSON)**
JSON is a lightweight computer data interchange format derived from JavaScript Programming Language. It is a text-based, human-readable format for representing collections of name/value pairs (i.e. objects) and ordered list of values (i.e. arrays). [JSON.org] [Zyp et al. (2013a)]

**Message**
Message is a definition (or specification) of conveyance of information from a sender to a receiver.

**aMethod Message**
aMethod Message is a request message defined with an HTTP method (e.g., OPTIONS, GET, HEAD, POST, PUT, and DELETE). For example, a GET Message is a message defined with the GET method.

**Message Body**
The message-body (if any) of an HTTP message is used to carry the entity-body associated with the request or response. The message-body differs from the entity-body only when a transfer-coding has been applied to the entity-body, as indicated by the Transfer-Encoding header. [Fielding et al. (1999)]

**Message instance**
Message Instance is an instance of a message that complies with the Message.

**Object Class**
Object Class is a set of ideas, abstractions, or things in the real world that are identified with explicit boundaries and meaning and whose properties and behavior follow the same rules. [ISO 11179 (2003)]

**Origin Server**
The server on which a given resource resides or is to be created. [Fielding et al. (1999)]

**Representational State Transfer (REST)**
Representational State Transfer (REST) is a description of an architectural style that was developed by Roy Fielding in 2000 through derivation of the Web’s architectural style.

**Request Message (or Client Request Message)**
A message sent from a client to a server.

**aMethod Request**
A request is a message instance (i.e., Request) that is communicated with an HTTP method (e.g., OPTIONS, GET, HEAD, POST, PUT, and DELETE); it is used to request information from a server.

**Resource**

Resource is a concept that can be referenced by a unique identifier and manipulated by the uniform interface. [Masse (2011)] “Any information that can be named can be a resource: a document or image, a temporal service (e.g. “today’s weather in Los Angeles”), a collection of other resources, a non-virtual object (e.g. a person), and so on.” [Fielding (2000)] A REST API is composed of 4 distinct resource archetypes: document, collection, store, and controller. [Masse (2011)]

**Resource State Representation**

Resource State Representation is the rendered state of a resource in a representation (e.g. JSON, XML). A resource’s desired state can be represented in a client’s request message. A resource’s current state can be represented in a server’s response message. [Masse (2011)]

**Response Message (or Server Response Message)**

A message sent from a server to client, that defines the result of a request message.

**aMethod Response Message**

*aMethod* Response Message is a response message that defines the result from a *aMethod* Message (defined with an HTTP method (e.g., OPTIONS, GET, HEAD, POST, PUT, and DELETE). For example, GET Response Message is a message that defines the response defined with the GET method.

**aMethod Response**

A request’s response is a message instance that is communicated in response to *aMethod* Request.

**RESTful Web API**

RESTful Web API is a Web API that conforms to the Web’s REST architectural style.

**Request (or Client Request)**

Request is a message instance sent from a client component to a server component. It is defined with one of the client request methods: OPTIONS, GET, HEAD, POST, PUT and DELETE.

**Response (or Server Response)**

Response is a message instance sent from a server component to a client component as a result of a client component’s request to a server component.

**Resource Model**

The resource model describes an API’s key concepts that are exposed in the API’s URIs’ path.

**Server**

Server is a component or system that acts as a provider of services.

**Service**

Service is a software program that makes its functionality available via a published technical interface. [Erl et al. (2012)]

**Status Monitor**

A Status Monitor is a type of Controller. Its job is to encapsulate the processing required to determine if an asynchronous unit of work has completed.

**System**
A System (also known as a Business Component System) is a composition of business components assembled together to deliver a solution to a business problem (i.e. application). [Herzum (2000)]

**Uniform Resource Identifier (URI)**
A Uniform Resource Identifier (URI) is a compact sequence of characters that uniquely identifies an abstract or physical resource. [Berners-Lee (2005)]

**URI Template**
A URI Template is a compact sequence of characters for describing a range of Uniform Resource Identifiers through variable expansion. [Gregorio (2012)]

**User Agent**
User Agent is the client that initiates a request. These are often browsers, editors, spiders (web-traversing robots), or other end user tools. [Fielding et al. (1999)]

**Web API**
Web API is an Application Programming Interface (API) to a Web Service (i.e. a Service exposed on the World Wide Web).

### 2.4.2 Terminology

This document uses the following terminology:

1. **MUST**: This word means that the requirement is absolutely REQUIRED to be implemented with no exceptions.
2. **MUST NOT**: This phrase means that the requirement specifies an absolute PROHIBITION and is not to be implemented.
3. **SHOULD**: This word means that the requirement is REQUIRED unless an exception has been granted through the exception process.
4. **SHOULD NOT**: This phrase means that the requirement is REQUIRED NOT to be implemented unless an exception has been granted through the exception process.
5. **MAY**: This word means that the requirement is OPTIONAL.

*Note: Terminology adapted from Scott O. Bradner, “Key words for use in RFC’s to Indicate Requirement Levels,” The Internet Engineering Task Force (IETF) RFC (Requests for Comments) 2119, March 1997.*
3 Rationale

This specification establishes a set of techniques, patterns and formats with associated rules that must be reused and common to OAGi’s RESTful Web APIs. This offers consistency in the design and use across OAGi’s RESTful Web APIs; benefits include:

- Agility in API design and implementation for the Service Provider
- Ease of API adoption and use by Service Consumers (e.g. Clients)

4 Versioning

A Web API realizes a Web APISpecification. A given specification has a distinct version. A specification may comprise several constituent parts that are separately managed artifacts. As with the specification, these managed artifacts are also versioned. Referring to Figure 1, an API operation may have its request message body defined in a separately managed schema (e.g. a noun schema); further, the request message body schema may be composed of other separately managed schemas (e.g. component or field schemas). Each of these separately managed schemas is versioned. Changes to the separately managed schemas affects not only their version identifier(s), but also the version identifier(s) of the specifications that reference them.

![Figure 1: API Specification Versioning and Dependency](image-url)
Each managed artifact must be versioned with a major version identifier, minor version identifier and patch version identifier. An increment in the patch version identifier indicates a bug fix that resolves incorrect behavior. An increment of the minor version identifier indicates a new version of the artifact that maintains backwards compatibility. An increment of the major version identifier indicates a new version of the artifact that breaks backwards compatibility. [Preston-Werner]

For any version identifier (major, minor or patch) or revision identifier, the following rules apply:

R1 A version identifier MUST be a non-negative integer.
R2 A version identifier MUST begin with the number 0.
R3 A version identifier MUST be incremented by 1.
R4 When a major version identifier is incremented, the minor and patch version identifiers MUST be reset to 0.

For any artifact (Web API or constituent part), the following rules apply:

R5 Version and revision information MUST be expressed in the form

\[ majorVersionID.minorVersionID.patchVersionID \]

R6 Use of 0 for a major version identifier MUST be limited to initial development.
R7 Version 1.0.0 MUST be used the initial public version.
R8 The major version identifier MUST be incremented when a new public version is created/issued that breaks backwards compatibility.
R9 The minor version identifier MUST be incremented when a new public version is created/issued that maintains backwards compatibility.
R10 The patch version identifier MUST be incremented when a bug fix is introduced.

Representation of the Web API version information in the exposed operational interface is described, below, in the section URI Design and Format.

4.1 Backwards Compatibility

A new version of a Web API is backwards compatible if it does not break clients using a previous version of the API. This means that a client, using a previous version of an API, can continue to use the new version of that API, offered on a server, without any negative impact. With a backwards compatible API change, a client, implementing an API’s previous version, will experience the same behavior from a server, implementing the API’s new version. Only those clients, implementing the API’s new version, will experience the new behavior from a server, implementing the API’s new version.
R12 Clients MUST be designed to ignore data elements that are not recognized.¹

The following examples illustrate some cases when backwards compatibility is maintained and when it is broken.

An API maintains backwards compatibility in cases when:

- An optional property is added.
- A value is added to a data element’s value domain (enumeration).²
- The metadata of a property is expanded (e.g. reduced minimum size, increased maximum size)
- An optional operation is added
- An optional query parameter is added

An API breaks backwards compatibility in cases when:

- A property is changed from optional to mandatory
- A property is removed.
- A property that is mandatory is added.
- A value is removed or changed from a data element’s value domain (enumeration).
- The metadata of a property is restricted or changed (e.g. different data type, increased minimum size, reduced maximum size).

¹ According to the robustness principle, also known as Postel’s law, which states “Be conservative in what you send, be liberal in what you accept.”

² While technically the addition of a value to a data element’s domain doesn’t break the interface’s backwards compatibility, the addition might require the client, itself, to change, if for example, the addition is associated with a mandatory compliance requirement.
5 Message Architecture

Messages of a RESTful Web API adhere to the HTTP message architecture. All HTTP messages (client request messages and server response messages) may comprise three components:

- start-line
- message-header
- message-body

The start-line represents the client request-line in the case of a request and the status-line in the case of a response. The request start-line includes the HTTP method, a URI that identifies the resource, and the HTTP version number. The response status-line contains the HTTP version number, a number indicating the status of the request and a short phrase describing the status.

The message headers are used to transfer a variety of data between message senders (e.g. clients and servers) and receivers (e.g. clients, intermediary caches and servers). Some headers transfer control data between message senders and receivers. When set by a client, such headers communicate client data used by a server to control its response to the client (e.g. preferred format); when set by a server, such headers communicate server data used by a client to control requests to the server (e.g. time duration to wait before retrying a failed request). Control data also includes directives for intermediary caches. Other headers transfer metadata (e.g. expiration date and time) on the resource representation in the message-body.

The message-body carries the entity-body of the request or response message. A common use of the entity-body is in the response message to convey the state of a request message’s identified resource. An entity-body differs from the message-body only if a transfer-coding has been applied to the message-body. [Fielding et al (1999)] Figure 2 shows the architecture of the HTTP message.

---

3 Transfer-codings are applied by the application to ensure safe and proper transfer of the message. Note: The header applies to the message, not the entity.
The HTTP/1.1 specification defines rules for when a message-body is allowed in a message; it differs for request and response messages. [Fielding et al (1999)]

For requests, the presence of a message-body is indicated by the inclusion of a **Content-Length** or **Transfer-Encoding** header in the message-headers.

R13  A message-body **MUST NOT** be included in a request if the specification of the request method does not allow sending an entity-body in requests.

R14  A server **SHOULD** read and forward a message-body on any request.

R15  If the request method does not include defined semantics for an entity-body, then the message-body **SHOULD** be ignored when handling the request.

For responses, the presence of message-body is dependent upon both the request method and the response status code. Details specific to a given request method are described below in the Confirmation Management section.

The components of the HTTP message and the details of their use, as part of RESTful Web API messages, are described in the sections that follow.
6 Message Headers

This section describes the message headers and their related usage rules. Related usage rules that are dependent to a resource management or confirmation management context are documented in their respective sections. For example, the confirmation management section describes the response header(s) that must be returned in a server response, conditional on the results status of processing the client request.

There are four types of HTTP message-headers [Spainhour (1996)] [Fielding et al. (1999)]:

- general-header – a header that has general applicability for both request and response messages and do not apply to the entity being transferred;
- request-header – a header that allows the client to communicate information about the request and about the client itself to the server;
- response-header –a header that allows the server to pass information about the response (that cannot be communicated in the status-line), about the server, and about further access to the resource (identified in the request URI) to the client;
- entity-header – a header that provides information on the entity-body or, if no entity-body is present, about the resource identified in the request.

All message-headers follow the same format: a field name followed by a colon, “:” and the field value. The field-value should be preceded by a single space (SP). Message-headers may extend over multiple lines by preceding each extra line with at least one space (SP) or horizontal-tab (HT).

Although the order in which headers of different field names are received is not significant, it is good practice to send general-headers first, followed by request-headers or response-headers, and ending with entity-headers. [Fielding et al. (1999)]

HTTP allows for multiple occurrences of a message-header with the same field name. However, it must be possible to combine the multiple headers into one field-name: field-value pair without changing the semantics of the message; each subsequent field-value is appended to the preceding field-value, separated by a comma. The order of the headers with the same field name is significant to the interpretation of the combined field-value. Therefore, the order of these headers must not be changed.

Subsections, below, describes the HTTP message-headers and their expected use.

Note: Use of the HTTP message-headers in this specification is intended to consistent with the HTTP message-header as defined in W3C’s HTTP 1.1 specification. Any modifications, in regard to the use of the HTTP response status codes in this specification, are limited to changes in requirement levels (e.g. change of requirement from a SHOULD to a MUST) or the addition of details specific to their use in a RESTful Web API. The purpose of these modifications is to constrain the space of response code usage to that required for partner interaction in a trading community.

There are about 50 HTTP message-headers. A subset of these headers is used in this specification.

Note: Security standards on authentication and authorization are separately managed and documented. Furthermore, the use of security-related HTTP headers may vary by the security solution (e.g. OAuth). While this document provides an overall view the HTTP headers used, readers must use the appropriate security standard for specific guidance of the use these HTTP headers.

R16 Any HTTP header not mentioned in this section MUST NOT be used.
6.1 General Headers

A general header has general applicability for both request and response messages and does not apply to the entity being transferred.

<table>
<thead>
<tr>
<th>Header Field Name</th>
<th>Obligation</th>
<th>Format &amp; Description</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache-Control</td>
<td>Conditional</td>
<td>“Cache-Control” “: ” directives</td>
<td>Conditions</td>
</tr>
<tr>
<td></td>
<td>- in request</td>
<td>Specifies directives that must be obeyed by all caching mechanisms along the</td>
<td>• Applicable security standard requirements.</td>
</tr>
<tr>
<td></td>
<td>- in response</td>
<td>request/response chain.</td>
<td>• Application (API) requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Caching requirements</td>
</tr>
<tr>
<td>Date</td>
<td>Optional</td>
<td>“Date” “: ” datetime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- in request</td>
<td>Indicates the date and time when the message originated.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- in response</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pragma</td>
<td>Conditional</td>
<td>“Pragma” “: ” directives</td>
<td>Conditions</td>
</tr>
<tr>
<td></td>
<td>- in request</td>
<td>Specifies implementation-specific directives that might apply to any recipient</td>
<td>• Applicable security standard requirements.</td>
</tr>
<tr>
<td></td>
<td>- in response</td>
<td>along the request/response chain.</td>
<td>• Application (API) requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Caching requirements</td>
</tr>
</tbody>
</table>

Table 1: General Headers

**R17** The Cache-Control header MUST be used to specify cache directives (i.e., field values) that are to be obeyed by all caching mechanisms along the request/response chain.

“Cache-Control” “: ” directives

Cache directives are classified into Cache request directives (i.e., directives that may be included on requests) and Cache response directives (i.e., directives that may be included on responses).

---

4 Obligation values are: Mandatory, Conditional, and Optional. Mandatory headers must exist and must conform to the provisions of this specification. Conditional headers must be treated as Mandatory if the associated condition is satisfied. Optional headers are not required, but if they exist they must conform to the provisions of this specification.
R17.1 The **directives** field value for *request directives* **MUST** be limited to an element of the value domain:

- “no-cache”
- “no-store”

R17.1.1 The **no-cache** field value **MUST** be used to instruct the caching mechanism that the response, elicited by the request, must not be used to satisfy a subsequent request without successful revalidation with the origin server.

R17.1.2 The **no-store** field value **MUST** be used to inform a caching mechanism to not store any part of either the request or any response to it.

R17.2 The **directives** field value for *response directives* **MUST** be limited to an element of the value domain:

- “no-cache” “=” field-name,
- “max-age” “=” seconds,
- “must-revalidate”,
- “no-store”.

R17.2.1 If the **no-cache** field value does not specify a **field-name**, then it **MUST** be used to instruct the caching mechanism that the response must not be used to satisfy a subsequent request without successful revalidation with the origin server.

R17.2.2 If the **no-cache** field value does specify one or more **field-name**(s), then it **MUST** be used to instruct the caching mechanism that the response may be used to satisfy a subsequent request; however, the specified **field-name**(s) must not be sent in the response to a subsequent request without revalidation with the original server.

*Note:* This allows an origin server to prevent re-use of certain header fields in a response (e.g. a Cookie header) while allowing the remainder of the response to be cached.

R17.2.3 The **max-age= seconds** field value **MUST** be used to inform a caching mechanism that the response is considered stale after the specified number of seconds from the time of making the request for the resource.

R17.2.4 The **must-revalidate** field value **MUST** be used to inform a caching mechanism that it must revalidate a cache entry on any subsequent use if the cached response is stale.

R17.2.5 The **no-store** field value **MUST** be used to inform a caching mechanism to not store any part of the response of the request that elicited it.

R17.3 A request or response **MAY** include the **Cache-Control** header.
The Date header MUST be used to indicate the date and time when the message originated.

“Date” :: ”datetime

The datetime field value MUST adhere to the format and value domains as specified in IETF’s RFC 1123 (an update to RFC 822).

A request MAY and a response MUST include the Date header.

Note: The Date header along with the Expires header in a response allow clients to determine the freshness of a resource representation. [Masse (2011)]

The Pragma header MUST be used to specify implementation-specific directives (i.e., field values) to any recipient along the request/response chain.

“Pragma” :: ”directives

Note: HTTP 1.1. prefers use of the Cache-Control header.

The directives field value response directives MUST be limited to an element of the value domain:

no-cache

The no-cache field value MUST be used to instruct the proxy to not cache the resource representation.

Note: The Pragma: no-cache directive has the same semantics as the Cache-Control: no-cache directive. The Pragma directive is allowed for HTTP 1.0 backwards compatibility. HTTP 1.1 prefers use of the Cache-Control: no-cache header.

A request or response MAY include the Pragma header.

Examples of the general headers are shown below:

Cache-Control: no-cache
Date: Sun, 06 Nov 1994 08:49:37 GMT
Pragma: no-cache

6.2 Request Headers

A request header allows the client to communicate information about the request and about the client itself to the server.
<table>
<thead>
<tr>
<th>Header Field Name</th>
<th>Obligation</th>
<th>Format &amp; Description</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accept</strong></td>
<td>Mandatory</td>
<td>“Accept” “: ” ( “<em>/</em>”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- in request</td>
<td>(type)”/” “*”)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(type)”/” “*”)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>) ” “q” “=” “qvalue”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>) ” “masked” “=” “true”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>”false”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Describes media type(s) and subtype(s) that are acceptable for the response. The optional qvalue represents an acceptable quality level for acceptable types.</td>
<td></td>
</tr>
<tr>
<td><strong>Accept-Charset</strong></td>
<td>Mandatory</td>
<td>“Accept-Charset” “: ” (</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- in request</td>
<td>character-set</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>” “*” )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>) ” “q” “=” “qvalue”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifies the character set(s) that are acceptable for the response. The optional qvalue represents a quality level for acceptable languages.</td>
<td></td>
</tr>
<tr>
<td><strong>Accept-Encoding</strong></td>
<td>Conditional</td>
<td>“Accept-Encoding” “: ” (</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- in request</td>
<td>encoding-scheme</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>” “*” )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>) ” “q” “=” “qvalue”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifies the encoding scheme(s) used for the entity-body that are acceptable for the response. The optional qvalue represents a quality level for acceptable content codings.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Application (API) requirements, the entity-body may require encoding to ensure safe and proper transfer.</td>
<td></td>
</tr>
<tr>
<td><strong>Accept-Language</strong></td>
<td>Conditional</td>
<td>“Accept-Language” “: ” (</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- in request</td>
<td>language</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>” “*” )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>) ” “q” “=” “qvalue”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifies the language(s) that are acceptable for the response. The optional qvalue represents a quality level for acceptable languages.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Application (API) requirements.</td>
<td></td>
</tr>
<tr>
<td><strong>Authorization</strong></td>
<td>Conditional</td>
<td>“Authorization” “: ” scheme</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- in request</td>
<td>credentials</td>
<td></td>
</tr>
<tr>
<td>Header Field Name</td>
<td>Obligation(^4)</td>
<td>Format &amp; Description</td>
<td>Condition</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------</td>
<td>----------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Cookie</td>
<td>Conditional</td>
<td>“Cookie” :: &quot;name&quot; = &quot;value&quot;  &lt;br&gt; Contains name/value pairs (i.e., cookies) for that URI previously sent by the server with the Set-Cookie header.</td>
<td>Conditions  &lt;br&gt; • Applicable security standard requirements.  &lt;br&gt; • Application (API) requirements, used to identify a user’s state.</td>
</tr>
<tr>
<td>Host</td>
<td>Mandatory</td>
<td>“Host” :: &quot;hostname [&quot;&quot;:&quot;port]&quot;  &lt;br&gt; Specifies the host and port number of the URI.</td>
<td></td>
</tr>
<tr>
<td>If-Match</td>
<td>Conditional</td>
<td>“If-Match” :: &quot;&quot;</td>
<td>entity-tag&quot;  &lt;br&gt; Used with a method to make it conditional; the method is performed only if the client entity (via the given entity tag, ETag header) matches the server entity.</td>
</tr>
<tr>
<td>If-Modified-Since</td>
<td>Conditional</td>
<td>“If-Modified-Since” :: &quot;&quot;</td>
<td>datetime&quot;  &lt;br&gt; Used with a method to make it conditional; the method is performed only if the resource representation has been modified since the date given in this header.</td>
</tr>
<tr>
<td>If-None-Match</td>
<td>Conditional</td>
<td>“If-None-Match” :: &quot;&quot;</td>
<td>entity-tag&quot;  &lt;br&gt; Used with a method to make it conditional; the method is performed only if the client entity (via the given entity tag, ETag header) does not match the server entity.</td>
</tr>
<tr>
<td>If-Range</td>
<td>Conditional</td>
<td>“If-Range” :: &quot;&quot;</td>
<td>entity-tag&quot;</td>
</tr>
<tr>
<td>Header Field Name</td>
<td>Obligation&lt;sup&gt;4&lt;/sup&gt;</td>
<td>Format &amp; Description</td>
<td>Condition</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------</td>
<td>----------------------</td>
<td>-----------</td>
</tr>
</tbody>
</table>
| If-Unmodified-Since | Conditional - in request | “If-Unmodified-Since” “: " **datetime**  
Used with a method to make it conditional; the method is performed only if the resource representation has *not* been modified since the date given in this header. | Conditions  
- Application (API) requirements  
- Caching requirements |
| Prefer | Conditional - in request | “Prefer” “: " **preference**  
**preference** = **token** [“ = ” **value**] [” ; " **parameter** "=" **value**]  
Used to indicate that particular server behaviors are preferred by the client. | Conditions  
- Application (API) requirements |
| Range | Conditional - in request | “Range” “: ” **byte-range** | Conditions  
- Efficient recovery from partially failed transfers.  
- Efficient partial retrieval of large entities. |
| Referer | Conditional - in request | “Referer” “: " **uri**  
Provides the URI of the resource from which the requested URI was obtained. | Conditions  
- Applicable security standard requirements.  
- Application (API) requirements  
- Logging requirements  
- Caching requirements |
| User-Agent | Conditional - in request | “User-Agent” “: " **user-agent**  
**user-agent** = "**product** [“=" **product-version** ] | Conditions  
- Application (API) requirements, used for statistical purposes, tracing protocol violations and automated recognition of user-agents. |

Table 2: Request Headers
The Accept header MUST be used to describe the media type, subtype and masking criteria that are acceptable for the response.

```
"Accept" "(""/*"*/ | (type="*" | "*")) |
(type="*"/"*" subtype)) ["";"" ""q""=""qvalue] ["";"" ""masked"" =""true"" | ""false""]
```

Note: "/*" matches all media types; type="*" matches all subtypes of that type.

The type/subtype value MUST adhere to the value domain governed by IANA as the set of registered media types.

Note: See the IANA Registry of MIME Media Types [IANA].

Using the q parameter, each type/subtype value MAY be associated with a quality value (qvalue).

The quality value MUST range between "0" and "1" and adhere to the format and value domain as specified in IETF’s RFC 2616. If no quality value has been specified, then the quality value defaults to "1".

Note: The quality value represents the user’s preference. A quality value of "1" indicates the user’s preferred media type.

Note: See RFC 2616 [Fielding et al. (1999)].

A request MUST include the Accept header.

Using the masked parameter, each type/subtype value MAY be associated with a masked value.

The masked value MUST be limited to an element of the value domain:

"true",
"false".

The "true" value MUST be used by the client to indicate that masked sensitive data is acceptable for the response.

The "false" value MUST be used by the client to indicate that unmasked sensitive data is acceptable for the response.

If the masked parameter is missing from the request, then the default interpretation MUST be "masked = true".
The **Accept-Charset** header **MUST** be used to specify the character set(s) that are acceptable for the response.

```
"Accept-Charset" "": " ( character-set | "*" )
["; " "q" "=" "qvalue"

*Note:* "*" matches all character sets.
```

**R21.1** The **character-set** value **MUST** adhere to the value domain governed by IANA as the set of registered character sets.

*Note:* See the IANA Registry of Character Sets [IANA(2013d)].

**R21.2** Using the **q** parameter, each character set value **MAY** be associated with a quality value (**qvalue**).

**R21.2.1** The quality value **MUST** range between "0" and "1" and adhere to the format and value domain as specified in IETF’s RFC 2616. If no quality value has been specified, then the quality value defaults to "1".

*Note:* The quality value represents the user’s preference. A quality value of "1" indicates the user’s preferred character set.

*Note:* See RFC 2616 [Fielding et al. (1999)].

**R21.3** If the character set of the **Accept-Charset** header cannot be generated then the server **SHOULD** return a **406 Not Acceptable** status code.

**R21.5** A request **MUST** include the **Accept-Charset** header.

The **Accept-Encoding** header **MUST** be used to specify the encoding scheme(s) used for the entity-body that the client can accept.

```
"Accept-Encoding" "": " ( encoding-scheme | "*" )
["; " "q" "=" "qvalue"

*Note:* "*" matches all encoding schemes.
```

**R22.1** The **encoding-scheme** value **MUST** be limited to an element of the value domain governed by IANA as the set of registered HTTP content-coding values.

*Note:* See the IANA Registry of HTTP Content-Coding Values [IANA(2013c)].

**R22.2** Using the **q** parameter, each encoding scheme value **MAY** be associated with a quality value (**qvalue**).
R22.2.1 The quality value MUST range between “0” and “1” and adhere to the format and value domain as specified in IETF's RFC 2616. If no quality value has been specified, then the quality value defaults to “1”.

Note: The quality valued represents the user’s preference. A quality value of “1” indicates the user’s preferred encoding scheme.

Note: See RFC 2616 [Fielding et al. (1999)].

R22.3 If the content code of the Accept-Encoding header cannot be generated then the server SHOULD return a 406 Not Acceptable status code.

R22.5 A request MAY include the Accept-Encoding header.

R22.6 If no Accept-Encoding header is present in a request, the server MAY assume that the client will accept any content coding.

R23 The Accept-Language header MUST be used to specify the language(s) that are acceptable for the response.

```
Accept-Language” “;” ( language | “*” )
[“;” “q”=”qvalue”]
```

Note: “*” matches all languages.

R23.1 The language value MUST adhere to the format and value domains as specified in IETF’s RFC 5646.


R23.2 Using the q parameter, each language value MAY be associated with a quality value (qvalue).

R23.2.1 The quality value MUST range between “0” and “1” and adhere to the format and value domain as specified in IETF’s RFC 2616. If no quality value has been specified, then the quality value defaults to “1”.

Note: The quality valued represents the user’s preference. A quality value of “1” indicates the user’s preferred language.

Note: See RFC 2616 [Fielding et al. (1999)].

R23.3 If the content code of the Accept-Language header cannot be generated then the server SHOULD return a 406 Not Acceptable status code.

R23.5 A request MAY include the Accept-Language header.
The Authorization header MUST be used to provide the client’s authorization to access the resource representation at a URI.

“Authorization” “: ” scheme credentials

Note: For a resource requiring authorization, the server will return to a user agent, that attempted to access that resource without the proper credentials, a 401 Unauthorized response including a WWW-Authenticate header that describes the type of authorization required.

The scheme credentials field value MUST adhere to the format and value domains of the applicable security standard.

A user agent that wishes to authenticate itself with a server, after receiving a 410 Unauthorized response MUST include the Authorization header.

The Cookie header MUST be used to contain name/value pairs for that URI previously sent by the server with the Set-Cookie header.

“Cookie” “: ” name “=” value

The name = value field value SHOULD adhere to the format and value domains as specified in IETF’s RFC 6265.

Note: See IETF RFC 6265 [Barth (2011)].

A request MAY contain the Cookie header.

The Host header MUST be used to specify the host name and optionally port number of the resource being requested, as obtained from the URI.

“Host” “: ” hostname [“:” port]

The hostname and port field value SHOULD adhere to the format and value domains as specified in IETF’s RFC 2616 syntax for host representation.

Note: See RFC 2616 [Fielding et al. (1999)].

A request MUST include the Host header.

The If-Match header MUST be used with a method to make it conditional; the method is performed only if the client entity (via the given entity-tag, ETag header) matches the server entity.

“If-Match” “: ” “*” | entity-tag
### R26.1
The **If-Match** field value **MUST** be limited to an element of the value domain:
```
"*",
entity-tag,
```

- **R26.1.1** The "*" field value **MUST** be used to match any entity at the server.
- **R26.1.2** The **entity-tag** field value **SHOULD** adhere to the format and value domains as specified in to IETF’s RFC 2616 for entity tag representation.

*Note:* See RFC 2616 [Fielding et al. (1999)].

### R26.2
If any of the entity tags supplied in the request **matches** the entity tag (that would have been returned in the response to a **GET** request on that resource), then the method **MAY** be performed by the server.

### R26.3
If "*" is supplied in the request and any current entity exists for that resource, then the method **SHOULD** be performed by the server.

### R26.4
If none of entity tags supplied in the request **match** the entity tag of the entity (that would have been returned in the response to a **GET** request) or if "*" is given and no current entity exists, then the server **MUST NOT** perform the requested method and **MUST** return a **412 Precondition Failed** status code in the response.

### R27
The **If-Modified-Since** header **MUST** be used with a method to make it conditional; the method is performed **only if** the resource representation has been modified since the date given in this header.

```
"If-Modified-Since": "datetime"
```

- **R27.1** The **datetime** field value **MUST** adhere to the format and value domains as specified in to IETF’s RFC 1123 (an update to RFC 822).

- **R27.2** If the resource representation **has been modified**, then the method **MAY** be performed by the server.

- **R27.3** If the resource representation **has not been modified** since the date provided in the **If-Modified-Since** header, then the server **MUST** return a **304 Not Modified** status code in the response.

- **R27.4** Clients when sending the **If-Modified-Since** header **SHOULD** use the exact date string received in a previous **Last-Modified** header.

- **R27.5** A request **MAY** include the **If-Modified-Since** header.
**R28** The If-None-Match header **MUST** be used with a method to make it conditional; the method is performed **only if** the client entity (via the given entity-tag, ETag header) does **not match** the server entity.

```
“If-None-Match” “-” “*” | entity-tag
```

**R28.1** The If-None-Match field value **MUST** be limited to an element of the value domain: “*”, entity-tag.

**R28.1.1** The “*” field value **MUST** be used to match any entity at the server.

**R28.1.2** The entity-tag field value **SHOULD** adhere to the format and value domains as specified in IETF’s RFC 2616 syntax for entity tag representation.

*Note:* See RFC 2616 [Fielding et al (1999)].

**R28.2** If any of the entity tags, supplied in the request, **do not match** the entity tag (that would have been returned in the response to a GET request on that resource), then the method **MAY** be performed by the server and **MUST** ignore any If-Modified Since header in the request.

**R28.3** If “*” is supplied in the request and the representation does not exist for that resource, then the method **SHOULD** be performed by the server.

**R28.4** If the entity tags, supplied in a GET or HEAD request, **match** the entity tag of the entity (that would have been returned in the response to a GET request), then the server **MUST** return a 304 Not Modified status code in the response.

**R28.5** If any of the entity tags, supplied in a request (other than GET or HEAD request) **match** the entity tag of the entity (that would have been returned in the response to a GET request) or if “*” is given and any current entity exists for that resource, then the server **MUST NOT** perform the requested method and **MUST** return a 412 Precondition Failed status code in the response.

**R28.6** A request **MAY** include the If-None-Match header.

**R29** The If-Range header **MUST** be used to specify a conditional request for a portion of the entity that is missing if it has not changed and the entire entity if it has changed.

```
“If-Range” “-” “*” | entity-tag | datetime
```

**R29.1** The If-Range field value **MUST** be limited to an element of the value domain:

*entity-tag*

datetime
R29.1.1 The entity-tag field value SHOULD adhere to the format and value domains as specified in IETF’s RFC 2616 syntax for entity tag representation.

Note: See RFC 2616 [Fielding et al (1999)].

R29.1.2 The datetime field value MUST adhere to the format and value domains as specified in IETF’s RFC 1123 (an update to RFC 822).

R29.2 The If-Range header SHOULD only be used in conjunction with the Range header.

R29.3 If the entity tag provided in the If-Range header matches the current entity tag for the entity, then the server SHOULD return the sub-range of the entity using a 206 Partial Content response.

R29.4 If the entity tag provided in the If-Range header does not match the current entity tag for the entity, then the server SHOULD return the entire entity using a 200 OK response.

R29.5 If the client has no entity tag for an entity but has the Last-Modified date, then the client MAY use that date as the datetime field value of the If-Range header.

R29.6 A request MAY include the If-Range header.

R30 The If-Unmodified-Since header MUST be used with a method to make it conditional; the method is performed only if the resource representation has not been modified since the date given in this header.

“If-Unmodified-Since” “: “ datetime

R30.1 The datetime field value MUST adhere to the format and value domains as specified in IETF’s RFC 1123 (an update to RFC 822).

R30.2 If the resource representation has not been modified, then the method MAY be performed by the server.

R30.3 If the resource representation has been modified since the date provided in the If-Modified-Since header, then the server MUST return a 412 Precondition Failed status code in the response.

R30.4 A request MAY include the If-Unmodified-Since header.

R31 The Prefer header MUST be used to indicate that particular server behaviors are preferred by the client, but not required for successful completion of the request.

“Prefer” “: “ preference
preference = token [“ = ” value] [“ ; ” parameter “=” value]
### R31.1

The *preference* field value **MUST** adhere to the format and value domains as specified in IETF’s RFC 7240 for preference representation.

*Note:* See the RFC 7240 [Snell (2014)].

### R31.2

The *preference* field value **MUST** be limited to an element of the value domain:

- "respond-async",
- "wait" " = " *time*,
- "return=representation",
- "return=minimal",
- "/oagi/confirm-message",
- "/oagi/long-polling".

#### R31.2.1

The *respond-async* field value **MUST** be used to indicate that the client prefers the server to respond asynchronously to a request.

#### R31.2.2

The *wait = time* field value **MUST** be used by the client to provide an upper bound on the length of time, in seconds, the client expects it will take the server to process the request one it has been received.

#### R31.2.3

The *return=representation* field value **MUST** be used to indicate that the client prefers that the server include a representation of the current state of the resource in the response to a successful request.

*Note:* This preference is intended to provide a means of optimizing communication between the client and server by eliminating the need for a subsequent GET request to retrieve the current resource representation following a modification [Snell (2014)].

#### R31.2.4

The *return=minimal* field value **MUST** be used to indicate that the client prefers that the server return a minimal response to a successful request.

*Note:* This preference is intended to reduce the amount of data the server is required to return to the client following a request. The definition of what constitutes a minimal response is at that the discretion of the server [Snell (2014)].

#### R31.2.5

The *return=representation* and *return=minimal* field values **MUST NOT** be communicated in a single request.

#### R31.2.6

The */oagi/confirm-message* field value **MUST** be used to indicate that the client prefers that the server include a Confirm Message in the response.

#### R31.2.7

The */oagi/long-polling* field value **MUST** be used to indicate that the client prefers that the server uses the long polling server-push mechanism to respond to a request.
| R31.2.8 | If the request Prefer header preference field value specifies “respond-async”, then the server MUST use an asynchronous communication model.  

*Note:* In the general case, the Prefer header indicates a client preference; it is possible that the server either not recognize or be unable to comply with the client preference. See RFC 7240 Prefer Header [Snell (2014)]. However, in the more specific case of a Web API contract, requirements on server behavior must be agreed upon and supported. |

| R31.3 | If the request Prefer header preference field value specifies “respond-async, wait = time”, and if generating a response will take greater than the time specified, then the server MUST use an asynchronous communication model.  

*Note:* See Patterns for Asynchronous Communication section. |

| R31.4 | If the request Prefer header preference field value specifies “respond-async, wait = time”, and if generating a response will take less than or equal to the time specified, then the server MUST use a synchronous communication model.  

*Note:* See Patterns for Asynchronous Communication section. |

| R31.5 | If the request Prefer header preference field value specifies “return=representation”, then the server MUST provide a representation of the current state of the resource in the response message-body. |

| R31.6 | If the request Prefer header preference field value specifies “return=minimal”, then the server MUST return a minimal response.  

*Note:* The definition of what constitutes a minimal response is at that the discretion of the server [Snell (2014)]. |

| R31.7 | If the request Prefer header preference field value specifies “/oagi/confirm-message”, then the server MUST provide a Confirm Message in the response message-body.  

*Note:* The client’s ability to request a Confirm Message is limited to those cases where a Confirm Message is applicable. The cases are identified in the Confirmation Management section. |

| R31.8 | If the request Prefer header preference field value specifies “/oagi/long-polling”, then the server MUST use the long polling server push mechanism to respond to the client.  

*Note:* The client’s ability to request a long polling is limited to those cases where long polling is applicable (e.g. Event Notifications). See the Pattern for Event Notifications section. |

| R31.9 | A request MAY include the Prefer header. |
R32  The Range header MUST be used to specify the partial range(s) requested of an entity.

"Range" ": " range-specifier

R32.1  The range-specifier field value, using a byte range, SHOULD adhere to the format and value domains as specified in IETF’s RFC 2616 for byte range representation.

Note:  See RFC 2616 [Fielding et al (1999)].

R32.2  If a server supports the Range header, the specified range(s) is appropriate for the entity and the (unconditional or conditional) GET request is successful, then the server modifies what is returned and MUST provide a 206 Partial Content response (instead of the 200 OK response).

R33  The Referer header MUST be used to provide the URI of the resource from which the requested URI was obtained.

"Referer" ": " uri

R33.1  The uri field value MUST adhere to the format and value domains as specified in IETF’s RFC 2616 syntax for URI representation.

Note:  See RFC 2616 [Fielding et al (1999)] and RFC 3986 [Berners-Lee (2005)].

R33.2  If the referrer has its own URI, a request MAY include the Referer header.

R34  The User-Agent header MUST be used to provide identifying information on the client.

"User-Agent" ": user-agent
user-agent = product [ "/" product-version ] | comment

R34.1  The user-agent field value SHOULD adhere to the format and value domains as specified in IETF’s RFC 2616 for user agent representation.

Note:  See RFC 2616 [Fielding et al (1999)].

R34.2  A request MAY include the User-Agent header.

Accept: application/json, application/xml
Accept: application/json; masked=true
Accept: application/json; masked=false
Accept: application/json; q=0.8; masked=true, application/xml; q=0.5; masked=false
Accept-Charset: utf-8
Accept-Encoding: gzip, deflate
6.3 Response Headers

A response header allows the server to pass information about the response (that cannot be communicated in the status-line), about the server, and about further access to the resource (identified in the request URI) to the client.

<table>
<thead>
<tr>
<th>Header Field Name</th>
<th>Obligation</th>
<th>Format &amp; Description</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETag</td>
<td>Conditional - in response</td>
<td>“ETag” “: ” <em>entity-tag</em> &lt;br&gt; Defines the entity tag for use with the If-Match and If-None-Match request headers.</td>
<td>Conditions&lt;br&gt;• Application (API) requirements, used as a concurrency control mechanism for Update requests.&lt;br&gt;• Caching requirements</td>
</tr>
<tr>
<td>Location</td>
<td>Conditional - in response</td>
<td>“Location” “: ” <em>uri</em> &lt;br&gt; Used to redirect the recipient to a location other the request URI for completion of the request or identification of a new resource.</td>
<td>Conditions&lt;br&gt;• 201 Created, 3xx Moved Permanently response status codes.</td>
</tr>
<tr>
<td>Set-Cookie</td>
<td>Conditional - in response</td>
<td>“Set-Cookie” “: ” <em>name “=” value [; options]</em> &lt;br&gt; Used to send name/value pairs (i.e., cookies) for that URI from the server to the user agent.</td>
<td>Conditions&lt;br&gt;• Applicable security standard requirements.&lt;br&gt;• Application (API) requirements, usually</td>
</tr>
<tr>
<td>Header Field Name</td>
<td>Obligation</td>
<td>Format &amp; Description</td>
<td>Condition</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Retry-After</td>
<td>Conditional</td>
<td>“Retry-After” “:” *datetime</td>
<td>Conditions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains either a date time or an integer number of seconds after which the client may try the request again.</td>
<td>• 503 Service Unavailable response status code</td>
</tr>
<tr>
<td>WWW-Authenticate</td>
<td>Conditional</td>
<td>“WWW-Authenticate” “:” *scheme “realm” “=” realm-name</td>
<td>Conditions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifies the authorization scheme and realm required from a client at the requested URI.</td>
<td>• Applicable security standard requirements. and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 401 Unauthorized response status code</td>
</tr>
</tbody>
</table>

Table 3: Response Headers

R35  The ETag header MUST be used to specify the entity tag (*entity-tag*) of the response’s entity.

*Note:* The entity comprises metadata in the form of entity headers and content in the form of an entity-body [Fielding et al. (1999)].

R35.1 The *entity-tag* field value MUST adhere to format and value domain as specified by IETF’s RFC 2616 for entity tag representation.

*Note:* See RFC 2616 [Fielding et al (1999)].

R35.2 The *entity-tag* field value generation MUST NOT be host-specific and ensure that the same value is generated for the same representation.

R35.3 A response MAY include the ETag header.

R36  The Location header MUST be used to redirect the recipient to a location other than the request URI for completion of the request or identification of a new resource.

“Location” “:” *uri*
R36.1  The *uri* field value **MUST** adhere to the format and value domains as specified in IETF’s RFC 7231 syntax for URI representation.

*Note:* See RFC 7231 [Fielding et al (2014)] and RFC 3986 [Berners-Lee (2005)]. RFC 7231 permits the use of relative URIs and fragment identifiers; its predecessor, RFC 2616, did not permit their use.

R36.2  A request or response **MAY** include the *Location* header.

*Note:* The Confirmation Management section provides the conditions for when a *Location* header is to be returned.

R37  The *Set-Cookie* header **MUST** be used to send name/value pairs (i.e., cookies) to be retained for the URI at the user agent.

```
“Set-Cookie” “:” name “=” value [; options]
```

R37.1  The *name=value* of the field value **MUST** adhere to the format and value domains as specified in IETF’s RFC 6265.

*Note:* See IETF RFC 6265 [Barth (2011)].

R37.2  The *options* parameter of the field value **MUST** be limited to one or more of the following:

- “Expires” “=” *datetime*
- “Max-Age” “=” *seconds*
- “Domain” “=” *domain-name*
- “Path” “=” *path-value*
- “Secure”
- “HttpOnly”

R37.2.1  The *Expires=datetime* option **MUST** be used to indicate the maximum lifetime of the cookie, represented as the date and time at which the cookie expires.

R37.2.2  The *Max-Age=seconds* option **MUST** be used to indicate the maximum lifetime of the cookie, represented as the number of seconds until the cookie expires.

R37.2.3  The *Domain=domain-name* option **MUST** be used to specify those hosts to which the cookie will be sent.

R37.2.3.1  The *domain-name* value **SHOULD** adhere to the format and value domains as specified in IETF’s RFC 6265 syntax for domain name representation.

*Note:* See IETF RFC 6265 [Barth (2011)].
R37.2.4 The Path=path-value option MUST be used to identify a set of paths that specifies the scope of the cookie.

R37.2.4.1 The path-value value SHOULD adhere to the format and value domains as specified in IETF’s RFC 6265 syntax for path value representation.

Note: See IETF RFC 6265 [Barth (2011)].

R37.2.5 The Secure attribute option MUST be used to limit the scope of the cookie to secure channels.

R37.2.6 The HttpOnly attribute option MUST be used to limit the scope of the cookie to HTTP requests.

R37.3 A response MAY include the Set-Cookie header.

R38 The Retry-After header MUST be used to indicate either a date time or an integer number of seconds after which the client may try the request again.

“Retry-After” “:” datetime | seconds

R38.1 The datetime field value MUST adhere to the format and value domains as specified in IETF’s RFC 1123 (an update to RFC 822).

R38.2 The seconds field value SHOULD adhere to format and value domain as specified IETF’s RFC 2616 for Delta Seconds.

Note: See RFC 2616 [Fielding et al (1999)].

R38.3 A response MAY include the Retry-After header.

Note: The Confirmation Management section provides the conditions for when a Retry-After header is to be returned.

R39 The WWW-Authenticate header MUST be used to specify the authorization scheme and realm required from a client at the requested URI.

“WWW-Authenticate” “:” scheme “realm” “=” realm-name

R39.1 The scheme realm= realm-name field value MUST adhere to the format and value domains of the applicable security standard.

R39.2 A response MAY include the WWW-Authenticate header.

Examples of the response headers are shown below:

Etag: 737060cd8c284d8af7ad3082f209582d
### 6.4 Entity Headers

An entity-header provides information on the entity-body or, if no entity-body is present, about the resource identified in the request.

<table>
<thead>
<tr>
<th>Header Field Name</th>
<th>Obligation</th>
<th>Format &amp; Description</th>
<th>Condition/Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>Conditional</td>
<td>“Allow” “: &quot; methods&lt;br&gt;Contains a list of methods that are allowed at a request URI.</td>
<td>Conditions&lt;br&gt;• Request method was OPTIONS or&lt;br&gt;• 405 response status code</td>
</tr>
<tr>
<td>Content-Disposition</td>
<td>Conditional</td>
<td>“Content-Disposition” “: &quot; type [&quot;;&quot; disposition-parameter]&lt;br&gt;Specifies the presentational disposition of a message instance or message body part (i.e., inline, attachment) and may be used to indicate a default archival disposition (i.e., a filename).</td>
<td>Condition&lt;br&gt;• Optional for a message instance or body-part of Content-Type: multipart/mixed OR multipart/related&lt;br&gt;• Required for a body-part of Content-Type: multipart/form-data</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>Optional</td>
<td>“Content-Encoding” “:&quot; encoding_schemes&lt;br&gt;Specifies the encoding scheme(s) used for the entity-body.</td>
<td>Usage&lt;br&gt;• The entity-body may require encoding to ensure safe and proper transfer.</td>
</tr>
<tr>
<td>Content-Language</td>
<td>Conditional</td>
<td>“Content-Language” “: &quot; languages&lt;br&gt;Specifies the languages for which the entity-body is intended.</td>
<td>Conditions&lt;br&gt;• An entity-body exists</td>
</tr>
<tr>
<td>Content-Length</td>
<td>Optional</td>
<td>“Content-Length” “: &quot; n&lt;br&gt;Specifies the length of the data (in bytes) of the entity-body in a message instance.</td>
<td></td>
</tr>
<tr>
<td>Header Field Name</td>
<td>Obligation</td>
<td>Format &amp; Description</td>
<td>Condition/Usage</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>----------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Content-Range</td>
<td>Conditional</td>
<td>“Content-Range” “; ” “bytes” n “-“ m “/” length</td>
<td>A partial resource representation is being sent.</td>
</tr>
<tr>
<td></td>
<td>- in request - in response</td>
<td>Specifies where the partial resource representation should be inserted followed by the total size of the full resource representation body.</td>
<td></td>
</tr>
<tr>
<td>Content-Type</td>
<td>Conditional</td>
<td>“Content-Type” “; ” “type &quot;&quot; subtype [&quot;;&quot; &quot;masked&quot; &quot;=&quot; &quot;true&quot; ] [&quot;;&quot; &quot;false&quot; ] [&quot;;&quot; &quot;boundary&quot; &quot;=&quot; &quot;boundary&quot; ]</td>
<td>An entity-body exists.</td>
</tr>
<tr>
<td></td>
<td>- in request - in response</td>
<td>Describes the media type and subtype of an entity-body.</td>
<td></td>
</tr>
<tr>
<td>Expires</td>
<td>Mandatory</td>
<td>“Expires” “; ” datetime</td>
<td>Specifies the date and time the representational state of the resource is considered stale.</td>
</tr>
<tr>
<td></td>
<td>- in response</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Last-Modified</td>
<td>Mandatory</td>
<td>“Last Modified” “; ” datetime</td>
<td>Specifies the date and time the representational state of the resource was last modified.</td>
</tr>
<tr>
<td></td>
<td>- in response</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Link</td>
<td>Conditional</td>
<td>link = “&lt;&quot;uri “;&quot; “rel” “=&quot; “rel” “=&quot; &quot;relation-type [ &quot;anchor &quot;=&quot; uri [&quot;;&quot; &quot; target-attributes]</td>
<td>Application (API) requirements</td>
</tr>
<tr>
<td></td>
<td>- in request</td>
<td>Expresses a typed relationship with another resource.</td>
<td>Link to Confirm Message entity is provided, as described in the section, Confirmation Management</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: The value of relation-type is a quoted string.</td>
<td>Link to Status Monitor, as described in the section, Asynchronous Communication</td>
</tr>
</tbody>
</table>

Table 4: Entity Headers

R40 The Allow header MUST be used to contain a list of methods that are allowed at a request URI.

“Allow” “; ” methods
R40.1 The methods field value **MUST** adhere to the value domain as specified in IETF’s RFC 2616 for method representation.

R40.2 A response **MAY** include the **Content-Language** header.

*Note:* The Resource and Confirmation Management sections provide the conditions for when an Allow header is to be returned.

R246 The **Content-Disposition** entity header **MUST** be used to tag a message instance or message body-part with the intended presentational semantics (e.g. display inline, display of an attachment).

“Content-Disposition” “: ” type [“;” disposition-parameter]

R246.1 The **Content-Disposition** entity header **MAY** be included as a message header or message body-part header.

R246.2 The type field value **MUST** adhere to the format and value domain as specified in IETF’s RFC 2183.

*Note:* See RFC 1806 [Troost et al. (1995)] and RFC 2183 [Troost et al. (1997)]

R246.3 The disposition-parameter field value **MUST** adhere to the format and value domain as specified in IETF’s RFC 2183.

*Note:* See RFC 1806 [Troost et al. (1995)] and RFC 2183 [Troost et al. (1997)]

R41 The **Content-Encoding** header **MUST** be used to specify the encoding scheme applied to the entity-body.

“Content-Encoding” “:” encoding_schemes

R41.1 The encoding_schemes field value **MUST** be limited to an element of the value domain as specified by IANA as the set of registered HTTP Content-Coding Values.

*Note:* See the IANA Registry of HTTP Content-Coding Values [IANA(2013c)].

R41.2 A request or response **MAY** include the **Content-Encoding** header.

R42 The **Content-Language** header **MUST** be used to specify the language for which the entity-body is intended.

“Content-Language” “: ” languages
R42.1 The languages field value **MUST** adhere to the format and value domains as specified in IETF’s RFC 5646.


R42.2 A request or response **MAY** include the **Content-Language** header.

The **Content-Length** header, used to specify the length of the data communicated in the entity-body of a message instance, allows the message receiver to determine whether it has read the correct number of bytes from the connection. In addition, a client may send a HEAD request to determine the size of the entity-body before requesting it. [Masse (2011)]

R43 The **Content-Length** header **MUST** be used to specify the length (i.e. size) of the data communicated in the entity-body in a message instance.

```
“Content-Length” “:” n
```

R43.1 The n field value **MUST** adhere to the format and value domains as specified in IETF’s RFC 2616 syntax for content length representation.

*Note: See RFC 2616 [Fielding et al (1999)].*

R43.2 A request or response **MAY** include the **Content-Length** header.

R44 The **Content-Range** header **MUST** be used to specify the bytes where the partial resource representation is to be inserted in the full resource representation and the total byte size of the full resource representation.

```
“Content-Range” “:”
“bytes” n “-” m “/” length
```

R44.1 The bytes n-m/length field value **MUST** adhere to the format and value domains as specified in IETF’s RFC 2616 syntax for content range representation.

*Note: See RFC 2616 [Fielding et al (1999)].*

R44.2 A request or response **MAY** include the **Content-Range** header.
R45  The Content-Type header MUST be used to describe the media type, subtype, masking and body-part boundary of an entity body within a request or response message-body.

```
“Content-Type” “: ” type “/” subtype [“;” “masked” “=” “true” | “false”] [“;” “boundary” “=” “boundary”]
```

Note: In the case of a HEAD request, the Content-Type header is used to describe the media type that would have been sent in response to a GET request.

R45.1  The type/subtype field value MUST adhere to the format and value domains governed by IANA as the set of registered media types or the case of HTML forms, the format and value domain governed by W3C as the set of form content types.

Note: See the IANA Registry of MIME Media Types [IANA], [Raggett (1999)].

R45.2  The type/subtype field value of the response MUST match a media type specified in the Accept header of the request.

R45.3  A request or response MAY include the Content-Type header.

R45.4  Using the masked parameter, each type/subtype value MAY be associated with a masked value.

R45.4.1  The masked value MUST be limited to an element of the value domain:

"true"
"false"

R45.4.1.1  The "true" value MUST be used to indicate that sensitive data is masked.

R45.4.1.2  The "false" value MUST be used to indicate that sensitive data is unmasked.

R45.5  Using the boundary parameter, body-part boundary delimiter MAY be specified.

---

5  According to the HTTP 1.1 Specification., 3.7 Media Types, parameters may follow the type/subtype in the form of attribute/value pairs. This approach of managing the masking at the media type-level (vs. managing it separately from the media type, i.e., at the message level) has a couple of advantages. In the case of multipart Content-Type (where a single message's entity-body may contain different media types), masking may be separately specified at the "part" level. In the case of multiple acceptable media types in a single request, masking may be separately specified for each media type. [Fielding et al. (1999)]
| R45.5.1 | The **boundary** parameter value **MUST** adhere to the format and value domains as specified in IETF’s RFC 2046.  

*Note*: See IETF RFC 2046 [Freed (1996)] |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R46</td>
<td>The <strong>Expires</strong> header <strong>MUST</strong> be used to indicate the date and time when the resource representation is considered stale.</td>
</tr>
<tr>
<td>R46.1</td>
<td>The <strong>datetime</strong> field value <strong>MUST</strong> adhere to the format and value domains as specified in IETF’s RFC 1123 (an update to RFC 822).</td>
</tr>
</tbody>
</table>
| R6.2 | A response **MAY** include the **Expires** header.  

*Note*: The **Expires** header is required for HTTP 1.0 caches [Masse (2011)]. |
| R47 | The **Last-Modified** header **MUST** be used to indicate the date and time when the resource representation was last modified. |
| R47.1 | The **datetime** field value **MUST** adhere to the format and value domains as specified in IETF’s RFC 1123 (an update to RFC 822). |
| R47.2 | A response **SHOULD** include the **Last-Modified** header. |
| R48 | The **Link** header **MUST** be used to express a typed relationship with another resource.  

```
"Link" : "link"  
link = "<" uri "" ; " rel" = "relation-type" [ "anchor" = " uri" ] [: ; " target-attributes"]
```

| R48.1 | The **link** field value **MUST** adhere to the format and value domains as specified in IETF’s RFC 5988 syntax for link representation.  

*Note*: See IETF RFC 5988 [Nottingham (2010)]. |
| R48.2 | The **relation-type** of the **link** field value **MUST** be limited to the elements of the value domain governed by IANA as the set of registered link relations and OAGi for specific extensions.  

The OAG value domain extensions include:  
/oagi/confirm-message  
/oagi/callback  
/oagi/processing-status  
/oagi/request-result |
| R48.2.1 | The **/oagi/confirm-message** value **MUST** be used to inform a client that the URI of the **link** field value identifies a Confirm Message location. |
R48.2.2 The `/oagi/callback` value **MUST** be used to inform a client that the URI of the `link` field value identifies a callback location.

R48.2.3 The `/oagi/processing-status` value **MUST** be used to inform a client that the URI of the `link` field value identifies a processing-status location.

R48.2.4 The `/oagi/request-result` value **MUST** be used to inform a client that the URI of the `link` field value identifies a request-result location.

**R48.3** A response **MAY** include the Link header.

Examples of the entity headers are shown below:

<table>
<thead>
<tr>
<th>Header</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>GET, HEAD</td>
</tr>
<tr>
<td>Content-Disposition</td>
<td>attachment; filename=&quot;att-1111-1.png&quot;</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>gzip</td>
</tr>
<tr>
<td>Content-Language</td>
<td>en-GB</td>
</tr>
<tr>
<td>Content-Length</td>
<td>250</td>
</tr>
<tr>
<td>Content-Range</td>
<td>bytes 2145-7431/14323</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json; masked=true</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json; masked=false</td>
</tr>
<tr>
<td>ETag</td>
<td>&quot;737060cd8c284d8af7ad3082f209582d&quot;</td>
</tr>
<tr>
<td>Expires</td>
<td>Thu, 31 Dec 2013 17:00:00 GMT</td>
</tr>
<tr>
<td>Last-Modified</td>
<td>Tue, 15 Nov 2013 14:45:00 GMT</td>
</tr>
<tr>
<td>Link</td>
<td>&lt;<a href="http://api.abc.com/hr/v1/Confirm">http://api.abc.com/hr/v1/Confirm</a> Messages/abc102030xyz&gt;; rel=&quot;/oagi/confirm-message&quot;; method=&quot;GET&quot;</td>
</tr>
<tr>
<td>Link</td>
<td>&lt;<a href="http://api.abc.com/pr/v1/associates/12121212/timeCards">http://api.abc.com/pr/v1/associates/12121212/timeCards</a> &gt;; rel=&quot;/oagi/callback&quot;; method=&quot;POST&quot;</td>
</tr>
</tbody>
</table>

### 6.5 Custom Headers

Custom headers shouldn't be used to change the behavior of the HTTP methods. Custom headers should be used for informational purposes only; clients and servers should not fail when they do not find expected custom headers. Information that is conveyed through a custom header should not be needed for the correct interpretation of a request or response. [Masse (2011)] This objective or constraint serves to promote broader interoperability.
<table>
<thead>
<tr>
<th>Header Field Name</th>
<th>Obligation</th>
<th>Format &amp; Description</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAGi-Allow-CustomOperator</td>
<td>Conditional</td>
<td>&quot;OAGi-Allow-CustomOperator&quot; &quot;::&quot; custom-operators</td>
<td>Conditions • Request method was OPTIONS and custom operator exists or 400 response status code on request for controller resource invocation.</td>
</tr>
<tr>
<td>OAGi-Context-ExpressionID</td>
<td>Optional</td>
<td>&quot;OAGi-Context-ExpressionID&quot; &quot;::&quot; identifier</td>
<td>Conditions • Application (API) requirements • Routing requirements • Logging requirements</td>
</tr>
<tr>
<td>OAGi-CorrelationID</td>
<td>Conditional</td>
<td>&quot;OAGi-CorrelationID&quot; &quot;::&quot; identifier</td>
<td>Conditions • Application (API) requirements • Logging requirements</td>
</tr>
<tr>
<td>OAGi-ConversationID</td>
<td>Optional</td>
<td>&quot;OAGi-ConversationID&quot; &quot;::&quot; identifier</td>
<td>Contains the identifier of the conversation in which the request or response participates. A conversation is the coordinated exchange of multiple messages between two or more partners. The scope of the identifier spans an entire conversation.</td>
</tr>
<tr>
<td>OAGi-IntermediaryID</td>
<td>Optional</td>
<td>&quot;OAGi-IntermediaryID&quot; &quot;::&quot; identifier</td>
<td>Contains the identifier of the system that acts as an intermediary between the sending and receiving systems.</td>
</tr>
<tr>
<td>OAGi-MessageID</td>
<td>Optional</td>
<td>&quot;OAGi-MessageID&quot; &quot;::&quot; identifier</td>
<td>Contains the identifier of the message instance (i.e. the request or response).</td>
</tr>
<tr>
<td>Header Field Name</td>
<td>Obligation</td>
<td>Format &amp; Description</td>
<td>Condition</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>OAGi-OriginatorID</td>
<td>Optional</td>
<td>“OAGi-OriginatorID” “.” “identifier”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains the identifier of the system that initiated (i.e. originated) the need for</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>the request to be created.</td>
<td></td>
</tr>
<tr>
<td>OAGi-ReferenceID</td>
<td>Optional</td>
<td>“OAGi-ReferenceID” “.” “identifier”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains the identifier of the business task instance that initiated the need for</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>the request to be created.</td>
<td></td>
</tr>
<tr>
<td>OAGi-ScenarioID</td>
<td>Optional</td>
<td>“OAGi-ScenarioID” “.” “identifier”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains the identifier of the business scenario in which the request is participating.</td>
<td></td>
</tr>
<tr>
<td>OAGi-SenderID</td>
<td>Optional</td>
<td>“OAGi-SenderID” “.” “identifier”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains the identifier of the system that sent a request.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- in request</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- in response</td>
<td></td>
</tr>
<tr>
<td>OAGi-TaskID</td>
<td>Optional</td>
<td>“OAGi-TaskID” “.” “identifier”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains the identifier for the business task (command or event) that initiated the</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>need for the request to be created.</td>
<td></td>
</tr>
<tr>
<td>OAGi-UserID</td>
<td>Optional</td>
<td>“OAGi-UserID” “.” “identifier”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains the identifier of the user that initiated the request.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- in request</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- in request</td>
<td></td>
</tr>
<tr>
<td>OAGi Custom Header</td>
<td>Conditional</td>
<td>“OAGiX-” “custom-field-name” “.” “field-value”</td>
<td></td>
</tr>
<tr>
<td>Extension Pattern</td>
<td></td>
<td>Used for custom header extensions (e.g. application specific) that not already</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>defined.</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Custom Headers

R49  A custom header **MUST** be defined using the following convention:

`Namespace-` “`HeaderName`”, where Namespace identifies a scope and HeaderName represents the name associated with the Header.
R52 The OAGi-Allow-CustomOperator header MUST be used to contain a list of custom operators that are allowed in a request at a specified URI.

“OAGi-Allow-CustomOperator” “:: ” custom-operators

R52.1 The custom-operators field value MUST be limited to the elements of the the value domain of custom operators as defined by an API Specification and allowable for a resource at a specified URI.

R52.1.1 The custom operators MUST adhere to the rules for URI path and resource model representation.

R52.2 A response MAY include the OAGi-Allow-CustomOperator header.

R55 The OAGi-Context-ExpressionID header MUST be used to contain the identifier of the context expression for the request.

“OAGi-Context-ExpressionID” “:: ” identifier

Note: A context is indicated by a context expression that specifies a set of context nodes (of a graph) that can be resolved to context values that together represent a specific context. A content node is associated with a context category (e.g. business process, geopolitical).

R55.1 A request MAY include the OAGi-Context-ExpressionID header.

R56 The OAGi-CorrelationID header MUST be used to contain the identifier of the related request.

“OAGi-CorrelationID” “:: ” identifier

R56.1 The identifier field value MUST be the identifier (i.e. OAGi-MessageID header field value) of the request related to the response.

R56.2 A request or response MAY include the OAGi-CorrelationID header.

Note: A callback request, as used in the asynchronous service provider push pattern, is an example of where the request may include this header.

R261 The OAGi-ConversationID header MUST be used to contain the identifier of the conversation in which the request or response participates.

“OAGi-ConversationID” “:: ” identifier

Note: A conversation is the coordinated exchange of multiple messages between two or more partners. The scope of the identifier spans an entire conversation.
R261.1 The **identifier** field value **MUST** be globally unique.

R261.2 A request or response **MAY** include the **OAGi-ConversationID** header.

R290 The **OAGi-IntermediaryID** header **MUST** be used to contain the identifier of the system that acts as an intermediary between the sending and receiving systems.

<table>
<thead>
<tr>
<th><strong>“OAGi-IntermediaryID” “: ” identifier</strong></th>
</tr>
</thead>
</table>

R290.1 The **identifier** field value **MUST** be limited to an element of a defined identifier value domain.

R290.2 A request **MAY** include the **OAGi-IntermediaryID** header.

R57 The **OAGi-MessageID** header **MUST** be used to contain the identifier of the message instance (i.e. the request or response).

<table>
<thead>
<tr>
<th><strong>“OAGi-MessageID” “: ” identifier</strong></th>
</tr>
</thead>
</table>

R57.1 The **identifier** field value **MUST** be globally unique.

R57.2 A request or response **MAY** include the **OAGi-MessageID** header.

R284 The **OAGi-OriginatorID** header **MUST** be used to contain the identifier of the system that initiated (i.e. originated) the need for the request to be created.

<table>
<thead>
<tr>
<th><strong>“OAGi-OriginatorID” “: ” identifier</strong></th>
</tr>
</thead>
</table>

*Note:* An event message, that communicates the event occurrence as a result of a command request, may include this header to communicate the initiator or originator. This header is not intended to be used as a correlation identifier for messages across systems participating in a collaboration.

R284.1 The **identifier** field value **MUST** be limited to an element of a defined identifier value domain.

R284.2 A request **MAY** include the **OAGi-OriginatorID** header.

R288 The **OAGi-ReferenceID** header **MUST** be used to contain the identifier of the business task instance that initiated the need for the request to be created.

<table>
<thead>
<tr>
<th><strong>“OAGi-ReferenceID” “: ” identifier</strong></th>
</tr>
</thead>
</table>

R288.1 The **identifier** field value **MUST** be globally unique.

R288.2 A request **MAY** include the **OAGi-ReferenceID** header.
**R289** The OAGi-ScenarioID header **MUST** be used to contain the identifier of the business scenario in which the request is participating.

```
“OAGi-ScenarioID” “: ” identifier
```

**R289.1** The `identifier` field value **MUST** be limited to an element of a defined identifier value domain.

**R289.2** A request **MAY** include the `OAGi-ScenarioID` header.

---

**R290** The OAGi-SenderID header **MUST** be used to contain the identifier of the system that sent a request.

```
“OAGi-SenderID” “: ” identifier
```

**R290.1** The `identifier` field value **MUST** be limited to an element of a defined identifier value domain.

**R290.2** A request **MAY** include the `OAGi-SenderID` header.

---

**R291** The OAGi-TaskID header **MUST** be used to contain the identifier of the business task (command or event) that initiated the need for the message to be created.

```
“OAGi-TaskID” “: ” identifier
```

**R289.1** The `identifier` field value **MUST** be limited to an element of a defined identifier value domain.

**R289.2** A request **MAY** include the `OAGi-TaskID` header.

---

**R58** The OAGi-UserID header **MUST** be used to contain the identifier of the user that initiated the request.

```
“OAGi-UserID” “: ” identifier
```

*Note:* This header is not intended for authentication purposes; it may be used when the server needs knowledge of the user who made the request.

**R58.1** A request **MAY** include the `OAGi-UserID` header.

Examples of custom headers are shown below:

- `OAGi-CorrelationID: 1070fdc4-0222-410d-9398-c51e9176299d`
- `OAGi-MessageID: 1070fdc4-0222-410d-9398-c51e9176299d`
- `OAGi-UserID: jsmith@gmail.com`
An extension pattern is provided for custom headers that are not already defined in this specification. The extensions should be limited to application-specific requirements.

<table>
<thead>
<tr>
<th>R63</th>
<th>Application-specific headers <strong>MAY</strong> be defined using the custom header extension pattern.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R63.1</td>
<td>A custom header extension <strong>MUST</strong> be named according to the following representation pattern:</td>
</tr>
<tr>
<td></td>
<td>“OAGX-” <em>custom-field-name</em> “: ” <em>field-value</em></td>
</tr>
<tr>
<td>R63.2</td>
<td>A custom header extension <strong>MUST</strong> be registered with OAGi.</td>
</tr>
</tbody>
</table>

*Note:* Registration requires submission, definition and approval of the custom header. Central registration facilitates coherency and reuse across OAGi custom header extensions.
6.6 Caching

A caching mechanism is a local data store that manages copies of resource representations. Caching offers several benefits including: reduced client perceived latency, increased reliability, and reduced load on the servers. Caching mechanisms may exist anywhere along the request/reply chain (client network, content delivery network and server network).

RFC 2616 [Fielding et al (1999)] states, “Caching would be useless if it did not significantly improve performance. The goal of caching in HTTP/1.1 is to eliminate the need to send requests in many cases, and to eliminate the need to send full responses in many other cases. The former reduces the number of network round-trips required for many operations; we use an "expiration" mechanism for this purpose. The latter reduces network bandwidth requirements; we use a "validation" mechanism for this purpose."

6.6.1 Expiration Mechanism

Servers assign expiration times to responses with the expectation that the entity will not significantly change before the expiration time. The expiration mechanism applies only to responses taken from a cache and not first-hand responses to the client request. The expiration time is the primary mechanism for avoiding requests to the origin server allowing a response from a cache to satisfy subsequent requests. A response in a cache that has exceeded the expiration time is known as a "stale" entry; one that has not exceeded the expiration time is known as "fresh" entry.

Servers specify expiration times using the Expires header or the “max-age” directive of the Cache-Control header.

R64 A served representation that is to be cached SHOULD include:
- a Cache-Control: max-age= seconds header,
- a Date header
- and an Expires header (for legacy HTTP 1.0 caches).

Note: Inclusion of the Date header helps clients compute the freshness of the representation. [Masse (2011)]

The “must-revalidate” directive of the Cache-Control header may be used by an origin server to force an HTTP/1.1 cache to revalidate a cache entry once it becomes stale.

R64.1 If the origin server requires revalidation of a cache entry on any subsequent use of a response, then the server representation SHOULD include a Cache-Control: must-revalidate header.

Note: A server may also assign an expiration time in the past to require validation of a cache entry of any subsequent response; however, a cache may be configured to ignore a server's designated expiration time.

R64.2 If the Cache-Control: must-revalidate is present in a response, then the cache MUST NOT use the entry after it becomes stale to satisfy to a subsequent request without first revalidating it with the origin server.
A served representation that is *not* to be cached **SHOULD** include:

- a **Cache-Control: no-cache, no-store** header
- a **Pragma: no-cache** header (required by legacy HTTP1.0 caches)
- an **Expires: 0** header (required by legacy HTTP1.0 caches) [Masse (2011)]

### 6.6.2 Validation Mechanism

When a cache has a stale entry that may be used as a response to a client’s request, it must first check with the origin server (or intermediate cache with a fresh response) to determine if the cache entry is still usable. This process is known as “validating” the cache entry.

HTTP/1.1 supports cache validators that are used by the caching mechanism to validate the cache entry. The cache validator is attached to the origin server’s full response and stored along with the cache entry. There are two cache-validators: the Last-Modified Date (the value of the **Last-Modified** entity header) value) and the Entity Tag (the value of the **ETag** response header).

When a client makes a request for a resource for which there exists a stale cache entry, the cache requests the server to validate the entry by including the validators in associated headers in the request for comparison with the current validators at the server.

HTTP/1.1 supports both **positive and negative approaches to cache validation**. In other words, it is possible to request either that a method be performed if and only if a validator matches or if and only if no validators match, respectively. For the positive approach to cache validation, if the comparison results in a match for Last-Modified Date and Entity Tag validators communicated in the **If-Modified-Since** and/or **If-None-Match** (respectively), then the server returns a **304 Not Modified** response. For the negative approach to cache validation, if the comparison results in *no* match for the Last Modified Date and Entity Tag validators communicated in the **If-Unmodified-Since, If-Match** (respectively), then the server returns a **304 Not Modified** response.

*Note:* Additional information on caching-related headers that may be set as a result of message processing, are provided in the Confirmation Management section.
The figure shown below diagrams the Positive Approach to Cache Validation.

Figure 3: Positive Approach to Cache Validation
Notice that when the cache entry is not stale, the need to send the request to the server is eliminated. When the cache entry is stale and the validators of the cache match those at the server, the need to send a full response is eliminated. In both cases, overall performance is improved.

### 7 Message Resource Identification

Each distinct web-based concept is known as a resource and may be addressed through a unique identifier. RESTful Web Services use Uniform Resource Identifiers (URIs) used to identify resources.

#### 7.1 Resource Types

Three types of resources discussed by [Masse (2011)] include:

**Collection**

Collection is a server-managed directory of resources. A client may request to add, update or delete resources in a collection. The collection has ownership of the contained resources and manages the resources’ URIs.

In each of the following examples the URI identifies a collection resource where *service-domains* and *services* are collection resources.

- http://api.abc.com/service-domains
- http://api.abc.com/service-domains/hr/services

**Instance**

Refers to an individual resource of the collection and is often referred to as a document instance, comprising properties and links to other resources.

In each of the following examples the URI identifies an instance resource where *hr* and *employeeManagement* identify instance resources.

- http://api.abc.com/service-domains/hr
- http://api.abc.com/service-domains/hr/services/employeeManagement

**Controller**

Controller represents a procedural concept. Controller resources are used to invoke application-specific functions that are not supported by one of the standard HTTP methods (supporting create, read, update and delete operations).

The following example is a controller resource where *convert* identifies a currency conversion function of the US Dollar (USD). The USD amount to be converted and the target currency is communicated in the message-body of the request and the converted amount is communicated in the message-body of the response.

- http://api.abc.com/core/v1/currencies/USD/convert

**Instance Resource Set**

An instance resource set is a set of instance resources that is determined by a server (at a point in time) to satisfy the set’s membership criteria (i.e., selection, filter, expansion and
search criteria) of a resource management operation (e.g. GET request) upon a collection resource. It may be considered to be a kind of Dynamic Virtual Collection.

7.2 URI Design and Format

The generic URI syntax is described in IETF’s RFC 3986 [Berners-Lee (2005)] as:

$$\text{URI} = \text{scheme} \cdot \text{authority} \cdot \text{path} \cdot \text{query} \cdot \text{fragment}$$

The syntax consists of a hierarchical sequence of components:

- A required scheme component that refers to a specification for assigning identifiers within that scheme (e.g. http).
- An optional authority component that refers to a naming authority (e.g. api.abc.com)
- A required path component that identifies a resource within the scope of the URI’s scheme and naming authority (if any); it is usually organized in a hierarchical form.
- An optional query component, containing non-hierarchical data, that along with data in the path component identifies a resource within the scope of the URI’s scheme and naming authority (if any).
- An optional fragment component allows indirect identification of a secondary resource by reference to a primary resource and additional identifying information.

Since the URIs for our Web APIs are exposed for consumption both internally and externally, it is essential that the vocabulary used in the URI correspond to the enterprise shared (i.e., canonical) vocabulary.

<table>
<thead>
<tr>
<th>R66</th>
<th>Vocabulary used in the URI components MUST belong to the enterprise shared (i.e., canonical) vocabulary.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R67</td>
<td>URIs represented in HTTP messages (e.g. headers, entity-body) MUST be absolute (i.e. include the scheme and authority).</td>
</tr>
</tbody>
</table>

7.2.1 Service Owner

URI schemes may include a hierarchical element for a naming authority where the governance of the name space defined by the remainder of the URI is delegated to that authority.

7.2.1.1 Design

The generic authority syntax is further described by RFC 3986 [Berners-Lee (2005)] as:

$$\text{authority} = [ \text{userinfo} \cdot \text{@} \cdot \text{host} \cdot [ \cdot \text{port} \cdot ] ]$$

It provides a common means for distinguishing an authority based on a registered name or server address, along with optional port and user information.

The host subcomponent of the authority is identified by either an IP literal (encapsulated in brackets), an IP address (in dotted decimal form) or a registered name.

The representation of the host subcomponent in an API’s URI authority may differ to support the specific requirements of that APIs operational use; therefore, it is not possible (in this specification) to restrict the authority host subcomponent to be of one type of representation (e.g. registered name). For example, an API that has generally available to the partner community through a central, external gateway may use a registered name for the host subcomponent, while an API that has limited availability to a particular set of devices, such as time clocks, may use an IP address for the host subcomponent.
The host subcomponent of a URI Authority MAY be represented by one of the following:
- IP literal
- IP address
- registered name

A registered name is usually defined within a host or service name registry. The most common registry mechanism is the Domain Name System (DNS). DNS registered names, also known as domain names, consist of a sequence of domain labels separated by the period, ".". For a fully qualified domain name, the rightmost label is referred to as the top-level domain. Labels to the left of the top-level domain are referred to as subdomains; the rightmost subdomain is referred to as first-level subdomain.

The subdomains of a domain name, exposed in a URI Authority, should be consistently named.

For any URI Authority host, represented as a domain name, the following rules apply:

R69 The first-level subdomains SHOULD identify the service(s) owner.

R70 The second-level subdomain MAY identify the service owner.

R71 The subdomains identifying the service owners SHOULD be consistently named.

The following example shows the host subcomponent as a registered name (domain name) where "abc" identifies the service owner.

http://abc.com

The following example shows the host subcomponent as a registered name (domain name) where "product-xyz" and "abc" identifies the service owner.

http://product-xyz.abc.com

7.2.2 API and Developer Domains

API and developer domains are key concepts for exposing and managing Web APIs; these concepts are represented in either the URI Authority or URI Path components of the URI. Given the required variability of the authority host subcomponent (described above) it is not possible to limit the representation of these concepts to one component of the URI. This section describes how these concepts may be represented.

7.2.2.1 API Domain

The “api” concept serves the access point for all exposed APIs of a set of services.

R72 The term “api” MUST be used to represent the API concept in the URI as the access point for all exposed APIs of a set of services.

R285 The term “api” MUST be represented in the subdomain or in the first URI path segment.

The following example shows the host subcomponent as a registered name (domain name) where “abc” is the service(s) owner and the “api” concept is represented as a subdomain.

http://api.abc.com
The following examples show the host subcomponent as a registered name and an IP address, respectively, where the “api” concept is represented as a URI path segment.

<table>
<thead>
<tr>
<th>URI</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://product-xyz.abc.com/api">http://product-xyz.abc.com/api</a></td>
<td></td>
</tr>
<tr>
<td><a href="http://170.146.39.174/api">http://170.146.39.174/api</a></td>
<td></td>
</tr>
</tbody>
</table>

### 7.2.2.2 Developer Domain

The "developer" concept represents the developer public portal that helps clients with API-related documentation, discussion forums, etc.

**R73** The term “developer” **MUST** be used to represent the developer concept in the URI as the access point for the developer portal of a set of services.

**R286** The term “developer” **MUST** be represented either in the subdomain before the service owner or in the first URI path segment.

The following example shows the host subcomponent as a registered name (domain name) where “abc” is the service(s) owner and the “developer” concept is represented as a subdomain.

http://developer.abc.com/

The following examples show the host subcomponent as a registered name and an IP address, respectively, where the “developer” concept is represented as a URI path segment.

<table>
<thead>
<tr>
<th>URI</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://product-xyz.abc.com/developer">http://product-xyz.abc.com/developer</a></td>
<td></td>
</tr>
<tr>
<td><a href="http://170.146.39.174/developer">http://170.146.39.174/developer</a></td>
<td></td>
</tr>
</tbody>
</table>

### 7.2.3 URI Path

#### 7.2.3.1 Service Domain

Service domains are used to organize, collect and manage service inventories; the set of services may be independently owned and operated.

If the service owner, represented in the URI authority, is at a broad or high-level (e.g. at an enterprise or business unit level), then representation of the service domain in the URI is required to support managing the domain’s services’ APIs.

Depending upon whether or not the host subcomponent of the URI authority includes the “api” concept, the service domain name may occur in either the first path segment or path segment that is subsequent to the “api” concept.

**R74** The service domain value of the Web API **MUST** be represented in the URI path segment.

**R74.1** If the “api” concept is represented in the URI authority, then the first path segment **MUST** identify the service domain.

**R74.2** If the “api” concept is represented as a path segment in the URI path, then the subsequent path segment **MUST** identify the service domain.
R75  The service domain value **MUST** be limited to the elements of the value domain governed by the service owner (organization), as the set of registered service domain values.

In the following example, hr represents the human resources service domain where the “api” concept is represented in the URI authority.

http://api.abc.com/hr

In the following example, hr represents the human resources service domain where the “api” concept is represented in the URI path.

http://product-xyz.abc.com/api/hr

7.2.3.2 API Version

The major version identifier of each web API must precede the resources managed through the API. The path segment in the URI must be used to represent the service version.

R76  The major version identifier of the Web API **MUST** be represented in the URI path segment.

R76.1 If the “api” concept is represented in the URI authority and the first path segment identifies the service domain, then the second path segment **MUST** identify the major version identifier of the service.

R76.2 If the “api” concept is represented as a path segment in the URI path, and the next path segment identifies the service domain, then the subsequent path segment **MUST** identify the major version identifier of the service.

In the following example, v1 represents version 1 of the associates Web API where the “api” concept is represented in the URI authority.

http://api.abc.com/hr/v1/associates

In the following example, v1 represents version 1 of the associates Web API where the “api” concept is represented in the URI path.

http://product-xyz.abc.com/api/hr/v1/associates

7.2.3.3 Resource Model

The URI path (following the version identification) represents the RESTful Web API’s resource model. Each forward slash path segment identifies a unique resource in the model.

R77  For the URI resource model, each path segment **MUST** identify a unique resource.

The following example identifies a collection resource.

http://api.abc.com/hr/v1/associates

The following example identifies an instance resource.

http://api.abc.com/hr/v1/associates/12121212
The following rules provide for consistent resource naming according to the different resource types, discussed above.

**R78** A named instance resource **SHOULD** be named by a singular noun or noun phrase path segment.

The following example shows a named resource instance of *mary-jones*.

```
http://api.abc.com/hr/v1/associates/12121212/contacts/mary-jones
```

**R79** A collection resource **SHOULD** be named by a plural noun or noun phrase path segment.

The following example shows a collection resource named *associates*.

```
http://api.abc.com/hr/v1/associates
```

**R80** A controller resource **SHOULD** be named with a verb or verb phrase so as to communicate its operator.

**R81** A controller resource name **SHOULD** occur as the last segment in a URI path.

The following example shows a controller resource, *hire*.

```
http://api.abc.com/hr/v1/associates/12121212/hire
```

As mentioned above, each path segment of the URI identifies a unique resource in the resource model of the RESTful Web API. The goal of resource modeling is to establish the API’s key concepts. Some guidelines follow:

- URI path segments should not be forced to have the same hierarchy as the payload (body) of a message or underlying object class model. Treat the URI as the identifier only, not as a predictor of the message body layout itself. It is fine (and common) that the message body hierarchy varies over time, even for the same (non-varying) URI.
- As such there is no standard pattern for representing object class model relationships (e.g. associations, aggregations, compositions) in a resource model.
- The principle of addressability requires that every resource have its own URI [Richardson (2013)]

A resource model should not include concepts that do not serve to identify the resource. For example, concepts that might be needed to route requests to a specific application or application instance should not exist as part of the resource model.

**R82** Concepts used solely for routing a request **SHOULD NOT** exist in the URI resource model.

### 7.2.3.4 Format

The URI path may comprise one or more path segments. A path segment represents

**R83** For the URI path, the forward slash character, “/”, **MUST** be used to indicate a hierarchical relationship.

The following example illustrates use of the forward slash.
To avoid possible confusion between the lack of a trailing slash character and the existence of a trailing slash character, a URI must not include a trailing slash.

**R84** For the URI path, a trailing forward slash character, “/”, **MUST** not be used.

The following example illustrates improper use of the forward slash.

http://api.abc.com/hr/v1/associates/ is not allowed.

In order to ease the readability of URI paths the hyphen character “-” should be used in multi-part (e.g. multi-word) segments.

**R85** For URI path segments, consisting of more than a single word, a hyphen character “-” **SHOULD** be used to separate the words.

**R85.1** The use of the hyphen character “-” in a URI path segment **MUST** be limited to the separation of words.

The following example illustrates use of the hyphen in a named instance resource.

http://api.abc.com/hr/v1/associates/12121212/contacts/emergency-contact

In order to avoid confusion (e.g. due to the possibility of being obscured), the underscore character “_” should not be used in URIs.

**R86** For the URI path, the underscore character “_” **SHOULD** not be used in URIs.

The scheme and host components of the URI are case-insensitive. The other components are case-sensitive; therefore, in order to avoid confusion, lower case letters should be used. [Berners-Lee (2005)]

**R87** For the URI path, lower case letters **SHOULD** be used.

The following example illustrates improper and proper use of upper and lower case letters, respectively.

http://api.abc.com/hr/v1/Associates/12121212/Addresses is not preferred.

http://api.abc.com/hr/v1/associates/12121212/addresses is preferred.

Format preferences (e.g. abc12345.json) should not be communicated in the URI. Instead HTTP’s provided format selection mechanism, the **Accept** request header, must be used.

**R88** File extensions **SHOULD NOT** be used in the URI to indicate format preference.

### 7.2.4 URI Query

The URI query is an optional component of the URI. If provided, it contributes to the unique identification of a resource. The query component comprises a set of parameters that qualifies the resource identified by the path component.
7.2.4.1 Design

The query component supports additional interaction capabilities such as selection (or partial response) and filtering. The detailed description of these capabilities, their usage and rules are available in dedicated sections, below.

R89  The URI query component **MUST** be used to express query criteria on collection resources.

   *Note:* The section, Message Resource Management, defines each type of query criterion.

7.2.4.2 Format

R90  The URI query component **MUST** be indicated by the first question-mark, “?”.

R91  The parameter names occurring in a list in the URI query component **MUST** be delimited by the comma, “,”.

R92  The conjunction of parameter-value pairs **MUST** be represented by the ampersand, “&”.

Note: *The section, Message Resource Management* further specifies the format for each type of query criterion supported.

7.3 URI Encoding

IETF’s RFC 3986 [Berners-Lee (2005)] restricts URI characters to the ASCII character-set. The RFC identifies a set of reserved characters (e.g. “:”, “/”) used to delimit components (and subcomponents) within a URI. Use of either reserved characters outside of their intended purpose or unreserved characters outside of the ASCII character-set must be encoded.

URI encoding or percent-encoding is a mechanism used to represent a data octet in a URI component when that octet’s character is outside the allowed set (i.e. unreserved characters of the ASCII character-set).

**For any request, the following rule applies:**

R93  The URI **MUST** be percent-encoded.

   *Note:* This specification does not encode the URIs provided as examples.

93.1 Reserved characters (of the ASCII character-set) that have a special meaning in a certain context **MUST be percent-encoded.**

   *Note:* See IETF’s 3986 [Berners-Lee (2005)] for the list of reserved characters.

R93.2 Unreserved characters (of the ASCII character-set) **MUST NOT** be percent-encoded.

   *Note:* See IETF’s 3986 [Berners-Lee (2005)] for the list of unreserved characters.

R93.3 Characters that are *not* part of the ASCII character-set **MUST** be percent-encoded.
7.4 URI Template Design and Format

A URI Template is a compact sequence of characters for describing a range of Uniform Resource Identifiers through variable expansion. The specification for URI Template syntax, process for expanding a URI Template into a URI reference and Internet usage guides are defined by Gregorio et al. (2012).

*Note:* The goal of this section is to introduce the URI Template specification by Gregorio et al. (2012); it is limited to an overview of some of the fundamental goals and concepts of the specification. API designers are referred to the URI Template specification for a comprehensive treatment of URI Template design and format. API designers must adhere to this specification in the design of URI Templates.

URI Templates provide a means of representing abstract resource identifiers so that variable parts can be easily identified and described. URI Templates have many uses including: service discovery, configuring resource mappings, defining computing links, specifying interfaces, etc. This specification is specifically interested in their use for the specification of interfaces.

A URI Template may have both literals and expressions. The literals are fixed values that have been determined by the API designer. The variable expressions must receive value substitutions in order to resolve to a resource.

- **Expression** – the text between “{“ and “}”, including the enclosing braces. Each expression contains an optional operator that identifies the expression type and its expansion process, followed by a comma-separated list of variable names and optional value modifiers.

- **Expansion** – the string result obtained from a template expression after processing it according to its expression types, list of variables, and value modifiers (e.g. a prefix, such as max length, that limits a variable’s value string).

<table>
<thead>
<tr>
<th>R94</th>
<th>A URI Template <strong>MAY</strong> contain zero or more expressions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R95</td>
<td>A URI Template’s expression <strong>MUST</strong> be delimited by a matching pair of braces, “{“and “}”.</td>
</tr>
</tbody>
</table>

| R95.1 | Expressions **MUST NOT** be nested. |

Several expression types exist (e.g. simple string expansion; fragment expansion; string expansion with multiple variables; form-style query expansion). The default expression type is the simple string expansion where a single named variable is replaced by its value as a string (after percent-encoding any characters not in the set of unreserved URI characters). The expression type is determined by the first character of the opening brace, for example, a fragment expansion is indicated by the crosshatch, “#”, operator and a form-style query expansion is indicated by the question-mark “?” operator.

The following example illustrates a URI Template with a single expression of type simple string expansion. The URI Template contains one expression: {associateID}.

```
http://api.abc.com/hr/v1/associates/{associateID}
```

The complete set of expression types (and corresponding operators) with examples are provided by Gregorio et al. (2012) in the URI Template specification.
8 Message Resource Management

The definition of a resource management operation\(^6\) includes the client’s request message and the server’s response message(s), as shown in Figure 4. A resource management operation is further classified according to how the resource is managed: CRUD (Create, Read, Update and Delete) Operation, Custom (non-CRUD) Operation.

The request message is used to invoke the operation and communicate how a resource is to be managed, i.e., it conveys the detailed data management instruction for a given resource. This section focuses on the specification for defining request messages\(^7\). Their specification is limited to considering only those HTTP components that are relevant to managing the resource\(^8\); those components include:

- Mandatory identification of the resource being managed. (i.e., a URI in the request start-line)
- Mandatory identification of the operator related to the identified resource. (i.e. an HTTP method in the request start-line or custom operator in URI)
- Conditional message headers (i.e. request message-headers such as Accept, If-Modified-Since)
- Conditional, resource representation. (i.e. request message-body)

Note: All the headers, their purpose and their usage are found above, in the message headers section. Any mention of headers in this section is related to resource management and not intended to be complete and comprehensive.

The remaining subsections provide the specification for request messages supporting:

- CRUD Operations
- Custom Operations (i.e. non-CRUD operations)
- Bulk Operations
- Operations with Large URIs and Query Components with Sensitive Data

---

\(^6\) A resource management operation is an abstract concept, generalizing the different types of operations.

\(^7\) The specification for defining response messages are only provided for Read Operations, necessary to support pagination scenarios with successive request-reply exchanges.

\(^8\) Other HTTP components, such as those needed to address other client-server interaction requirements (e.g. Cache-Control, Transfer-Encoding, and Authorization headers), are addressed separately, in their respective section.
8.1 Query Criteria in the Query Component

Multiple types of query criteria (e.g. selection, filtering) are available to the query component of the URI as part of an HTTP request message. Any such criterion, communicated in a request, is considered part of the request message.

This specification uses a subset of the OData Version 4.0 URL Conventions [OASIS (2014b)] syntax for the representation of different types of query criteria in the URI query component:

- selection criterion
- expansion criterion
- filter criterion
- start sequence criterion
- maximum number criterion
- count criterion
- ordering criterion
- search criterion
- pagination criterion

OData refers to these query criteria as system query options.

The filter criterion may be used in a request message to limit the instance resources being managed to a subset of a collection. The remaining types of criteria are applicable to read

---

9 The rationale for using the OData syntax for the query criterion includes: clear delineation of query criterion boundary and purpose, prevention of name clashes in the URI query that, otherwise, is more prevalent in the generic name-value pair pattern, support of relational and logical operators, and broad capability of the query language provide a backwards compatible path forward for incremental adoption, as needed.
operations only. This section provides the rules that are common across the query criterion types as well as the rules that are specific to the filter criterion.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R96</td>
<td>Filter criterion <strong>MAY</strong> be used in the request message of a Read, Update, Delete or Custom operation.</td>
</tr>
<tr>
<td>R97</td>
<td>Selection, expansion, start sequence, maximum number, count, ordering, search and pagination criteria <strong>MAY</strong> be used in the request message of a Read operation.</td>
</tr>
<tr>
<td>R98</td>
<td>Selection, expansion, start sequence, maximum number, count, ordering, search and pagination criteria <strong>MUST NOT</strong> be used in the request message of a non-Read operation (i.e., Create, Update, Delete or Custom operation).</td>
</tr>
</tbody>
</table>

All OData-defined query criterion parameters are prefixed with the dollar sign "$" character. Any custom query criterion, not defined in the OData specification, must not begin with the "$" character.

**For any query criterion in a message upon a collection or instance resource, the following rules apply:**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R99</td>
<td>The URI query component <strong>MUST</strong> be used to specify the query criteria of the resource representation to be returned in the response.</td>
</tr>
<tr>
<td>R100</td>
<td>Query criterion <strong>MUST</strong> adhere to the OData specification [OASIS (2014b)].</td>
</tr>
<tr>
<td>R100.1</td>
<td>OData query criterion parameters <strong>MUST</strong> be prefixed with the dollar sign, &quot;$&quot;.</td>
</tr>
<tr>
<td>R100.2</td>
<td>Custom query criterion parameters (that are not defined in the OData specification) <strong>MUST NOT</strong> be prefixed with the dollar sign, &quot;$&quot;.</td>
</tr>
</tbody>
</table>

**Note:** The URI Query format rules require that the conjunction of name-value pairs be represented by the ampersand, "&". The query criterion are treated as name-value pairs; therefore, their conjunction is also represented by the ampersand,"&".

In order to specify a nested property of a resource (e.g., a property of a complex property of a resource) as part of a query criterion, the nested property must be qualified by the resource property path to the nested property.

**For any query criteria in a message upon a collection or instance resource, the following rule applies:**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R101</td>
<td>A nested property of a resource <strong>MUST</strong> be qualified by its property path where each segment of the path specifies a property.</td>
</tr>
<tr>
<td>R101.1</td>
<td>Property path segments <strong>MUST</strong> be delimited using the forward slash, &quot;/&quot;, as a delimiter between the containing properties and the nested property.</td>
</tr>
</tbody>
</table>

The following example illustrates a nested property of a person’s given name. In this example, personName is a containing property and givenName is the nested property.

`person/personName/givenName`
8.1.1.1 Specifying Filter Criterion

A client may specify filter criteria in the request message of a resource management operation upon a collection resource in order to constrain the results to a set of instance resources, satisfying the filter criteria. The server uses the filter criteria as search parameters to identify the instance resources in a collection resource. This section describes how filter criteria are to be represented in a RESTful Web API’s request messages.

Note: While the predominant use of filter criteria is in Read operations (i.e., GET messages), specification of filter criteria in other types of request messages is possible, for example, a Delete operation (i.e., DELETE message) that deletes instance resources satisfying a filter criteria.

Filter criterion are expressed in the URI query component with the $filter parameter (an OData system query option). The filter criterion value is expressed as property-value pairs.

For any request message upon a collection resource, the following rule applies:

R102 The $filter parameter MUST be used to specify the filter criterion.

R102.1 The filter criterion value MUST be composed of one or more of the following expressions: comparison expression, logical expression, built-in functional expressions.

R103 A comparison expression MUST adhere to the OData specification [OASIS (2014b)] and be limited to one of the following:

- left-operand “ eq ” right-operand
  - which returns true if the left operand is equal to the right operand, otherwise it returns false
- left-operand “ ne ” right-operand
  - which returns true if the left operand is not to the right operand, otherwise it returns false
- left-operand “ gt ” right-operand
  - which returns true if the left operand is greater than the right operand, otherwise it returns false
- left-operand “ ge ” right-operand
  - which returns true if the left operand is greater than or equal the right operand, otherwise it returns false
- left-operand “ lt ” right-operand
  - which returns true if the left operand is less than the right operand, otherwise it returns false
- left-operand “ le ” right-operand
  - which returns true if the left operand is less than or equal the right operand, otherwise it returns false
A logical expression **MUST** be limited to one of the following:

- **left-operand “ and ” right-operand**
  - which returns true if both the left and right operands evaluate to true, otherwise it returns false

- **left-operand “ or ” right-operand**
  - which returns false if both the left and right operands evaluate to false, otherwise it returns true

- **“not ” operand**
  - which returns true if the operand returns false, otherwise it returns false

A built-in functional expression be limited to one of the following:

- **“contains” (“string”,”string”)”**
  - which returns true if the second parameter value is a substring of the first parameter value, otherwise it returns false.

Precedence of expressions **MUST** be specified with the grouping operator, open and closed parenthesis, “(“ and”).

The following example illustrates a filter that constrains a collection resource to workers in the role of **employee** in the job code of **business analyst**.

```
/hr/v1/workers?$filter=role eq 'employee' and jobCode eq 'business-analyst'
```

The following example illustrates a filter that constrains a collection resource to associates with addresses that have postal code of 11122 or 22233 and status of **active**.

```
/hr/v1/associates?$filter=(address/postalCode eq '11122' or address/postalCode eq '22233') and status eq 'active'
```

The following example illustrates a filter that constrains a collection resource to associates with a family name that contains **smith**.

```
/hr/v1/associates?$filter=contains(familyName, 'smith')
```

### 8.1.1.2 “any/all” Lambda Operators

Cases exist where it is necessary to filter a collection resource based on a given property value for instances in a nested collection resource. There are two possible cases. In the first case, the filter restrict results to **any** instance in the nested collection resource that have a given property value. In the second case, the filter restrict results that have **all** (i.e., only) instances in the nested collection resource with a given property value.

In first case, consider the example of a filter needed to find all work assignments (of a worker) that have **any** work location in a particular city.

```
/hr/v1/workers/12121212/work-assignments?$filter=assigned-work-locations/any(x: x/address/cityName eq 'Charlotte')
```

In the second case, consider the example of a filter needed to find all work assignments (of a worker) that have **all** (only) work locations in a particular city.

```
/hr/v1/workers/12121212/work-assignments?$filter=assigned-work-locations/all(x: x/address/cityName eq 'Charlotte')
```
OData [OASIS (2014a)] defines two lambda operators for this purpose, **any** and **all**, that evaluate a boolean expression on a collection resource. The argument of a lambda operator is a lambda variable name followed by a colon and a boolean expression. The variable name refers to the property of related collection resource.

The **any** operator applies a boolean expression to each instance on a collection resource. The value of true is returned if the expression evaluates to true for **any** instances of the collection resource, otherwise false is returned.

The **all** operator applies a boolean expression to each instance on a collection resource. The value of true is returned if the expression evaluates to true for **all** instances of the collection resource, otherwise false is returned.

The remaining types of query criteria are discussed in the following subsections, in the context of the applicable resource management operation.

### 8.2 Query Criteria in the Path Component

A resource path that identifies a collection may be constructed with a specific resource management criterion that is to be applied against the collection. The criterion is included in the resource path located in path component of the URI as part of an HTTP message. This specification uses a *subset* of the OData Version 4.0 URL Conventions [OASIS (2014b)] syntax for the representation of different types of resource path query criteria in the URI:

- count instruction

Use of the count instruction is described in the context of the Read operation, below.

### 8.3 CRUD Operations

A client’s request message to a server may be part of a CRUD operation. A CRUD operation request message comprises: identification of the resource being managed, one of the CRUD operators (Create, Read, Update, or Delete), and conditionally a resource representation and message headers.

This section specifies the language to be used in the communication of CRUD operations. This section describes:

- The language constructs for defining CRUD operations
- Rules associated with the use of CRUD operations.

RESTful Web APIs leverage the HTTP request methods to define resource management operations.

The following table lists the request methods, description and related CRUD operator.
<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Description</th>
<th>CRUD Operator</th>
<th>Idempotent(^{10})</th>
<th>Safe(^{11})</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>Method used to retrieve metadata about the communications options implemented by the server and applicable to that resource (e.g. the entity header, Allow, which lists the methods supported by the resource at the specified URI at the server).(^{12})</td>
<td>Read</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GET</td>
<td>Method used to retrieve representation of a resource’s state at a specified URI at the server.</td>
<td>Read</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>HEAD</td>
<td>Method used to retrieve only the metadata associated with a resource’s state at a specified URI at the server. The metadata contained in the message headers in response to a HEAD request should be identical to the metadata contained in the message headers in response to a GET request.</td>
<td>Read</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PATCH</td>
<td>Method use to request a modify resource at the specified URI at the server.</td>
<td>Update</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>POST</td>
<td>Method used to create a new resource within a collection at the specified URI at the server.</td>
<td>Create</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PUT</td>
<td>Method used to replace a resource at the specified URI at the server.</td>
<td>Update</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DELETE</td>
<td>Method to delete the resource identified by the specified URI at the server.</td>
<td>Delete</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 6: HTTP Request Methods for Resource Data Management

Recall the RESTful Web API Maturity Model, described above. At Level 0, HTTP is essentially used as a tunneling mechanism. In tunneling, the message’s intent is encapsulated (e.g. in the message-body) and/or misrepresented. At a Level 2 maturity level, tunneling is not

\(^{10}\) Methods have the property of idempotence if the side-effects of N>0 identical requests is the same as for a single request. [Fielding et al (1999)]

\(^{11}\) Methods have the property of safe if there are no side-effects of a request (i.e., Read operations). [Fielding et al (1999)]

\(^{12}\) The description of the OPTIONS method is in the context of a Read operation on a resource. The OPTIONS method may also be used to retrieve a server’s communication options, in general.
permitted; all Create, Read, Update and Delete (CRUD) operators performed on a resource must use the established HTTP methods (e.g. POST, GET) for those operators.

<table>
<thead>
<tr>
<th>R107</th>
<th>The HTTP request methods (e.g. POST) <strong>SHOULD NOT</strong> be used to tunnel other HTTP request methods.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R108</td>
<td>Request messages for CRUD operations <strong>SHOULD</strong> use the established HTTP methods for those operations.</td>
</tr>
</tbody>
</table>

*Note:* CRUD operators (e.g. get) must not be represented in the URI.
Table 7 shows how the HTTP methods are used to represent the CRUD operators (according to the different resource types).

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Usage (by resource type identified in the request URI path)</th>
<th>Collection Resource</th>
<th>Instance Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>Read</td>
<td>Read</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>Read&lt;sup&gt;13&lt;/sup&gt;</td>
<td>Read</td>
<td></td>
</tr>
<tr>
<td>HEAD</td>
<td>Read</td>
<td>Read</td>
<td></td>
</tr>
<tr>
<td>PATCH</td>
<td>Update&lt;sup&gt;14&lt;/sup&gt;</td>
<td>Update&lt;sup&gt;+&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>Create&lt;sup&gt;15&lt;/sup&gt;</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>PUT</td>
<td>Update&lt;sup&gt;16&lt;/sup&gt;</td>
<td>Update&lt;sup&gt;+&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete&lt;sup&gt;17&lt;/sup&gt;</td>
<td>Delete</td>
<td></td>
</tr>
</tbody>
</table>

Table 7: HTTP Request Method Usage for CRUD Operations

The following sections provide the specifications for defining CRUD operations (shown in the table, above) on both Collections and Instance Resources.

---

<sup>13</sup> GET on a Collection Resource reads <em>all</em> instance resources in the collection; it is used in the request message of a bulk operation. A URI query component may serve to limit the instance resources to a subset of the collection.

<sup>14</sup> PATCH on a Collection Resource updates (incrementally) <em>one or more</em> instance resources in the collection; it is used in the request message of a bulk operation. Each instance resource being updated is represented in the entity-body.

<sup>15</sup> POST on a Collection Resource creates <em>one or more</em> instance resources in the collection; it is used in the request message of a bulk operation. Each instance resource being created is represented in the entity-body.

<sup>16</sup> PUT on a Collection Resource update (replace) <em>all</em> instance resources in the collection; it is used in the request message of bulk operation. A URI query component may serve to limit the instance resources being replaced to a subset of the collection.

<sup>17</sup> DELETE on a Collection Resource may delete <em>all</em> instance resources in the collection; it is used in the request message of a bulk operation. A URI query component may serve to limit the instance resources being deleted to a subset of the collection.
8.3.1 Create Operations

A Create operation is a resource management operation used for the creation of an instance resource. This section describes the language elements for the definition and use of request messages of Create operations.

As shown in Table 6, HTTP provides a method for Create operations.

- The POST method is used to create an instance resource(s) within a collection resource.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R109</td>
<td>The HTTP POST method MUST be used to create a new instance resource in a collection resource.</td>
</tr>
</tbody>
</table>

For any POST message, creating an instance resource, the following rule applies:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R110</td>
<td>The URI path component MUST be used to specify the resource collection where the resource is to be created.</td>
</tr>
<tr>
<td>R111</td>
<td>A message-body that contains the representation of the instance resource to be created MUST be included in the request message.</td>
</tr>
</tbody>
</table>

The following example uses the POST request on a collection resource to create an associate in the collection resource of associates.

```
POST /hr/v1/associates HTTP/1.1
Host: api.abc.com
Accept: application/json
Content-Type: application/json

{
    "associates": [ {
        "person": {
            "personNames": [ {
                "typeCode": {
                    "codeValue": "Birth"
                },
                "givenName": "John",
                "middleName": "Steve",
                "familyNames": [ {
                    "nameValue": "Smith",
                    "primaryIndicator": true
                } ]
            } ]
        },
        "birthDate": "1970-02-01"
    } ]
}
```
8.3.2 Update Operations

An Update operation is a resource management operation used for the update of a collection or instance resource. This section describes the language elements for the definition and use of request messages for Update operations.

There are two types of update operations:

- Full update (aka Replace or Snapshot Update)
- Partial update (aka Incremental Update)

In the partial update only the subset of the resource that has been changed or modified is updated in the server’s resource representation; this is in contrast to a full or replacement update, the complete resource that has been changed or modified is updated in the server’s resource representation.

As shown in Table 6, HTTP provides two methods for Update operations.

- For full update, the PUT method is used to replace a resource (e.g. a collection or instance resource).
- For partial update, the PATCH method is used to modify a resource (e.g. a collection or instance resource).

\[\text{R112} \text{The HTTP PUT method MUST be used to replace a representation of an existing collection or instance resource.}\]

\text{Note: The PUT method communicates a replacement (i.e., snapshot or full refresh) of the collection or instance resource. Therefore, a client must send all properties that it manages (even if those properties did not change).}

For any PUT message replacing a resource, the following rule applies:

\[\text{R113} \text{The URI path component MUST be used to specify the identification of the collection or instance resource to be replaced.}\]

\[\text{R114} \text{A message-body that represents the replacement of the collection or instance resource representation MUST be included in the request message.}\]

The following example illustrates a PUT request on an instance resource that communicates a complete replacement of an existing associate representation.

```
PUT /hr/v1/associates/12121212 HTTP/1.1
Host: api.abc.com
Accept: application/json
Content-Type: application/json

{"associates": [{
  "associateID": {
    "idValue": "12121212"
  },
  "person": {
    "personNames": [{
      "typeCode": {
        "codeValue": "Birth"
      }
    }
  }
}
```

Updating a component of a resource (e.g. an Address of an Associate) using the resource URI is discouraged. Instead the component should have its own URI and its own entity-tag, supporting the use of the **If-Match** header for conditional update requests at the component level.

**For any PATCH message modifying a collection or instance resource, the following rule applies:**

**R274** The HTTP PATCH method **MUST** be used to modify a representation of an existing collection or instance resource.

*Note:* The PATCH method communicates a modification (i.e. incremental or delta) of the collection or instance resource.

**R275** The URI path component **MUST** be used to specify the identification of the collection or instance resource to be modified.

**R276** A message-body that represents the modification to the collection or instance resource representation **MUST** be included in the request message.

The following example illustrates a PATCH request on an instance resource that communicates a modification to an existing associate representation.

```
PATCH /hr/v1/associates/12121212 HTTP/1.1
Host: api.abc.com
Accept: application/json
Content-Type: application/json
{
    "associates" : [
        {
            "associateID" : {
                "idValue" : "12121212"
            },
            "person" : {
                "birthDate" : "1970-03-01"
            }
        }
    ]
}
```
Note: The component-level action code may be used in conjunction with the PATCH (e.g. An Associate instance resource, whose Name and Address components to be updated, are included in the entity-body; both components, personName and address, communicate a “Change” (aka update) action code. Only those fields (with the exception of identifier fields) that are to be updated are included in the component (e.g. only givenName and postalCode are updated and included in the entity-body for the personName and address components, respectively. Such use, however, would most likely not be able to leverage the entity-tag to make the update conditional (see the following note).

Note: A conditional PATCH request, that makes use of the entity-tag in the If-Match header (see the section below, Conditional Operations) may have the effect of preventing an allowable update to a resource property (that has remained unchanged while the resource (i.e., entity-body associated with the entity-tag), as a whole, has changed.

8.3.3 Delete Operations

A Delete operation is a resource management operation used for the removal of an instance resource. This section describes the language elements for the definition and use of request messages for Delete operations.

As shown in Table 6, HTTP provides a method for Delete operations.

- The DELETE method is used to remove a resource (e.g. a collection or instance resource).

```
R116 The HTTP DELETE method MUST be used to remove an existing collection or instance resource.
```

For any DELETE message removing resource, the following rule applies:

```
R117 The URI path component MUST be used to specify the identification of the collection or instance resource to be removed.
```

The following example illustrates a DELETE request on an instance resource.

```
DELETE /hr/v1/associates/12121212 HTTP/1.1
Host: api.abc.com
```

Once a DELETE request for an instance resource has been processed, the instance resource is no longer available to clients.

8.3.4 Read Operations

A Read operation is a resource management instruction used for the retrieval or query of instance resources. This section describes the language elements for the definition and use of request messages for Read operations. The definition and use of response messages are also described in support of successive read requests as part of pagination.

As shown in Table 6, HTTP provides three methods for Read operations.

- The GET method is used to retrieve the state of instance resources in a representation.
- The HEAD method is similar to the GET method, except the server will not return a resource representation in the response message-body; instead the server returns only the headers.
- The OPTIONS method is used to retrieve metadata that identifies the methods supported for a resource.
The HTTP GET method **MUST** be used to retrieve a representation of a resource.

*Note*: This includes the following resource types: Instance, Collection.

The following example illustrates a **GET** request on a collection resource that returns a list of associates in the response message-body.

```
GET /hr/v1/associates HTTP/1.1
Host: api.abc.com
```

The HTTP HEAD method **SHOULD** be used to retrieve metadata associated with a resource. The metadata is returned in a response with message headers and no message-body.

The following example illustrates a **HEAD** request on an instance resource that returns a response with message headers without the message-body.

```
The request:

HEAD /hr/v1/associates/12121212 HTTP/1.1
Host: api.abc.com

The response to the request:

HTTP/1.1 200 OK
Content-Type: application/json
```

The HTTP OPTIONS method **SHOULD** be used to retrieve metadata identifying communication options available for a resource.

*Note*: This includes the following resource types: Instance, Collection, Controller.

**R120.1** For an instance or collection resource, the response **MUST** include an **Allow** header that lists the valid HTTP methods for the requested resource.

**R120.2** For a controller resource, the response **MUST** include an **OAGi-Allow- CustomOperator** header that lists the valid custom operators for the requested resource.

The following example illustrates an **OPTIONS** request on an instance resource that returns a response with message headers that includes an Allow header that lists the HTTP methods allowed at the specified instance resource.

```
The request:

OPTIONS /hr/v1/associates/12121212 HTTP/1.1
Host: api.abc.com

The response to the request:

HTTP/1.1 200 OK
Allow: GET, PUT, DELETE
```
For any GET message for a resource, the following rule applies:

R121 A client MUST NOT include a message-body.

8.3.4.1 Specifying Selection Criterion (for a Partial Response)

A client, requesting a resource representation, may specify a subset of a resource’s properties that are to be returned in the response. As the response includes only a part of the resource, it is commonly referred to as a partial response. This section describes how selection criteria are to be represented in a RESTful Web API’s GET messages.

Selection criterion may be applied in a request message of a read operation for a resource (e.g. a collection or instance resource).

Selection criterion is expressed in the URI query component with the $select parameter (an OData system query option). The $select parameter may express both simple properties such as attributes and complex properties such as other object classes. Selection criterion may be used to specify properties that are to be returned in the response.

For any GET message for a resource, the following rule applies:

R122 The $select parameter MUST be used to specify the selection criterion.

R122.1 The selection criterion value MUST specify a list of properties to be returned in the response.

The list of properties in the selection criterion informs the server that those properties are to be included in the resource representation of the response. If a property is not included, this informs the server that the property is not to be included in the response.

R122.1.1 The properties of the selection criterion value MUST be delimited by a comma ",".

R122.1.2 The star "*" operator MUST be used to specify all properties.

The following example illustrates a GET request on an instance resource that specifies the properties to be returned in the response include the person name and address properties.

GET /hr/v1/associates/12121212?$select=personName,address HTTP/1.1
Host: api.abc.com

The following example illustrates a GET request on an instance resource that specifies a list of properties that includes two nested properties, line3 and line4, of the address complex property of the associate.

GET /hr/v1/associates/12121212?$select = personName,address/line3,address/line4 HTTP/1.1
Host: api.abc.com

8.3.4.2 Specifying Expansion Criterion

A client may specify a set of related resources to be included in line with the returned resource representation using an expansion criterion. A resource may have related resources; more specifically, an associating resource (i.e., acting as a source resource) may be related to associated resources (i.e., acting as target resources). By default, related resources are not included in line with the returned resource representation. The default
behavior avoids unnecessarily large representations, improving communication efficiency and performance. This section describes how the expansion criterion are to be represented in a RESTful Web API’s GET messages.

Expansion criteria may be applied in a request message of a read operations for a resource (e.g. a collection or instance resource). This criterion is expressed in the URI query component with the $expand parameter (an OData system query option) and specifies one or more target resources that are related to the source resource, identified in the URI path component.

Note: Use of the expansion criterion requires that an API specification indicate, for a given resource, all related resources that are expandable.

For any GET message for a resource, the following rule applies:

R123 The $expand parameter **MUST** be used to specify the expansion criterion.

<table>
<thead>
<tr>
<th>R123.1</th>
<th>The expansion criterion value <strong>MUST</strong> specify a list of related resources to be returned in the response.</th>
</tr>
</thead>
</table>

The list of related resources in the expansion criteria informs the server that those resources are to be included in the resource representation of the response. If a related resource is not included, this informs the server that the resource is not to be included in the response.

R123.1.1 The related resources of the expansion criteria value **MUST** be delimited by a comma “,”.

<table>
<thead>
<tr>
<th>R123.1.2</th>
<th>The star “*” operator <strong>MUST</strong> be used to specify all related resources.</th>
</tr>
</thead>
</table>

Recall (above) that the selection criterion is used to specify a subset of a resource's properties to be returned in the response; if no selection criterion exists then all the resource's properties are to be returned in the response. When using an expansion criterion, a selection criterion specified upon the resource identified in the URI path component may also include an expanded resource and its properties.

The following example illustrates a GET request on a collection resource, *associates*, that specifies a related resource, *work assignments*, to be returned in the response.

```
GET /hr/v1/associates?$select=personName,workAssignments/positionTitle&expand=workAssignments HTTP/1.1
Host: api.abc.com
```

The following example illustrates a GET request on a collection resource, *associates*, that specifies two related resources, *work assignments* and *employer*, to be returned in the response.

```
GET /hr/v1/associates?$select=personName,workAssignments/positionTitle,employer/name&expand=workAssignments,employer HTTP/1.1
Host: api.abc.com
```
OData [OASIS (2014b)] also allows the use of query criterion within the expansion criterion to further qualify the expansion. The allowed query criteria include: selection criterion ($select), expansion criterion ($expand), filter criterion ($filter), start sequence criterion ($skip), maximum number criterion ($top), count criterion ($count), ordering criterion ($orderby) and search criterion ($search). When applied to an expansion criterion, the query criteria must be a semicolon-delimited list of query criterion, enclosed in parenthesis and appended to the related resource being expanded.

**R123.2** Query criteria **MAY** be appended to the related resource being expanded.

**R123.2.1** The query criteria **MUST** be delimited by a semicolon ";" and enclosed in parenthesis.

Consider an expansion criterion that has been appended with a selection criterion; the selection criterion is specified upon an expanded resource to specify a subset of the expanded resource's properties to be returned in the response. Recall that by default, if no selection criterion exists within the expansion criterion, then all the expanded resource's properties are to be returned in the response.

The following example illustrates a GET request on a collection resource that specifies the related resources to be returned in the response to include the associate’s work assignments' position titles, and work location city.

```
GET /hr/v1/associates?$expand=workAssignments($select=jobCode,jobTitle,positionID,positionTitle,homeWorkLocation/address/cityName) HTTP/1.1
Host: api.abc.com
```

The following example illustrates an expansion criteria that has been appended by two other query criteria. The example further qualifies the previous example by filtering the related resources (i.e. workAssignments) to those workers that are executives.

```
GET /hr/v1/associates?$expand=workAssignments($select=jobCode,jobTitle,positionID,positionTitle,homeWorkLocation/address/cityName;$filter=executiveIndicator eq 'true') HTTP/1.1
Host: api.abc.com
```

### 8.3.4.3 Specifying Instance Resource Start Sequence Criterion

A client may specify the start sequence (or start position) from which instance resources of a collection or instance resource set are be returned in a response. An *instance resource set* (or set of instance resources) is determined by a server to satisfy the set’s membership criteria (i.e., selection, filter, expansion and search criteria) of a resource management request (e.g. GET request) upon a collection resource. The server uses the start sequence criterion to identify the instance resources (of the instance resource set) to be included in the response. This section describes how start criterion are to be represented in a RESTful Web API’s request messages.
Start sequence criterion is expressed in the URI query component with the $skip parameter (an OData system query option). The start sequence criterion value is expressed by a number of instance resources that are to be skipped and not included in the result.

**For any GET message upon a collection resource or instance resource set, the following rule applies:**

**R124** The $skip parameter **MUST** be used to specify the start sequence criterion.

**R124.1** The start sequence criterion value **MUST** specify a non-negative integer, n, for the number of instance resources that are to be skipped and not included in the response.

*Note:* The instance resources returned start at sequence, n+1.

The following example illustrates a GET request on a collection resource that specifies the first 10 associate instance resources should be “skipped” and that the instance resources to be returned start at sequence 11.

```
GET /hr/v1/associates?$skip=10
Host: api.abc.com
```

### 8.3.4.4 Specifying Instance Resource Maximum Number Criterion

A client may specify a limit on the number of instance resources of a collection or instance resource set that are to be returned in a response. An *instance resource set* (or set of instance resources) is determined by a server to satisfy the set’s membership criteria (i.e., selection, filter, expansion and search criteria) of a resource management request (e.g. GET request) upon a collection resource. The server uses the maximum number criterion to limit the number of instance resources (of the instance resource set) to be included in the response. This section describes how this criterion is to be represented in a RESTful Web API’s request messages.

Maximum number criterion is expressed in the URI query component with the $top parameter (an OData system query option). The maximum number criterion value is expressed by a number of instance resources that must not be exceeded in the result.

**For any GET message upon a collection resource or instance resource set, the following rule applies:**

**R125** The $top parameter **MUST** be used to specify the maximum number criterion.

**R125.1** The maximum number criterion value **MUST** specify a non-negative integer, n, that indicates the maximum number of instance resources that may be included in the result.

The following example illustrates a GET request on a collection resource that specifies the maximum number of associate instance resources that may be returned is 50.

```
GET /hr/v1/associates/12121212?$top=50 HTTP/1.1
```
8.3.4.5 Specifying Instance Resource Total Number Criterion

A client may specify that a count of the total number of instance resources in a collection resource or a set of instance resources is to be returned in a response (including the instance resources). An instance resource set (or set of instance resources) is determined by a server to satisfy the set’s membership criteria (i.e., selection, filter, expansion and search criteria) of a resource management request (e.g. GET request) upon a collection resource. The server uses the total number criterion to determine whether it should return a count of the total number of instance resources in the response. This section describes how this instruction is to be represented in a RESTful Web API’s request messages.

Total number criterion is expressed in the URI query component with the $count parameter (an OData system query option). The total number criterion value is expressed with a value of true or false, indicating that server should or should not, respectively, determine and return a count of the instance resources.

For any GET message upon a collection resource or instance resource set, the following rule applies:

R126 The $count name MUST be used to specify the count criterion in the query of the URI query component to return the total number of instance resources, along with the instance resources, of the related collection resource or instance resource set.

R126.1 The total number criterion value MUST be limited to one of the following:

- "$count=true" - which indicates that the service should return a count
- "$count=false" - which indicates that the service should not return a count

The following example illustrates a GET request on a collection resource that specifies that a count of the total number of instance resources in a collection resource be returned. The total number is returned as a value to the totalNumber parameter of an object class called meta; this object class and response parameter are defined in the pagination section, below.

```
GET /hr/v1/associates?count=true HTTP/1.1
Host: api.abc.com
Accept: application/json

The response to the Get request:

HTTP/1.1 200 OK
Content-Type: application/json
{
  "meta": {
    "totalNumber": 25,
    } [resourceRepresentation]
}
```

A client may also specify that a count of the total number of instance resources in a collection resource or instance resource set (specified by additional filter or search
instructions) is to be returned in a response (that does not include the instance resources). The server uses the count criterion to determine whether it count and return the total number of instance resources in the response. This section describes how this instruction is to be represented in a RESTful Web API’s request messages.

The count instruction is expressed by appending $count to the resource path of URI path component.

**For any GET message upon a collection resource or instance resource set, the following rule applies:**

| R252 | The $count name MUST be used to specify the count criterion on a resource path of the URI path component to return the total number of instance resources of the related collection resource or instance resource set. |

The following example illustrates a GET request on a collection resource that specifies that a count of the total number of instance resources in a collection resource be returned. The total number is returned as a value to the totalNumber parameter of an object class called meta; this object class and response parameter are defined in the pagination section, below.

```
GET/hr/v1/associates/$count HTTP/1.1
Host: api.abc.com
Accept: application/json

The response to the Get request:
HTTP/1.1 200 OK
Content-Type: application/json
{
  "meta": {
    "totalNumber": 1025
  }
}
```

### 8.3.4.6 Specifying Order Criterion

A client may specify the order in which instance resources of a collection or instance resource set are to be returned in a response. An instance resource set (or set of instance resources) is determined by a server to satisfy the set’s membership criteria (i.e., selection, filter, expansion and search criteria) of a resource management request (e.g. GET request) upon a collection resource. The server uses the order criterion to sort the instance resources (of the instance resource set) to be returned in the response. This section describes how this criterion is to be represented in a RESTful Web API’s request messages.

Order criterion are expressed in the URI query component with the $orderby parameter (an OData system query option). The order criterion value is expressed by a comma-separated list of expressions that are used to sort the instance resources.

**For any GET message upon a collection resource or instance resource set, the following rule applies:**

| R127 | The $orderby parameter MUST be used to specify the order criterion. |
R127.1 The order criterion value **MUST** be limited to the following:

```plaintext
"$orderby="property-name [" asc"] [" desc"]
```

R127.1.1 The **asc** suffix **MUST** be used to specify an ascending order.

R127.1.2 The **desc** suffix **MUST** be used to specify a descending order.

R127.1.3 If the **asc** or **desc** suffix is not specified, then the server **MUST** order by the property-name in ascending order.

The following example illustrates a GET request on a collection resource that specifies an order for associate instance resources in which they are to be returned.

```
GET /hr/v1/associates?$orderby=personName/familyName asc HTTP/1.1
Host: api.abc.com
```

**8.3.4.7 Specifying Search Criterion**

A client may specify search criterion in a request message of a read operation upon a collection resource or instance resource set to constrain the results to a set of instance resources, satisfying the search criteria. The server uses the search criteria as free-text search parameters to identify the instance resources of the result. This section describes how search criteria are to be represented in a RESTful Web API’s request messages.

Search criterion are expressed in the URI query component with the `$search` parameter (an OData system query option). The seach criterion value is expressed by a comma-separated list of expressions that are used to sort the instance resources.

**For any request message upon a collection resource or instance resource set, the following rule applies:**

R128 The `$search` parameter **MUST** be used to specify the search criterion.

R128.1 The search criterion value **MUST** be limited to the following:

```plaintext
"$search= search-expression

search-expression = search-term | "("search-expression ")" | ["NOT"] search-expression [ [ "AND" | "OR"] search-expression ]
- where each search-term returns true if the search-term is matched, otherwise it returns false
- where a NOT search-expression returns true if the expression is not matched, otherwise it returns false
- where search-expressions separated by an OR return true if either of the expression evaluates to true, otherwise it returns false
- where search-expressions separated by an AND if both of the expressions evaluate to true, otherwise it returns false
```
The search-term **MUST** be used to specify a single word or phrase. A phrase **MUST** be enclosed in double-quotes, " ".

A group expression **MUST** be specified with the grouping operator, open and closed parenthesis, "( " and ")".

The following example illustrates a GET request on a collection resource that returns a list of associates that match the search terms of *smith* or *jones*.

```
GET /hr/v1/associates?$search=smith OR jones HTTP/1.1
Host: api.abc.com
```

### 8.3.4.8 Specifying Pagination Criteria

This section discusses the technique for handling multiple instance resources resulting from an initial Read request upon a collection resource. This technique is also referred to as pagination.

The technique is necessary when the read results cannot be either returned (by the responding system) or consumed (by the requesting system) in a single response message instance. This is often the case when either the requesting or responding systems have message size performance measures whose thresholds cannot be exceeded in order to maintain adequate system performance.

The technique leverages a set of pagination-related parameters that are applicable to either the GET message or the GET response message, GET message page-parameters and GET response page-parameters, respectively. The GET message page-parameters leverage the OData types of Query Criteria parameters where applicable. The set of the parameters communicated in a Get request is referred to the pagination criteria. The parameters rely on a concept called the *instance resource set*. An *instance resource set*, is determined by a server to satisfy the set’s membership criteria (i.e., selection, filter, expansion and search criteria) of a resource management request (e.g. GET request) upon a collection resource.

A client may or may not require read consistency as it paginates through the instance resource set. Requirements for read consistency are specific to the use scenario of a particular interface. An interface’s specification must indicate whether or not read operations support pagination and read consistency.

Two common approaches for read consistency include:

- **The client indicates to the server that read consistency is required.** The server may save the instance resource set in a cache. The implementation might choose to cache a key set or the full instance resource.
- **The client indicates to the server that read consistency is required.** The server may use reflection to alter the instance resources of the page to be returned to the client. The server may keep track of *new* and *missing* instance resources (e.g. via timestamps) and only includes instance resources in the page that existed at the time the initial request was made.
- **In the case that the client does not require read consistency,**
- **The client indicates to the server that read consistency is not required.** The server does not need to save the instance resource set. The client accepts the possibility that inconsistency may exist when paginating through the instance resource set.
The pagination technique used in this specification, aligns with the first approach (above) and allows clients to specify whether or not read consistency is required by indicating to the server that the instance resource set should be saved or not saved, respectively.

**GET request message page-parameters:**

- **$skip** - Indicates the instance resources that are to be skipped and not included in the response. This attribute is specified on subsequent Get requests, not the initial GET request. The client may determine this sequence from the prior GET response (see the Get response message parameters, below, for more information).
- **$top** - Communicates the maximum number of instance resources that should be returned in a response.
- **uniqueIndicator** - Indicates whether duplicates should be filtered out.
- **resourceSetSaveIndicator** – Indicates whether the server should save the instance resource set; a saved instance resource set supports read consistency requirements while paginating through the instance resource set.
- **resourceSetID** - Unique identifier of the instance resource set. It is generated by the server as a result of the original Get request.

**GET response page-parameters:**

- **startSequenceNumber** - The instance resource sequence identifying the first resource returned in the response. The server generates this sequence. It is used by the client to determine the start sequence of the subsequent Get request.

---

18 This document differentiates, as needed, initial GET requests from subsequent ones. Subsequent GET request(s) may be communicated when the initial GET request results in more records than can be returned in a single response.
• **completeIndicator** - Indicates whether the response completes the return of all the resources of the instance resource set to the requesting system.
• **returnedNumber** - Number of instance resources in the response.
• **totalNumber** - Number of total instance resources in an instance resource set.
• **resourceSetID** - Unique identifier of the instance resource set. It is generated by the server as a result of the original Get request.

For any GET message using pagination, the following rules apply:

**R129** The URI query component **MUST** be used to specify the pagination criteria (i.e., set of GET message page-parameters) used to request the number of instance resources to be included in the response.

**R129.1** The pagination criteria **MUST** be specified as parameter-value pair(s).

*Note: The GET message page-parameters represent the parameters available.*

**R129.1.1** A conjunction of parameter-value pairs **MUST** be specified by an ampersand “&”. 

---

**Figure 6: Get Response Message**
R130 The client **MAY** assign a value to the parameter:
- `$top`
  to specify the maximum number of instance resources to be returned in a response.

*Note:* The server may have a message size performance measure with respect to the message production. Therefore, the number of instance resources in the response should always correspond to the more restrictive performance measure among the requesting and responding systems/components. In other words, the resource count in the response should equal the lesser of the client’s maximum number (of instance resources) and the server’s maximum number (of resources).

R130.1 If the client has not assigned a value to the `$top` parameter, then the server **MUST** use a default value, deemed appropriate, for the given interface operation as offered by the server (service provider).

*Note:* If the server does not maintain a maximum number default for the specific interface read operations, then the server may assign and use a general default value across interface read operations, for example\(^*_{19}\):
- `$top=10$

R131 The client **MAY** assign a value to the parameter:
- `uniqueIndicator`

R132 The server **MUST** assign a value to the parameters:
- `startSequenceNumber`
- `completeIndicator`
- `returnedNumber`

R133 The server **MAY** assign a value to the parameter:
- `totalNumber\(^*_{20}\)`

R133.1 If the server determines that it cannot not provide a value assignment to the `totalNumber` parameter, then the server **MUST** provide a **null** assignment.

R133.2 If the server determines that no results satisfy the GET request, then the server **MUST** return the following assignment:
- `totalNumber = 0`

---

\(^*_{19}\) This value may be adjusted and tends to have an inverse relationship with the size of the resource representation.

\(^*_{20}\) Value assignment to the `totalNumber` parameter requires that the system determine the total number of instance resources satisfying the GET request. This requires additional logic to be executed in addition to the read operation that may result in performance issues.
For any initial GET request using pagination, the following rules apply:

R134 The client MAY assign a value to the parameter:
- resourceSetSaveIndicator
to specify whether or not the server is required to save the instance resource set.

R134.1 If the client requires read consistency, then the client MUST assign the parameter, resourceSetSaveIndicator = “true”.

R134.1.1 If the client has assigned the parameter, resourceSetSaveIndicator = “true”, then the server MUST assign and return a value to the resourceSetID parameter in the response.

R134.2 If the client does not require read consistency, then the client MUST assign the parameter, resourceSetSaveIndicator = “false”.

R134.3 If the client has not specified a value assignment for the resourceSetSaveIndicator, then the server MUST default the value to “false”.

R135 The client MUST NOT assign a value to the parameter:
- resourceSetID

R253 The client MAY assign a value to the parameter:
- $skip

For any subsequent GET request using pagination, the following rules apply:

R136 The client MUST assign a value to the parameter:
- $skip
to identify the number of the instance resource from the resource set that are to be skipped (i.e. not included) in the response.

Note: The $skip must be calculated using the following equation: GetRequest_{i+1}.$skip = GetRequestResponse_{i}.number + GetRequest_{i}.$skip, where i represents a Get request and Get request response pair. The number of instance resources returned is always limited by the value of the $top parameter specified by the client in the Get request. This parameter is set per the message size performance measure of the client with respect to message consumption.

R136.1 The $skip parameter SHOULD be initialized at “1”.

R137 The client MUST NOT include the parameter:
- resourceSetSaveIndicator

R138 If the server has returned a value to the resourceSetID parameter in the response, then the client MUST use this value in the resourceSetID assignment of the request.
R139 The server **MUST** return the **GET response page-parameters** in an object class, named **paginationResponse**, in the message-body of the response to a **GET** request.

Recall that a client may or may not require read consistency as it paginates through the instance resource set. In the first case, the client indicates to the server that read consistency is required. In the second case, the client indicates to the server that read consistency is not required. The following two sections describe these two cases and provide examples.

### 8.3.4.8.1 Client Requires Pagination Read Consistency

The client specifies in the initial GET request that pagination read consistency is not required (by having assigned the **resourceSetSaveIndicator** parameter to “false”). In this case, the server **may or may not** save an instance resource set.

**Note:** Although it is not necessary to save an instance resource set from the client’s perspective, the server may still elect to save an instance resource set.

The server may uniquely identify the instance resource set (i.e., **resourceSetID** parameter) and return its identifier in the GET request response along with additional information on the instance resources, such as the number of instance resources (i.e., **returnedNumber** parameter) being returned. If an instance resource set identifier (i.e., **resourceSetID** parameter) was provided, then it must be specified on any subsequent Get requests where additional instance resources of the set are requested.

**An Example**

A client sends a GET request for all associates (in an organization), where no more than 10 associates are to be returned in a single response. The client does not require read consistency and informs the server that it does not need to save the instance resource set. Subsequent GET requests are issued for additional associates (beyond those included in the initial response).

**The initial Get request:**

```
GET /hr/v1/associates?$top=10 HTTP/1.1
Host: api.abc.com
Accept: application/json
```

The server processes the GET request, constructs, and executes a query that returns the first 10 associates for the organization. The server sends the 10 associates in the response message instance.

**The response to the initial Get request:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{  
  "paginationResponse": {  
    "startSequenceNumber": 1,  
    "returnedNumber": 10,  
    "totalNumber": 25,  
    "completeIndicator": false  
  }  
}  
```

The client, having received the response, then requests the next 10 associates. It sends the following GET request:
GET /hr/v1/associates?$top=10&$skip=10 HTTP/1.1
Host: api.abc.com
Accept: application/json

The server processes the GET request returns the following response:

HTTP/1.1 200 OK
Content-Type: application/json

{
  "paginationResponse": {
    "startSequenceNumber": 11,
    "returnedNumber": 10
    "totalNumber": 25
    "completeIndicator": false
  }
  [resourceRepresentation]
}

The client, having received the response, makes a final request for the remaining associates. It sends the following GET request:

GET /hr/v1/associates?$top=10&$skip=20 HTTP/1.1
Host: api.abc.com
Accept: application/json

The server processes the GET request returns the following response:

HTTP/1.1 200 OK
Content-Type: application/json

{
  "paginationResponse": {
    "startSequenceNumber": 21,
    "returnedNumber": 5
    "totalNumber": 25
    "completeIndicator": true
  }
  [resourceRepresentation]
}

A client sends a GET request for all associates (in an organization), where no more than 10 associates are to be returned in a single response. The client requires read consistency and requests that the server save the instance resource set. Subsequent GET requests are issued for additional associates (beyond those included in the initial response).

The initial Get request:

GET /hr/v1/associates?$top=10&resourceSetSaveIndicator=true HTTP/1.1
Host: api.abc.com
Accept: application/json

The server processes the GET request, constructs, and executes a query that returns the first 10 associates for the organization. The server sends the 10 associates in the response message instance.

The response to the initial Get request:

HTTP/1.1 200 OK
Content-Type: application/json

{
  "paginationResponse": {
    "startSequenceNumber": 1,
    "returnedNumber": 10
    "totalNumber": 25
    "completeIndicator": false
  }
  [resourceRepresentation]
}
The client, having received the response, then requests the next 10 associates. It sends the following GET request:

GET /hr/v1/associates? $top=10 & $skip=10 & resourceSetID=7001 HTTP/1.1
Host: api.abc.com
Accept: application/json

The server processes the GET request returns the following response:

HTTP/1.1 200 OK
Content-Type: application/json

{
  "paginationResponse": {
    "startSequenceNumber": 11,
    "returnedNumber": 10
    "totalNumber": 25
    "completeIndicator": false
    "resourceSetID": "7001"
  }
  [resourceRepresentation]
}

The client, having received the response, makes a final request for the remaining associates. It sends the following GET request:

GET /hr/v1/associates? $top=10 & $skip=20 & resourceSetID=7001 HTTP/1.1
Host: api.abc.com
Accept: application/json

The server processes the GET request returns the following response:

HTTP/1.1 200 OK
Content-Type: application/json

{
  "paginationResponse": {
    "startSequenceNumber": "21",
    "returnedNumber": "5",
    "totalNumber": "25",
    "completeIndicator": "true",
    "resourceSetID": "7001"
  }
  [resourceRepresentation]
}

When leveraging this approach, the instance resource set timeout settings should be maintained by the responding system. Once threshold for an instance resource timeout has been met the responding system may recover the system resources that were used to manage that instance resource set. Timeout settings should be agreed to between trading partners as part of the service level agreement of the interface contract.
8.3.4.8.2 Client Does Not Require Pagination Read Consistency

The client specifies in the initial GET request that pagination read consistency is not required (by having assigned the `resourceSetSaveIndicator` parameter to “false”). In this case, the server may or may not save an instance resource set.

Note: Although it is not necessary to save an instance resource set from the client’s perspective, the server may still elect to save an instance resource set.

The server may uniquely identify the instance resource set (i.e., `resourceSetID` parameter) and return its identifier in the GET request response along with additional information on the instance resources, such as the number of instance resources (i.e., `returnedNumber` parameter) being returned. If an instance resource set identifier (i.e., `resourceSetID` parameter) was provided, then it must be specified on any subsequent Get requests where additional instance resources of the set are requested.

If the server has elected to not save the instance resource set, then the server must re-execute the initial GET request (i.e., the query) upon any subsequent Get requests where additional instance resources of the set are requested.

An Example

---

A client sends a GET request for all associates (in an organization), where no more than 10 associates are to be returned in a single response. The client does not require read consistency and informs the server that it does not need to save the instance resource set. Subsequent GET requests are issued for additional associates (beyond those included in the initial response).

The initial Get request:

```
GET /hr/v1/associates?$top=10 HTTP/1.1
Host: api.abc.com
Accept: application/json
```

The server processes the GET request, constructs, and executes a query that returns the first 10 associates for the organization. The server sends the 10 associates in the response message instance.

The response to the initial Get request:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "paginationResponse": {
    "startSequenceNumber": 1,
    "returnedNumber": 10,
    "totalNumber": 25,
    "completeIndicator": false
  }
  {resourceRepresentation}
}
```

The client, having received the response, then requests the next 10 associates. It sends the following GET request:

```
GET /hr/v1/associates?$top=10&$skip=10 HTTP/1.1
Host: api.abc.com
Accept: application/json
```

The server processes the GET request returns the following response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
```
"paginationResponse": {  
  "startSequenceNumber" : 11,  
  "returnedNumber" : 10  
  "totalNumber": 25  
  "completeIndicator" : false  
}  
}  

The client, having received the response, makes a final request for the remaining associates. It sends the following GET request:

GET /hr/v1/associates? $top =10&$skip=20 HTTP/1.1  
Host: api.abc.com  
Accept: application/json

The server processes the GET request returns the following response:

HTTP/1.1 200 OK  
Content-Type: application/json  
{
  "paginationResponse": {  
    "startSequenceNumber" = "21"  
    "returnedNumber" = "5"  
    "totalNumber" = "25"  
    "completeIndicator" = "true"  
  }  
}  

8.3.4.9 Specifying View Criterion

A client may specify certain views of a given resource. The selection criterion is used by a client to specify the subset of a resource's properties to be returned in the response. A view, on the other hand, is a pre-defined subset of a resource's properties to be returned in the response.

For any GET request for a resource, the following rules apply:

R283 The view parameter MUST be used to specify the view criterion.

R283.1 The view criterion value MUST specify a view to be returned in the response.

R283.1.1 The view criterion value MUST be limited to an element of the value domain:
- "minimal",
- "standard",
- "extended".

R283.1.1.1 The minimal value MUST be used to indicate the minimal or summary view of the resource.

R283.1.1.2 The standard value MUST be used to indicate the standard or most common view of the resource.
R283.1.1.3 The extended value MUST be used to indicate the extended or more detailed view of the resource.

GET /hrv1/associates?view=minimal HTTP/1.1
Host: api.adp.org

8.3.5 Conditional Operations

R115 The If-Unmodified-Since and If-Match headers MAY be used in request messages of Read, Update and Delete operations to make a request conditional, based on the entity/resource representation having not changed.

Note: This header is used for optimistic concurrency control. With optimistic concurrency control, a given request is performed only if the resource representation was not modified since the datetime (specified in the If-Unmodified-Since header) and/or if the resource representation entity-tag matches the entity-tag (included in the If-Match header).

R251 The If-Modified-Since and If-None-Match headers MAY be used in request messages of Read, Update and Delete operations to make a request conditional, based on the entity/resource representation having changed.

8.3.6 A Note on Nulls

Null is defined as missing or unknown. A data element’s value may be assigned to Null if the value of the data element is missing or unknown. This is referred to as a Null Assignment.

The communication of null assignments for data elements in a resource representation is limited to partial update requests.

Recall that there are two types of update operations:

- Full update\(^{21}\) (aka Replace or Snapshot Update)
- Partial update\(^{22}\) (aka Incremental Update)

\(^{21}\) In the full update, the complete resource that has been changed or modified is updated in the server’s resource representation.

\(^{22}\) In the partial update only, the part of the resource that has been changed or modified is updated in the server’s resource representation.
R140 A resource representation related to a create, read, and full update request or its results (i.e. POST request, GET response, and PUT request) MAY communicate data elements with null assignments.

Note: If data elements with null assignments are excluded from the resource representation in a message instance, then the server must infer which data elements have null assignments. This requires that the server understand the relevance of data elements comprising a message in context of a Service Provider. For example, consider the data element, personName.middleName, of a person resource that is defined as part of a message; the data element is not relevant to Service Provider-XYZ (as the Service Provider does not manage the data element). By knowing the data elements is relevant/not relevant to a message, the server may infer null assignments for data elements that are excluded, but relevant, in a message instance. Alternatively, if data elements with null assignments are included in the resource representation in a message instance, then the server need not infer which data elements have null assignments.

R277 A resource representation of a partial update request, using the HTTP PATCH method, MUST communicate those data elements, whose value assignments have been deleted, with null assignments.

Note: To remove the value assignment of a data element in a resource, the resource representation must communicate that data element with a null assignment.

8.4 Custom Operations

A client’s request message to a server may be part of a custom operation. A custom operation (non-CRUD operation) includes: identification of the collection or instance resource being managed, a custom operator to be invoked on the resource and, conditionally, a resource representation and message headers.

Request messages of custom operations are used to invoke application-specific operators that are not supported by one of the standard HTTP methods (representing create, read, update and delete operators). The operators are specific or customized to the type of resource; therefore, they are referred to as custom operators.

This section specifies the language to be used in the communication of custom operations in client requests.

This section describes:
- The language constructs for defining custom operations
- Rules associated with the use of custom operations

RESTful Web APIs leverage the HTTP POST method to define custom operations. Table 1 describes the POST method. HTTP defines the POST method as neither idempotent nor safe and states that “the actual function performed by the POST method is determined by the server and is usually dependent on the request URI”. [Fielding et al (1999)] Therefore, the open-ended POST method is used to convey the equally open-ended application-specific, non-CRUD, custom operators.
### Table 8: HTTP Request Method for Custom Operations

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Description</th>
<th>Operator</th>
<th>Idempotent</th>
<th>Safe</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>Method used to execute a custom operator at the specified URI at the server.</td>
<td>Non-CRUD operator</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**R142** Custom operators **MUST NOT** be used for CRUD operations.

**R143** The HTTP POST method **MUST** be used to invoke a custom operator on a resource.

A custom operator is expressed in the last path segment of the URI path component. This path segment follows the identification of the collection or instance resource on which the operator applies.

**For any POST message, of a (non-CRUD) custom operation, the following rule applies:**

**R144** A path segment of the URI path component **MUST** be used to specify the identification of the operator to be invoked on the identified resource.

**R145** If the request message of a custom operation requires a resource representation, then a message-body that contains the resource representation **MUST** be included in the request message.

**Note:** The URI, comprising the identification of a collection or instance resource in addition to a custom operator, identifies a **controller resource**.

**For any POST request, communicating a non-CRUD operator, the following rules apply:**

**R146** A client **MAY** include message headers.

**R147** A client **MAY** include a message-body.

The following example illustrates a request message of a custom operation with the operator, hire, on the associate instance resource, 12121212.

```plaintext
POST /hr/v1/associates/12121212/hire HTTP/1.1
HOST: api.abc.com
```
The following example illustrates a request message of a custom operation with the operator, *approve*, on the *timeOffRequests* collection resource.

```
POST /time/v1/timeOffRequests/approve HTTP/1.1
HOST: api.abc.com
```

### 8.5 Bulk Operations

Uses cases exist that require bulk management of similar instance resources (i.e., the approval of multiple time-off requests) in order to promote greater efficiency in the user experience and in the message exchange between client and server. This is made possible with bulk operations.

A request message of a bulk operation comprises exactly one operator (i.e., CRUD operator or Custom operator) to be performed on multiple and similar instance resources.

Table 7, above, indicates those HTTP methods, when managing collection resources, used in request messages of bulk operations (acting upon multiple instance resources).

### 8.6 A Pattern for Large URIs and Query Components with Sensitive Data

Two patterns are presented in this section to address two common problems associated with the URI query component:

- Cases where the URI may become large and exceed URI size limitations,
- For example, request messages of bulk operations that specify multiple instance resources;
- Cases where the URI query component communicates sensitive data (e.g. personal identifiable information) as filter criteria.
- For example, a GET request for an associate that filters on a tax identifier.

Data that is considered to be sensitive per applicable security policies (e.g., personally identifiable information (PII)) must not be exposed in an insecure manner. Data that is represented in the query component of a request’s URI is not secure. Although the SSL protocol encrypts the query component string, securing the data in transit, overall security of the data is problematic:

- URIs can be stored in clear text in server logs
- URIs may be stored in clear text in user-agent (e.g. web browser) logs
- URIs are stored in clear text in Referer headers
- URIs can be bookmarked

As a result, sensitive data must not be communicated in the URI query component of that request’s URI.
The first pattern, below, leverages creates and saves an instance resource set and requires two pairs of request-reply messages. [Allamaraju et al. (2010)]

1. The Service Consumer’s first request specifies a set of instance resources (from a collection) for the creation of a resource set. It is a POST request to a URI that identifies a custom operator, save-resource-set, on the collection being managed. The entity-body contains the URI (i.e. a large URI and/or query components with sensitive data) that specifies the instance resources of the collection that are to be managed. The entity-body must be url encoded.

2. For a successful request, the Service Provider must return a 201 Created response status code and a URI in the Location header that provides an identifier for the set.

Figure 7: Systems Interaction for URIs with Voluminous and Sensitive Data - Save and Query an Instance Resource Set (Pattern 1)
of instance resources of the collection to be managed, resourceSetID. An entity-tag may be provided in the ETag header by the server to identify the version of the set of instance resources. For unsuccessful requests, see the Confirmation Management section for a list of possible error status codes.

3. The Service Consumer’s second request communicates the resource management request on the resource set. It uses the URI, provided in the Location header of the previous response that identifies the instance resource set. The entity-tag may be returned in the If-Match header to make the request conditional.

4. For a successful request, the Service Provider returns the 200 OK response status code and, if applicable, a resource representation (for the operation’s output data) or a Confirm Message.

Note: This pattern is limited to Read, Update, Delete and Custom operations; Create operations do not leverage the URI query component. The sequence diagram, below, illustrates this interaction pattern.

For any request message, with a URI that has a large query component or includes sensitive data, upon a collection resource (per pattern 1: Save and Query the Instance Resource Set), the following rules apply:

R148 First, the client MUST send a POST request that identifies a custom operator, save-resource-set, on the collection resource to create a resource set (i.e., a set of instance resources).

R148.1 The URI of the request message (i.e. with a large URI and/or query components with sensitive data) MUST be represented in the entity-body of the POST request as url-encoded content type.

R148.1.1 The Content-Type header field value MUST be assigned: “application/x-www-form-urlencoded”.

Note: The application/x-www-form-urlencoded content type (or media type) is described in the HTML 4.01 Specification [Raggett (1999)]

R148.2 The server, in the case of success, MUST return a 201 Created status with a resourceSetID in the Location header of the response.

Note: The resourceSetID should be transient. Services leveraging this pattern will need to determine the transient characteristics of their resourceSetIDs and manage them accordingly.

R148.2.1 Persistence and availability (i.e. time interval) of the URI, specified in the Location header, MUST be defined in the API Specification.

R148.3 The server, in the case of success, MAY return an entity-tag in the ETag header to identify the version of the set of instance resources.
Second, the client **MUST** send the request message using the URI returned in the **Location** header (in response to the first request) that includes the `resourceSetID` which identifies the resource set being managed in the request.

The client **MAY** include the entity-tag URI returned in the **ETag** header (in response to the first request) in the If-Match header to make the request conditional.

The second pattern, below, simplifies the first pattern for read operations by **not** requiring the Service Provider to save an instance resource set, thereby limiting the interactions to a single pair of request-reply messages. [Allamaraju et al. (2010)]

1. The Service Consumer’s first request specifies a set of instance resources (from a collection) for the read of a resource set. It is a **POST** request to a URI that identifies a custom operator, `read-resource-set`, on the collection being managed. The **entity-body** contains the URI (i.e. a large URI and/or query components with sensitive data) that specifies the instance resources of the collection that are to be managed. The **entity-body** must be url encoded.

2. For a successful request, the Service Provider must return a **200 OK** status code and the representation of the set of instance resources specified in the request. An **entity-tag** may be provided in the **ETag** header by the server to identify the version of the set of instance resources. The response includes the meta data associated with the GET response message (e.g., the totalNumber of instance resources).
3. If the number of instance resources is too large (i.e., greater than the maximum number of instance resources that either the client or the server can accommodate), the request will result in a **303 See Other** response including a Location header that provides the URI for a created and saved resource set. The Service Consumer may then submit GET requests to the URI to retrieve the instance resource representations using the pagination pattern. Note that the 303 See Other response will not include any of the instance resource representations.

For any request message, with a URI that has a large query component or includes sensitive data, upon a collection resource (per pattern 2: Read the Instance Resource Set), the following rules apply:

R273 First, the client **MUST** send a **POST** request that identifies a custom operator, **read-resource-set**, on the collection resource to read a resource set (i.e., a set of instance resources).

R273.1 The URI of the request message (i.e. with a large URI and/or query components with sensitive data) **MUST** be represented in the **entity-body** of the POST request as url-encoded content type.

R273.1.1 The **Content-Type** header field value **MUST** be assigned: “application/x-www-form-urlencoded”.

*Note:* The **application/x-www-form-urlencoded** content type (or media type) is described in the HTML 4.01 Specification [Raggett (1999)]

R273.2 The server, in the case of success, **MUST** return a **200 OK** status and, if applicable, the representation of the set of instance resources specified in the request.

R273.3 The server, in the case of success, **MAY** return an **entity-tag** in the **ETag** header to identify the version of the set of instance resources.

Factors for consideration in the use of the second pattern are provided below. [Allamaraju et al. (2010)]

Advantages:
- A response can be obtained for the results of a read request in a single request-response interaction.
- Atomicity of a bulk request is supported.

Disadvantages:
- Using POST for read operations weakens the HTTP's uniform interface since GET is defined for safe and idempotent read operations.
- The results are not cacheable causing the server to respond for the same request; this introduces extra latency for the client and reduced scalability for the server.
9 Hypermedia Controls

A resource's current state representation may include hypermedia controls (i.e. links) representing actions and associations that are available on a resource in a given state. Both actions and associations are described with link relations. This specification leverages the JSON Schema [Zyp et al. (2013b)] link description object class to describe link relations.

R150 The **Link** object class **MUST** be used to represent hypermedia controls in a message.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>href</td>
<td>The value of the href link description property is a template used to determine the target URI of the related resource. The value should be resolved as a URI.</td>
</tr>
<tr>
<td>rel</td>
<td>The value of the &quot;rel&quot; property indicates the name of the relation to the target resource.</td>
</tr>
<tr>
<td>title</td>
<td>A title for the link. The value must be a string. User agents MAY use this title when presenting the link to the user.</td>
</tr>
<tr>
<td>targetSchema</td>
<td>Schema that defines the expected structure of the resource representation (e.g. JSON representation) of the target of the link (of the response), if the target of the link is returned with a representation.</td>
</tr>
<tr>
<td>mediaType</td>
<td>The media type of the link target.</td>
</tr>
<tr>
<td>method</td>
<td>Method for requesting the target of the link.</td>
</tr>
<tr>
<td>encType</td>
<td>The media type in which to submit data along with the request.</td>
</tr>
</tbody>
</table>

![Link Description Model](image)

Figure 9: Link Description Model

In the context of a message schema, the link description object is used to define the link relations of the message instances. [Zyp et al. (2013b)] The properties of the **Link** object class are defined, below:

- **href** - The value of the href link description property is a template used to determine the target URI of the related resource. The value should be resolved as a URI.
- **rel** - The value of the "rel" property indicates the name of the relation to the target resource.
- **title** - A title for the link. The value must be a string. User agents MAY use this title when presenting the link to the user.
- **targetSchema** - Schema that defines the expected structure of the resource representation (e.g. JSON representation) of the target of the link (of the response), if the target of the link is returned with a representation.
- **mediaType** - The media type of the link target.
- **method** - Method for requesting the target of the link.
- **encType** - The media type in which to submit data along with the request.
schema - Schema describing the data to submit along with the request; the schema defines the acceptable structure of the submitted request. For example, for a GET request, this schema would define the properties for the query string and for a POST request, this would define the body.

For any Link instance, the following rules apply:

R151 The properties MUST adhere to the definitions and multiplicity constraints documented in the Link Description model.

R152 The href value MUST be used to specify a URI template and adhere to the format and value domains as specified in IETF’s RFC 6570 [Gregorio et al. (2012)].

R52.1 The href value SHOULD be resolved as a URI reference as specified in IETF’s RFC 3986 [Berners-Lee (2005)].

Note: Although relative URIs are supported in RFC 3986, this specification requires URIs to be absolute.

R153 The targetSchema value is advisory only; it MAY be used by a client to validate the returned representation, but it MUST NOT be used by a client to aid in the interpretation of the data received in response to following the link. [Zyp et al. (2013b)]

Note: The interpretation of data risks re-interpreting “safe” data.

R154 The rel value MUST be limited to an element of the value domain:

“alternate”
“create”,
“describedby”,
“edit-form”,
“enclosure”,
“full”,
“related”,
“root”,
“self”,
“up”,
“first”,
“next”,
“previous”,
“last”
“canonical”,
“search”,
“/oagi/invoke”,
“/oagi/confirm-message”,
“/oagi/codelist”,
“/oagi/externalLink”,
“/oagi/callback”,
“/oagi/processing-status”,
“/oagi/request-result”.

R154.10 The alternate value MUST be used to indicate that the link target identifies an alternate representation of the current representation.

*Note:* See the IANA Registry of Link Relations [IANA (2013a)]. The alternate representation is in the format as specified by the mediaType.

<table>
<thead>
<tr>
<th>R154.1</th>
<th>The create value MUST be used to indicate a target to use for creating new instances of a schema. [Zyp et al. (2013b)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>R154.2</td>
<td>The describedby value MUST be used to indicate the target of the link is the schema for the instance object. [Zyp et al. (2013b)]</td>
</tr>
</tbody>
</table>
| R154.11 | The edit-form value MUST be used to indicate that the link target identifies a resource where a submission form for editing the associated resource can be obtained.  

*Note:* See the IANA Registry of Link Relations [IANA (2013a)]. Although it is common to have the value of the link the same as the request URI used to fetch the representation of the resource, in some cases the server may choose to offer a separate URI for editing purposes.

<table>
<thead>
<tr>
<th>R154.12</th>
<th>The enclosure value MUST be used to indicate that the link target identifies a related resource that is potentially large and might require special handling.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R154.3</td>
<td>The full value MUST be used to indicate that the target of the link is the full representation for the instance object. The object that contains this link possibly may not be the full representation. [Zyp et al. (2013b)]</td>
</tr>
</tbody>
</table>
| R154.4  | The related value MUST be used to indicate that the target of the link is a related resource.  

*Note:* See the IANA Registry of Link Relations [IANA (2013a)].

<table>
<thead>
<tr>
<th>R154.5</th>
<th>The root value SHOULD be used to indicate that the target of the link be treated as the root or the body of the representation for the purposes of user agent interaction or fragment resolution. All other data in the document can be regarded as meta-data for the document. [Zyp et al. (2013b)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>R154.6</td>
<td>The self value MUST be used to indicate the object represents a resource and the instance object is treated as a full representation of the target resource identified by the specified URI. [Zyp et al. (2013b)]</td>
</tr>
<tr>
<td></td>
<td><em>Note:</em> See the IANA Registry of Link Relations [IANA (2013a)].</td>
</tr>
</tbody>
</table>
R154.7 The **up** value **MUST** be used to indicate a parent document in a hierarchy of documents.

*Note: See the IANA Registry of Link Relations [IANA (2013a)].*

R154.8 The **oagi/invoke** value **MUST** be used to indicate the target of the link is a custom operation.

*Note: This is an OAGi-defined link relation.*

R154.13 The **first** value **MUST** be used to indicate the link context\(^{23}\) is a part of a series (e.g. "pages" of instance resources) and that the link target is the *farthest preceding* resource (or first "page") in the series.

*Note: See the IANA Registry of Link Relations [IANA (2013a)].*

R154.14 The **next** value **MUST** be used to indicate the link context is a part of a series (e.g. "pages" of instance resources) and that the link target is the next ("page") in the series.

*Note: See the IANA Registry of Link Relations [IANA (2013a)].*

R154.15 The **previous** value **MUST** be used to indicate the link context is a part of a series (e.g. “pages” of instance resources) and that the link target is the previous (“page”) in the series.

*Note: See the IANA Registry of Link Relations [IANA (2013a)].*

R154.16 The **last** value **MUST** be used to indicate the link context is a part of a series (e.g. "pages" of instance resources) and that the link target is the *farthest following* (or last "page") resource in the series.

*Note: See the IANA Registry of Link Relations [IANA (2013a)].*

R154.17 The **canonical** value **MUST** be used to indicate the target of the link is the preferred version of a resource (from resources with duplicative content).

*Note: See the IANA Registry of Link Relations [IANA (2013a)].*

\(^{23}\) By default, the context of a link is the URL of the representation with which it is associated. [Nottingham (2010)]
<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
</table>
| R154.18 | The *search* value **MUST** be used to indicate the target of the link is a resource that can be used to search through the link context and related resources.  
*Note:* See the IANA Registry of Link Relations [IANA (2013a)]. |
| R154.19 | The `/oagi/confirm-message` value **MUST** be used to indicate the target of the link is a confirm message resource. |
| R154.20 | The `/oagi/codelist` value **MUST** be used to indicate the target of the link is a codelist resource. |
| R154.21 | The `/oagi/externalLink` value **MUST** be used to indicate the target of the link is a resource external to the enterprise. |
| R154.22 | The `/oagi/callback` value **MUST** be used to indicate the target of the link is a callback function resource. |
| R154.23 | The `/oagi/processing-status` value **MUST** be used to indicate the target of the link is a resource that can be used to determine the processing status of a submitted request. |
| R154.24 | The `/oagi/request-result` value **MUST** be used to indicate the target of the link is a resource that can be used to obtain the results of a submitted request. |
| R249    | The title property of the Link object class also **MAY** be used to annotate a Link object with an application-specific type (e.g. a link to a report) |
The mediaType value MUST be limited to an element of the value domain:

- "application/json",
- "text/html",
- "application/pdf",
- "image/jpeg",
- "image/gif",
- "image/png",
- "application/vnd.openxmlformats-officedocument.presentationml.presentation",
- "application/vnd.visio",
- "image/bmp",
- "application/vnd.ms-powerpoint",
- "video/mp4",
- "audio/mpeg",
- "video/x-msvideo",
- "video/x-ms-wmv",
- "application/rtf",
- "text/csv",
- "video/quicktime",
- "application/zip",
- "application/illustrator",
- "text/xml".

Note: See the IANA Registry of MIME Media Types [IANA]. The Registry provides references (e.g. RFC) for each media type.

A mediaType value MUST be used as defined in its related RFC.

The method value MUST be limited to an element of the value domain:

- "DELETE",
- "GET",
- "POST",
- "PUT",
- "PATCH".

Note: The methods are defined in RFC 2616 [Fielding et al. (1999)].

A method value MUST be used as defined in RFC 2616 [Fielding et al. (1999)].

The encType value MUST be limited to an element of the value domain:

- "application/json",
- "application/x-www-form-urlencoded".

Note: See the IANA Registry of MIME Media Types [IANA] for the application/json media type. See the HTML 4.01 Specification [Raggett (1999)] for the application/x-www-form-urlencoded content type (or media type).
9.1 Hypermedia Actions

Hypermedia actions offer clients a set of possible next steps that may be taken on the resource in the context of client-server interaction, realizing a use case. Action object is used to describe a state-sensitive action which the user or user-agent is allowed to initiate on the related resource. An Action references an operation (i.e. CRUD operation or Custom operation).

operationCode - Identifies a resource management operation.
canonicalURI - Identifies (uniquely) the resource management operation.
confirmationRequiredIndicator - True indicates that the application will prompt the user to confirm the selected action. False indicates that the action will be executed once selected by the user.
commentAllowedIndicator - True indicates that the System of Record (SOR) accepts a comment when invoking the operation.
defaultIndicator - True indicates that this is the default action.
colorCode - Code identifying the color to associate to the action. A positive action should be green (e.g. Approval) a negative action (e.g. Denial) should be red; red=FF0000, yellow=FFFF00, green=80FF00 using RGB codes.

The Attestation contains optional text provided to the user when they take an action to affirm to be correct, true, or genuine. Use of this requires that confirmationRequiredIndicator to be set to true. The properties of the Attestation are defined below.

messageTxt - Message text presented to the user. This is mutually exclusive with the messageLink which is an external link to the message text. If the text is sizable and fairly static then it should be accessed with an external link and may be cached.

Figure 10: Hypermedia Actions Logical Model
messageLink - The link of the attestation message text that should be used instead of the messageTxt property if it can be cached.

For any Action instance, the following rule applies:

**R157** The properties **MUST** adhere to the definitions and multiplicity constraints documented in the Hypermedia Actions logical model.

**R158** The rel value **MUST** be limited to an element of the value domain:
- “create”
- “describedby”
- “full”
- “related”
- “root”
- “self”
- “up”
- “/oagi/invoke”

```json
{
  "actions": [
    {
      "operationCode": {
        "codeValue": "timeSheet.review"
      },
      "confirmationRequiredIndicator": true,
      "commentAllowedIndicator": false,
      "links": [
        {
          "rel": "oagi/invoke",
          "title": "Approve Timesheet",
          "href": "http://api.oagi.com/service-domains/time/v1/timeSheets/123/review",
          "method": "POST"
        }
      ]
    }
  ]
}
```

## 10 Confirmation Management

The American Heritage dictionary defines confirm as “to support or establish the certainty or validity of,” and even more specifically as “the act of establishing the certainty or validity.” RESTful Web APIs use the status-line of the HTTP response message to return request processing results to client. In addition, a client may request additional information on the application-level processing of its request. This additional information is made available to the client with the **Confirm Message**, an object class specifically designed for this purpose.

The first subsection, below, describes the HTTP Response Status codes and their expected use. The next subsection explains the mechanism for requesting a **Confirm Message** response. The last subsection describes the **Confirm Message** response.
10.1 HTTP Response Status

A Level 2 RESTful Web API maturity requires that message confirmation status (i.e. success, partial failure or failure) use the established HTTP response status codes.

**R159 An HTTP Response Status MUST be returned in response to a request.**

This section specifies the patterns that are acceptable for conveying confirmation status back to a requesting client. The status codes and conditions the client must be capable of handling are described, below. The practice of “suppressing status codes” and managing status only by interrogating the contents of the message-body must not be performed.

**Note:** Use of the HTTP response status codes in this specification is intended to consistent with the HTTP response status codes as defined in W3C’s HTTP 1.1 specification. Any modifications in regard to the use of the HTTP response status codes in this specification are limited to changes in requirement levels (e.g. change of requirement from a SHOULD to a MUST) or the addition of details specific to their use in a RESTful Web API. The purpose of these modifications is to constrain the space of response code usage to that required for partner interaction in a trading community.

There are about 60 HTTP response status codes [IANA (2012)]. A subset of these codes is used in this specification.

**R160 Any HTTP response code not mentioned in this section, MUST NOT be used.**

There are 5 categories of response status codes:

- 1xx: Informational
- Request received, continuing process
- 2xx: Success
- The request was successfully received, understood and accepted
- 3xx: Redirection
- Further action must be taken in order to complete the request
- 4xx: Client Error
- The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error
- The server failed to fulfill an apparently valid request

The table below summarizes the HTTP response codes supported in this specification.

<table>
<thead>
<tr>
<th>Category</th>
<th>Code</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2xx</td>
<td>200</td>
<td>OK</td>
<td>The request was successful and the server’s response includes the requested data.</td>
</tr>
<tr>
<td></td>
<td>201</td>
<td>Created</td>
<td>The request has been fulfilled and resulted in a new resource being created.</td>
</tr>
<tr>
<td></td>
<td>202</td>
<td>Accepted</td>
<td>The request has been accepted for processing, but the processing has not been completed.</td>
</tr>
<tr>
<td></td>
<td>204</td>
<td>No Content</td>
<td>The server has fulfilled the request but does not need to return an entity-body, and might want to return updated metadata.</td>
</tr>
<tr>
<td>Category</td>
<td>Code</td>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>206</td>
<td>206</td>
<td>Partial Content</td>
<td>The server has fulfilled the partial GET request.</td>
</tr>
<tr>
<td>207</td>
<td>207</td>
<td>Multi-Status</td>
<td>The server conveys multiple status code about multiple resources managed in the request.</td>
</tr>
<tr>
<td>3xx</td>
<td>301</td>
<td>Moved Permanently</td>
<td>The requested resource has been assigned a new permanent URI and any future references to this resource SHOULD use one of the returned URIs.</td>
</tr>
<tr>
<td></td>
<td>303</td>
<td>See Other</td>
<td>The response to the request can be found under a different URI and SHOULD be retrieved using a GET method on that resource.</td>
</tr>
<tr>
<td></td>
<td>304</td>
<td>Not Modified</td>
<td>If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server SHOULD respond with this status code.</td>
</tr>
<tr>
<td></td>
<td>307</td>
<td>Temporary Redirect</td>
<td>The requested resource resides temporarily under a different URI.</td>
</tr>
<tr>
<td>4xx</td>
<td>400</td>
<td>Bad Request</td>
<td>The request could not be understood by the server due to malformed syntax.</td>
</tr>
<tr>
<td></td>
<td>401</td>
<td>Unauthorized</td>
<td>The request requires user authentication. If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials.</td>
</tr>
<tr>
<td></td>
<td>403</td>
<td>Forbidden</td>
<td>The server understood the request, but is refusing to fulfill it.</td>
</tr>
<tr>
<td></td>
<td>404</td>
<td>Not Found</td>
<td>The server has not found anything matching the Request-URI.</td>
</tr>
<tr>
<td></td>
<td>405</td>
<td>Method Not Allowed</td>
<td>The request method is not allowed for the resource identified by the request URI.</td>
</tr>
<tr>
<td></td>
<td>406</td>
<td>Not Acceptable</td>
<td>The API is not able to generate the any of the client’s preferred content characteristics according to the request’s accept headers.</td>
</tr>
<tr>
<td></td>
<td>408</td>
<td>Request Timeout</td>
<td>The client did not produce a request within a predetermined quantity of time.</td>
</tr>
<tr>
<td></td>
<td>409</td>
<td>Conflict</td>
<td>The request could not be completed due to a conflict with the current state of the resource.</td>
</tr>
<tr>
<td></td>
<td>410</td>
<td>Gone</td>
<td>The requested resource is no longer available at the server and no forwarding address is known.</td>
</tr>
<tr>
<td></td>
<td>412</td>
<td>Precondition Failed</td>
<td>The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.</td>
</tr>
<tr>
<td>Category</td>
<td>Code</td>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>413</td>
<td>Request Entity Too Large</td>
<td>The requested resource is larger than the server is willing or able to process.</td>
</tr>
<tr>
<td></td>
<td>415</td>
<td>Unsupported Media Type</td>
<td>The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.</td>
</tr>
<tr>
<td></td>
<td>416</td>
<td>Requested Range Not Satisfiable</td>
<td>The server is unable to satisfy a request for a partial resource representation expressed as a byte range in the Range header of the request.</td>
</tr>
<tr>
<td>5xx</td>
<td>500</td>
<td>Internal Server Error</td>
<td>The server encountered an unexpected condition which prevented it from fulfilling the request.</td>
</tr>
<tr>
<td></td>
<td>501</td>
<td>Not Implemented</td>
<td>The server does not support the functionality to fulfill the request.</td>
</tr>
<tr>
<td></td>
<td>503</td>
<td>Service Unavailable</td>
<td>The server is currently unable to handle the request due to a temporary overloading or maintenance of the server</td>
</tr>
</tbody>
</table>

Table 9: HTTP Response Status Codes
<table>
<thead>
<tr>
<th>HTTP Response Code</th>
<th>OPTIONS</th>
<th>GET</th>
<th>HEAD</th>
<th>PATCH</th>
<th>POST</th>
<th>PUT</th>
<th>DELETE</th>
<th>OPTIONS</th>
<th>GET</th>
<th>HEAD</th>
<th>PATCH</th>
<th>PUT</th>
<th>DELETE</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>201</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>✓</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>202</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>204</td>
<td>--</td>
<td>✓</td>
<td>--</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>--</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>206</td>
<td>--</td>
<td>✓</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>207</td>
<td>--</td>
<td>✓</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>301</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>303</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>304</td>
<td>--</td>
<td>✓</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>307</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>400</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>401</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>403</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>404</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>405</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>406</td>
<td>--</td>
<td>✓</td>
<td>--</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>408</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>409</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>410</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>412</td>
<td>--</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>413</td>
<td>--</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
With the exception of a POST request, the URI query component may serve to limit the instance resources to a subset of the collection.

Table 10: HTTP Response Status Code Usage
10.1.1 1xx Informational

**R161** All 1xx (Informational) responses **MUST NOT** include a message-body.

10.1.2 2xx Success

This category of response status codes indicates that the request from the client was received, understood, and accepted.

**R162 200 OK** - response status code **SHOULD** be used to inform the client that the request succeeded.

*Note:* This is the most common Status Code and should occur most of the time.

**R162.1** For a GET request, specifying a resource, the server **MAY** return a resource representation in the message-body in the media type specified by the accept headers in the request.

**R162.7** For a GET request, specifying a resource, the server (upon not finding the requested resource) **MUST NOT** return a resource representation in the message-body and **MUST** return the response meta data parameter, `totalNumber` (of instance resources) assigned to the value of "0".

**R162.2** For a HEAD request, the server **MUST NOT** return a message-body.

**R162.3** For a PUT or PATCH request, specifying a resource, the server **MAY** return a resource representation for the updated resource in the message-body (with the content characteristics specified by the accept headers in the request) or return a Confirm Message entity in the message-body (to describe the results from processing the request).

**R162.4** For a custom operator request (i.e., controller resource), the server **MAY** return a resource representation in the message-body (with the content characteristics specified by the accept headers in the request) or return a Confirm Message entity in the message-body (to describe the results from processing the request).

**R162.5** A 200 response status code **MUST NOT** be used to communicate errors.

**R162.6** For GET and HEAD requests caching expiration headers, `Cache-Control: max-age` and `Expires`, **MAY** be used.

A 201 status code indicates that the request has been fulfilled and resulted in an instance resource being created in a collection. In the case of a controller resource that as the result of its execution creates an instance resource, the 200 status must be used.

**R163 201 Created** - response status code **MUST** be used to inform the client that the resource was successfully created.
R163.1 The origin server **MUST** successfully create the new resource before returning the 201 response status code.

R163.2 The server **MAY** return the current value of the entity tag (for representation of the resource just created) in the ETag header.

R163.3 For a POST request of an instance resource, the server **MUST** return the URI for the new resource in the Location header.

R163.4 For a POST request of an instance resource, the server **MAY** return a resource representation for the newly created resource in the message-body (with the content characteristics specified by the accept headers in the request) or return a Confirm Message entity in the message-body (to describe the results from processing the request).

If the request can't be fulfilled immediately, the server must return a 202 status code to indicate successful acceptance of a request that has not completed processing. The response representation should include the request’s current processing status. [Fielding et al. (2014)]

R164 202 Accepted – response status code **MUST** be used to inform the client that a request was accepted for processing, but the processing has not been completed.

*Note:* A 202 Accepted response status code is the preferred method to indicate that asynchronous processing is occurring. Please see status code 303 to indicate that processing has completed.

A 204 status code indicates that the server has completed the request but does not need to return a message-body. This status code is usually sent out in response to an unsafe request such as a POST, PUT and DELETE request. This indicates that the server has completed the state transition, but declines to send back any representation or description of the state transition. A 204 may also be used in conjunction with a GET request to indicate that the requested resource exists, but has no state representation to return in the message-body. [Masse (2011), Richardson (2013)]

R165 204 No Content – response code **MUST** be used to inform the client that the message body is intentionally empty.

R165.1 If the client is a user agent, the client **SHOULD NOT** refresh the view that caused the request to be sent.

R165.2 The server **MAY** return updated HTTP Header information

R165.3 The server **MUST NOT** return a message-body in the response.

A 206 status code indicates that the server has completed the partial GET request for the resource.
R166 206 Partial Content – response code MUST be used to inform the client that the message body contains a partial resource representation as requested by the client in the Range header in the GET request.

R166.1 The server MUST return the Content-Range, Date, ETag (if the header would have been sent in a 200 OK response), Expires (if the field value might differ from that sent in a previous response for the same resource representation), and Cache-Control (if the field value might differ from that sent in a previous response for the same resource representation).

R166.2 The server MUST NOT return a byte range of "*" (indicating unknown) in the Content-Range header.

A 207 status code indicates that the server partially completed (i.e., successfully processed) a request that manage multiple resources. Multiple statuses are returned in the response, one status for each resource in the request.

R167 207 Multi-Status - response status code MUST be used to inform the client that the request, managing multiple resources, partially succeeded.

Note: See IETF RFC 4918 [Dusseault (2007)].

R167.1 The client MUST NOT repeat the request without modifications. This status code refers to the message body and not header information.

R167.2 The server MUST return a Confirm Message entity in the message-body that contains a status code for each resource along with detailed information related to the resource.

10.1.3 3xx Redirection

This category of response status codes indicates that further action needs to be taken by the user agent in order to fulfill the request. The action required may be carried out by the user agent without interaction with the user if and only if the method used in the second request is GET or HEAD. A client must detect infinite redirection loops, since such loops generate network traffic for each redirection. [Fielding et al (1999)]

For any 3xx response status code, the following rules apply:

R168 If the request method is GET or HEAD, the user agent MAY take further action without interacting with the user.

R168.1 A client MUST detect infinite redirection loops.

R169 Cache expiration headers, Cache-Control: max-age and Expires headers, MAY be used in responses for negative approach to cache validation to reduce the amount of redirecting and error processing load on the server.
The server MAY return the Retry-After header to indicate the minimum time the user agent is asked to wait before issuing the redirection request.

A 301 status code indicates that the RESTful Web API’s resource model has been redesigned; as a result, a new permanent URI has been assigned to the requested resource.

R171.301 Moved Permanently - response status code SHOULD be used to inform the client that the resource was relocated.

R171.1 The server MUST use one of the methods below to return the URI(s) to the client.

R171.1.1 The server SHOULD specify the new URI in the response’s Location header.

R171.1.2 The server MAY specify multiple URI’s by returning a Confirm Message entity that contains references to multiple resources.

R171.2 If the client receives the 301 status code in response to a request other than GET or HEAD, the client MUST NOT automatically redirect the request unless it can be confirmed by the user or validated by the application (if used as an API) (as this might change the conditions under which the request was issued).

R171.3 Any future requests by the client to the relocated resource MUST use the new URI.

A 303 status code indicates that the response to the request can be found at a different URI and should be retrieved using GET method on that resource.

R172 303 See Other - response status code SHOULD be used to refer the client to the returned URI(s).

R172.1 The server MUST use one of the methods below to return the URI(s) to the client.

R172.1.1 The server SHOULD specify the different URI in the response’s Location header for identifying a single resource.

R172.1.2 The server MAY specify multiple URI’s by returning a Confirm Message entity that contains references to multiple instance resources.

A 304 status code indicates, for a conditional GET request, that while state information exists for the resource, the client already has the current state information. By avoiding the unnecessary return of a message-body (i.e. resource representation) bandwidth is preserved.

R173 304 Not Modified - response status code SHOULD be used to indicate to the client that it already has the current (most recent) resource representation.

R173.1 The server MUST NOT return a message-body in the response.

A 307 status code indicates that the requested resource temporarily resides under a different URI. As a result, the client should resubmit the resource request to a temporary
URI specified in the response. Use of the response status code should be reserved for certain scenarios such as disaster recovery.

**R174307 Temporary Redirect** - response status code **MUST** be used to inform the client to resubmit the request to another URI.

**R174.1** The server **MUST** use one of the methods below to return the URI(s) to the client.

**R174.1.1** The server **SHOULD** specify the different URI in the response’s **Location** header for identifying a single resource.

**R174.1.2** The server **MAY** specify multiple URI’s by returning a **Confirm Message** entity that contains references to multiple instance resources.

**R174.2** If the client receives the 307 status code in response to a request other than GET or HEAD, the client **MUST NOT** automatically redirect the request unless it can be confirmed by the user or validated by the application (if used as an API) (as this might change the conditions under which the request was issued).

10.1.4 4xx Client Error

This category of response status codes is intended for cases where an error condition is generated due to an invalid request by the client. These status codes are applicable to any request method.

**For any 4xx response status code, the following rules apply:**

**R175** Cache expiration headers, **Cache-Control: max-age** and **Expires** headers, **MAY** be used in responses for **negative caching** to reduce the amount of redirecting and error processing load on the server.

A 400 status code indicates that the request could not be understood by the service due to a syntax error in the client request. Some examples include:

The request URI query component may include a parameter that is undefined in the specification.

The request resource representation in the entity-body may not conform to the resource representation schema.

**R176400 Bad Request** - response status code **MUST** be used to inform the client that the request could not be understood by the server due to malformed syntax.

**R176.1** The client **SHOULD NOT** repeat the request without modifications. This status code refers to the message body and not header information. Please see status code 412 for indicating issues with header information.

**R176.2** The server **MUST** return a **Confirm Message** entity in the message-body that contains the detailed information related to the error.
If no other 4xx response code is appropriate, then the 400 response code status SHOULD be used as a generic client-side error status [Richardson (2013)].

A request to controller resource includes an HTTP method, specified in the request-line, and a custom operator specified in the request-URI. The 400 status code will also be used to indicate that the client tried to use a custom operator that is not allowed for the resource identified by the request URI.

For any controller resource, the following rule applies:

R177.3 If no other 4xx response code is appropriate, then the 400 response code status SHOULD be used as a generic client-side error status [Richardson (2013)].

A request to controller resource includes an HTTP method, specified in the request-line, and a custom operator specified in the request-URI. The 400 status code will also be used to indicate that the client tried to use a custom operator that is not allowed for the resource identified by the request URI.

For any controller resource, the following rule applies:

R177 400 Bad Request – response status code MUST be used to inform the client that the custom operator specified in the request is not allowed for the resource identified by the request URI.

R177.1 The response MAY include an OAGi-Allow-CustomOperator header that lists the valid custom operators for the requested resource.

A 401 status code indicates that the request lacked the proper authorization to operate on a protected resource. For example, the client may have provided incorrect credentials.

R178 401 Unauthorized – response status code MUST be used to inform clients that the authorization has been refused for credentials submitted in a request on a protected resource (e.g. the request may have provided wrong credentials or none at all).

Note: If the server does not wish to make know why the request was not fulfilled, see rules on response status code 404.

R178.1 The client MAY repeat the request with a new set of credentials.

R178.2 The server MUST return the WWW-Authenticate header.

R178.3 The server MAY return a Confirm Message entity that contains the detailed information related to the error.

A 403 status code indicates that the request was understood but refused by the server. RESTful Web APIs use 403 to enforce application-level permissions. For example, a client may be authorized to interact with some, but not all resources of an API. [Masse (2011)] The 403 response is also used in cases where the resource may only be accessible at certain times or from certain IP addresses. [Richardson (2013)]

R179 403 Forbidden – response status code MUST be used to inform clients that attempted to interact with a resource beyond its permitted scope.

Note: If the server does not wish to make know why the request was not fulfilled, see rules on response status code 404.

R179.1 Authorization will not help, and the request SHOULD NOT be repeated.

Note: In other words, the response is not merely a case of insufficient client credentials (for which a 401 response is designated); therefore, resubmitting the request for authorization will not resolve the error.
If the server wishes to make known why the request was not fulfilled, it SHOULD describe the reason for the refusal in a Confirm Message entity.

Note: If the server does not wish to make know why the request was not fulfilled, see rules on response status code 404.

The 404 status code indicates that the request URI cannot be matched against a resource. The resource may be represented by a combination of uri and header values and the server did not find a match for what was requested.

R180 404 Not Found – response status code MAY be used to inform clients that the server has not found anything matching the description of the resource that was requested.

If the server does not wish to inform the client why the request was not fulfilled, then status code 404 (Not Found) SHOULD be used.

The 405 status code indicates that the client tried to use an HTTP method that is not allowed for the resource identified by the request URI.

R181 405 Method Not Allowed – response status code MUST be used to inform client that the HTTP method specified in the request is not allowed for the resource identified by the request URI.

The response MUST include an Allow header that lists the valid HTTP methods for the requested resource.

The 406 status code indicates that the API is not able to generate any of the client’s preferred content characteristics for identified resource, according to the accept headers in the request. For example, the client request Accept header specified the media type as application/json, but the API can only represent the data as application/xml.

R182 406 Not Acceptable – response status code MUST be used to inform the client that the resource identified by the request is not capable of generating a response entity with acceptable content characteristics, according to the Accept headers.

Note: Servers are allowed to return responses that do not satisfy the conditions of the Accept headers; this may be preferable in some cases. [Fielding et al (1999)]

The response MAY return a Confirm Message entity that contains a list of available content characteristics and the location(s) from which the user or user-agent may choose the most appropriate.

The 408 status code indicates that the client did not produce a complete request in some predetermined time (usually specified in the server’s configuration).

R183 408 Request Timeout – response status code MUST be used to inform the client that it did not produce a request within the time that the server was prepared to wait.

The client MAY repeat the request without modifications at any later time.
A controller resource **MAY** return a 408 status if the controller resource can determine that it cannot respond within a reasonable timeframe. This is typically done at the controller level if holding resources for an extended period of time could adversely affect the system or other systems with which it is interacting.

The 409 status code indicates that the request could not be completed due to a conflict with the current state of the resource. Conflicts are most likely in response to a PUT request where changes to a resource conflict with earlier changes (e.g. made by another client) to that resource.

**R184.09 Conflict** – response status code **MUST** be used to inform the client that its request could not be completed due to a conflict with the current state of the resource.

The server **MAY** return a **Confirm Message** entity that contains the detailed information related to the error that allows the user or user-agent to fix the problem.

The 410 status code indicates that the requested resource is no longer available at the server and no forwarding address is available.

**R185.10 Gone** – response status code **MUST** be used to inform the client that the requested resource is no longer available at the server and no forwarding address is available.

The server **SHOULD** use this response code for APIs that have reached end of life, have been permanently removed and for which there are no replacements.

The 412 status code indicates that one or more of the pre-conditions of the request specified in the IF... headers were not satisfied and as a result the request was not fulfilled.

**R186.12 Precondition Failed** – response status code **MUST** be used to inform the client that one or more of the pre-conditions specified in the request-header fields were not satisfied (i.e. evaluated to false on the server) and prevented the request from being fulfilled.

The server **MAY** return a **Confirm Message** entity that contains the detailed information related to the error.

The 413 status code indicates that the requested resource is larger than the server is willing or able to process.

**R187.13 Request Entity Too Large** – response status code **MUST** be used to inform the client that the request entity is larger than the server is willing or able to process.

*Note:* The status code also applies to GET requests, using pagination, if the $stop parameter value is greater than what the server is willing to process.

The server **MAY** return a **Confirm Message** entity that contains the detailed information related to the error.

If the condition is temporary, the server **SHOULD** include a Retry-After header field in the response that specifies when the client may try the request again.
The 415 status code indicates that the API is not able to process the media type of the client’s request, as indicated by the **Content-Type** entity header. For example, a client request included an entity formatted as **application/json**, but the API can only process data formatted as **application/xml**.

**R188.415 Unsupported Media Type** – response status code **MUST** be used to inform the client that the entity of the request is in a format, as specified by the media type given in the **Content-Type** header, not supported by the API.

**R188.1** The response **MAY** return a **Confirm Message** entity that contains a list of available content characteristics and the location(s) from which the user or user-agent may choose the most appropriate.

The 416 status code indicates that the API is not able to satisfy the client request for a partial resource, as indicated in the **Range** request header.

**R189 416 Requested Range Not Satisfiable** – response status code **MUST** be used to inform the client that the partial resource expressed as a range in the **Range** header of the request does not overlap with any of the ranges of the resource available at the server.

**R189.1** The server **SHOULD** specify the current length of the selected resource in the response’s **Content-Range** header.

### 10.1.5 5xx Server Error

This category of response status codes is intended for cases where the server is aware that it has erred or is incapable of performing the request. These status codes are applicable to any request method.

**For any 5xx response status code, the following rules apply:**

**R190** Except for responding to a **HEAD** request, the server **SHOULD** return a **Confirm Message** entity that contains the detailed information related to the error, and whether it is a temporary or permanent condition. [Fielding et al (1999)]

The 500 status code indicates that the server malfunctioned. The 500 error is never the fault of the client; therefore, it is reasonable for the client to retry the exact same request that triggered this response.

**R191 500 Internal Server Error** – response status code **MUST** be used to inform the client that the server encountered an unexpected condition which prevented it from fulfilling the request. This status code **MUST** be used for all errors that are not caused by the client, except for when a 503 status is more appropriate.

**R191.1** The client **MAY** retry the same request that triggered the response.

The 501 status code indicates that the server does not support the requested functionality. If a server does not recognize the request method and does not support it for any resource, then the 501 response status code is the appropriate response. An example includes:
• The request URI query component may include a parameter that is defined in the specification but not implemented by the server.

R192 501 Service Not Implemented – response status **MUST** be used to inform the client that the server does not support the requested functionality.

*Note:* Requested functionality refers to a request’s method and query parameter(s).

The 503 status code indicates that the server is temporarily unavailable and should be restored in the future.

R193 503 Service Unavailable – response status **MUST** be used to inform the client that the server is currently unable to handle the request due to a temporary overloading or maintenance of the server.

**R193.1** If the server knows the length of the delay, it **SHOULD** be indicated in a *Retry-After* header.

**R193.2** The client **MAY** retry the request within the timeframe specified in the *Retry-After* header. If the *Retry-After* header is not present, then it is not known when the service will be available again.

10.2 Confirm Message Request

The HTTP response status, alone, may be insufficient for conveying the results of processing requests. For these situations, a **Confirm Message** will be used to provide application-specific, detailed messages on the request’s processing result.

As described above, there are certain cases when a **Confirm Message** must, should or may be returned. For example, a request resulting in a 400 response status will always yield a **Confirm Message**.

In those cases where a **Confirm Message** is not required to be returned (yet can be made available) to the client, the client may request a **Confirm Message**. The server response must then either include a **Confirm Message** in the entity-body or include a link to the **Confirm Message** entity that is related to the processing of the related request.

**R194** A client **MUST** use the *Prefer* request header to request a Confirm Message from the server.

*Note:* See the Message Header section for details on using the *Prefer* request header for this purpose.

```
Prefer: /oagi/confirm-message
```

**R195** In response to a client Confirm Message request (that is supported by the server per the API specification), the server **MUST** produce a Confirm Message response.

*Note:* The API Specification should be used to indicate whether or not a Confirm Message can be returned for a given request message.
For any server response for which there exists a Confirm Message that was not returned in the response, the Confirm Message MUST include a link in the Link header with the link relation (rel) set to “/oagi/confirm-message”.

Note: The rules that describe use of the HTTP response status specify whether a Confirm Message must, should or may be supported.

The persistence and availability (i.e. time interval) of the URI, specified in the Link header, MUST be defined in the API Specification.

The Confirm Message is a message definition of information that provides details on request processing results (i.e., success, warning, error and informational messages). The Confirm Message does not communicate the results of the client request, but rather the results of processing the request. It is always communicated as an entity in the message-body of the response.

The Confirm Message MUST be used to communicate all application-specific success, warning, error and informational messages intended for users or client developers.

The figure below shows the Confirm Message model. The Confirm Message was designed to support the communication of response messages and status at both the request-level and at the request resource-level. This design supports resource-specific messages in batch and bulk request scenarios. The actual success, warning, error and informational messages may be communicated with the Message construct at two levels: the Confirm Message and the Resource Message. At the Confirm Message, the Message provides overall results for the request; at Resource Message, the Message provides results for the individual resource(s) being managed in the request.
The **Confirm Message** contains the processing results for the corresponding request. A request may have its processing reported as: succeeded, partially failed, or failed. The properties of the **Confirm Message** are defined, below.

**messageID** - An identifier for the instance of the confirm message. The identifier must be globally unique for storage and use (e.g. in a URI of a confirm message instance exposed in a Link Header).

**messageDateTime** - The date & time that the confirm message instance was created.

**requestID** - An identifier for an instance of a processing job (e.g. a processing job that is servicing a bulk or batch request).

**requestResultStatusCode** - The processing result status code for the request.

**requestProcessingStatusCode** - Status of the processing of the request message.

**links** - A link array that supports the return of one or more links associated with the request and/or **Confirm Message**.

The **Resource Message** contains the resource-specific processing results for resources being managed in the request. A resource may have its processing reported as: succeeded or failed. It must be used to represent resource-specific messages. In the case of a request managing multiple resources and resource-specific messages are to be returned, then this array will contain one object for each such resource of the request. The properties of the **Resource Message** are defined below.

**resourceID** - An identifier for the instance of the resource message. It must be unique within the scope of the **Confirm Message** instance.

**resourceResultStatusCode** - The result status code for a resource in a request.

**links** - A link array that supports the return of one or more links associated with the resource in a request.
The **Message** contains additional information associated with either the request (associated with the **Confirm Message**) and/or resources being managed in the request (associated with the **Resource Message**). Most often there will be a single instance for an associated request or resource, but this structure allows for more than one if needed, for example, if multiple errors exist for a single resource. The properties of the **Message** are defined below.

**messageCode** – A code that is associated with the content of the message.

**messagetypeCode** – Process Message instances may be of type: success, warning, error, or info.

**message** – The content or description of the message.

**resourcePath** – A path expression used to specify the part of a resource representation that corresponds to the message.

For any Confirm Message instance, the following rules apply:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R198</td>
<td>The properties MUST adhere to the definitions and multiplicity constraints documented in the <strong>Confirm Message</strong> logical model.</td>
</tr>
<tr>
<td>R199</td>
<td>The <strong>messageID</strong> value MUST be globally unique for storage and use (e.g. in a URI of a Confirm Message instance exposed in a Link Header).</td>
</tr>
<tr>
<td>R279</td>
<td>The <strong>processID</strong> value MUST be globally unique for storage and use (e.g. in a URI of the processing status for a processing job).</td>
</tr>
</tbody>
</table>
| R202 | The **requestResultStatusCode** value MUST be limited to an element of the value domain:  
| | “succeeded”,  
| | “partiallyFailed”,  
| | “failed”. |
| R202.1 | The **succeeded** value MUST be used to indicate that the processing of the request was successful; the management of all instance resources in the request was successful. |
| R202.2 | The **partiallyFailed** value MUST be used to indicate that the processing of the request partially failed; **partiallyFailed** requests are limited to cases where the management of at least one of multiple instance resources in the request was unsuccessful (e.g. bulk operations). |
| R202.3 | The **failed** value MUST be used to indicate that the processing of the request failed; the management of all instance resources in the request was unsuccessful. 

*Note:* For bulk operations, a given API Specification may specify what constitutes a failed request. In some cases, a processing error on a single resource instance may warrant the request as failed; in other cases, processing errors on all instances resources may warrant the request as failed.
R204 The **requestProcessingStatusCode** value **MUST** be limited to an element of the value domain:

- “received”
- “validated”
- “started”
- “completed”

*Note:* A request that is in a “received”, “validated” and “started” state is considered to be in-progress.

R204.1 The **received** value **MUST** be used to indicate that the request has been received by the server but has not been processed.

R204.2 The **validated** value **MUST** be used to indicate that the request has been validated against applicable business rules but has not been completed processing.

R204.3 The **completed** value **MUST** be used to indicate that the request has completed processing.

R204.4 The **started** value **MUST** be used to indicate that the request has begun processing (i.e. that the request is being executed).

R205 The **Resource Message.resourceID** value **MUST** be unique within the scope of the **Confirm Message** instance.

R206 The **Resource Message.resourceResultStatusCode** value **MUST** be limited to an element of the value domain:

- “succeeded”
- “failed”

R206.1 The **succeeded** value **MUST** be used to indicate that the processing of the instance resource was successful.

R206.2 The **failed** value **MUST** be used to indicate that the processing of the instance resource failed.

R280 The **Message.resourcePath** value **MUST** be written in one of the following expression languages:

- “XPath”
- “jPath”

R280.1 The **XPath** value **MUST** adhere to the format and value domains as specified in W3C’s XML Path Language.

*Note:* See [Berglund et al. (2010)].
The jPath value MUST adhere to the format and value domains as specified by Stefan Goessner.

Note: See [Goessner (2007)].

The Message.messageTypeCode value MUST be limited to an element of the value domain: "success", "warning", "error", "info".

The success value MUST be used to identify the Message as a success message.

The warning value MUST be used to identify the Message as a warning message.

The error value MUST be used to identify the Message as an error message.

The info value MUST be used to identify the Message as an informational message.

In the following example the Confirm Message communicates the result of a request that failed upon validation.

```json
{
    "confirmMessage": {
        "messageID": "69fe0381-ed80-45ef-b4c7-41e2db362b91",
        "messageDateTime": "2019-03-11T15:30:00-06:00",
        "requestProcessingStatusCode": "validated",
        "requestResultStatusCode": "failed",
        "messages": [
            {
                "messageCode": "10003",
                "messageTypeCode": "error",
                "message": "Unexpected parameter, 'select' in the URI Query Component."
            }
        ]
    }
}
```

In the following example the Confirm Message communicates the result of a partially failed where the first resource in the request processed successfully and the second resource in the request failed to process.

```json
{
    "confirmMessage": {
        "messageID": "69fe0381-ed80-45ef-b4c7-41e2db362b91",
        "messageDateTime": "2019-03-11T15:30:00-06:00",
        "requestProcessingStatusCode": "validated",
        "requestResultStatusCode": "failed",
        "messages": [
            {
                "messageCode": "10003",
                "messageTypeCode": "error",
                "message": "Unexpected parameter, 'select' in the URI Query Component."
            }
        ]
    }
}
```
{"confirmMessage": {
  "messageID": "69fe0381-ed80-45ef-b4c7-41e2db362b91",
  "messageDateTime": "2019-03-11T15:30:00-06:00",
  "requestProcessingStatusCode": "completed",
  "requestResultStatusCode": "partiallyFailed",
  "messages": [
    {
      "messageCode": "PREVIEW_CALC_ERRORS",
      "messageTypeCode": "error",
      "message": "Error(s) occurred while previewing the payrun."
    }
  ],
  "resourceMessages": [
    {
      "resourceMessageID": "1234",
      "resourceResultStatusCode": "failed",
      "messages": [{
        "messageCode": "PREVIEW_CALC_ERROR",
        "message": "An error has occurred in the payroll calculation. Please contact your Payroll support team for assistance. (Message ID 068)",
        "resourcePath": "]\employees[?(@.associateOID='AOID_1')]"
      }
    },
    {
      "resourceMessageID": "2345",
      "resourceResultStatusCode": "succeeded",
      "messages": [{
        "messageCode": "PREVIEW_SUCCESS",
        "message": "Employee processed successfully",
        "resourcePath": "]\employees[?(@.associateOID='AOID_2')]"
      }
    }
  ]
}
The following table shows the relationship of code values between `Confirm Message.requestResultStatusCode`, the `Resource Message.resourceResultStatusCode` and the `Message. messageTypeCode` (for the Confirm Message and the Resource Message). A `requestResultStatusCode` of “succeeded” indicates that all of the resources in the request “succeeded”. A `requestResultStatusCode` of “partiallyFailed” indicates that some of the resources in the request “succeeded” and some of the resources in the request “failed”. A `requestResultStatusCode` of “failed” indicates that all of the resources in the request “failed”.

<table>
<thead>
<tr>
<th>Confirm Message Status</th>
<th>Confirm Message</th>
<th>Resource Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestResultStatusCode</td>
<td>Resource Message.resourceResultStatusCode</td>
<td>Message. messageTypeCode</td>
</tr>
<tr>
<td>succeeded</td>
<td>succeeded</td>
<td>success, warning, info</td>
</tr>
<tr>
<td>partiallyFailed</td>
<td>succeeded</td>
<td>success, warning, info</td>
</tr>
<tr>
<td></td>
<td>failed</td>
<td>success, warning, error, info</td>
</tr>
<tr>
<td>failed</td>
<td>failed</td>
<td>success, warning, error, info</td>
</tr>
</tbody>
</table>

Table 11: Types of Messages by Request Processing Status

For any `Confirm Message` instance, the following rules apply:

**R210** If the `Confirm Message.requestResultStatusCode` value is `succeeded`, the `Resource Message.resourceResultStatusCode` MUST be limited to the value domain: “succeeded”.

**R211** If the `Confirm Message.requestResultStatusCode` value is `partiallyFailed`, the `Resource Message.resourceResultStatusCode` MUST be limited to the value domain: “succeeded”, “failed”.

**R212** If the `Confirm Message.requestResultStatusCode` value is `failed`, the `Resource Message.resourceResultStatusCode` MUST be limited to the value domain: “failed”.

If the `Confirm Message.requestResultStatusCode` value is `succeeded` where all `Resource Message.resourceResultStatusCode` values are `succeeded`, then the `Confirm Message.Message.messageTypeCode` MUST be limited to the value domain: “success”, “warning”, “info”.

For any `Resource Message` instance, the following rule applies:

R213.1 The `Confirm Message.Resource Message.Message.messageTypeCode` MUST be limited to the value domain: “success”, “warning”, “info”.

If the `Confirm Message.requestResultStatusCode` value is `partiallyFailed` where at least one `Resource Message.resourceResultStatusCode` value is `succeeded` and at least one `Resource Message.resourceResultStatusCode` value is `failed`, then the `Confirm Message.Message.messageTypeCode` MUST be limited to the value domain: “success”, “warning”, “info”.

For any `Resource Message` instance, the following rule applies:

R214.1 If `Resource Message.resourceResultStatusCode` value is `succeeded`, then the `Confirm Message.Resource Message Message.messageTypeCode` MUST be limited to the value domain: “success”, “warning”, “info”.

R214.2 If the `Resource Message.resourceResultStatusCode` value is `failed`, then the `Confirm Message.Resource Message.Message.messageTypeCode` MUST be limited to the value domain: “success”, “warning”, “error”, “info”.

Note: In the case of failed instance resource processing, success messages may still be communicated. For example, the instance resource representation may have successfully validated against its schema.
R215 If the `Confirm Message.requestResultStatusCode` value is `failed` where all `Resource Message.resourceResultStatusCode` values are `failed`, then the `Confirm Message.Message.messageTypeCode` MUST be limited to the value domain: “success”, “warning”, “error”, “info”.

For any Resource Message instance, the following rule applies:


The following table relates the `Confirm Message.requestResultStatusCode` and the `Confirm Message.Resource Message.resourceResultStatusCode` to the HTTP Response Status Codes.

<table>
<thead>
<tr>
<th>Confirm Message Status</th>
<th>HTTP Response Status for Request managing a</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestResultStatusCode</td>
<td>Single Instance Resource</td>
</tr>
<tr>
<td>succeeded</td>
<td>2xx</td>
</tr>
<tr>
<td>partiallyFailed</td>
<td>--</td>
</tr>
<tr>
<td>failed</td>
<td>3xx, 4xx, 5xx</td>
</tr>
</tbody>
</table>

Table 12: Confirm Message Status to HTTP Response Status Map

For any `Confirm Message` instance in response to a request managing a single instance resource, the following rules apply:

**R216** The `Confirm Message.requestResultStatusCode` value of `succeeded` MUST be used in conjunction with a 2xx HTTP Response Status Code.

**R217** The `Confirm Message.requestResultStatusCode` value of `failed` MUST be used in conjunction with a 4xx or 5xx HTTP Response Status Code.

For any `Confirm Message` instance in response to a request managing multiple instance resources, the following rules apply:
<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R218</td>
<td>The <code>Confirm Message.requestResultStatusCode</code> value of <code>succeeded</code> <strong>MUST</strong> be used in conjunction with a 2xx HTTP Response Status Code.</td>
</tr>
<tr>
<td>R219</td>
<td>The <code>Confirm Message.requestResultStatusCode</code> value of <code>partiallyFailed</code> <strong>MUST</strong> be used in conjunction with a <strong>207 Multi-Status</strong> HTTP Response Status Code.</td>
</tr>
<tr>
<td>R220</td>
<td>The <code>Confirm Message.requestStatusCode</code> value of <code>failed</code> <strong>MUST</strong> be used in conjunction with a 3xx, 4xx or 5xx HTTP Response Status Code.</td>
</tr>
</tbody>
</table>
11 Patterns for Asynchronous Communication

Case exists, for example a request for large bulk operation, requiring substantial server processing that prohibit a synchronous response. In these cases, the results of the request must be communicated asynchronously.

Three patterns are provided to communicate the results in an asynchronous manner. In the first pattern the Service Provider pushes the response to the client. In the second pattern the Service Consumer pulls the response from the server. In the third approach the Service Consumer polls the processing status of the request and pulls the response.

Factors for consideration in the selection of a pattern include:

- The application architecture of the service provider where,
  - In the push pattern, the service provider has responsibility for monitoring request processing and communicating the results upon processing completion.
  - In the polling pattern, the service consumer has responsibility for monitoring request processing and retrieving the results upon processing completion.
- The applicability of the interactions patterns for the use cases, supported by the API.

Note: All the asynchronous communications patterns may leverage the OAGi-Originator-ID header to facilitate routing of the message instances participating in the collaboration.

For any request, the following rules apply:

R272 The Service Provider, upon assessing a request and/or Service Provider constraints (e.g. capacity), MAY respond to the request asynchronously.

Note: Even though a Service Consumer does not specify a preference for an asynchronous response, the Service Provider may elect to respond to the request asynchronously.

For any request to which a service consumer prefers an asynchronous response, the following rules apply:

R221 The Service Consumer MUST include a Prefer header with a field value of “respond-async” in the request.

Note: This informs the service provider of the service consumer’s preference for an asynchronously response to the request.

R221.1 The Service Consumer MAY include a Prefer header with a field value of “wait = time”.

Note: The “wait = time” expression acts as a condition for an asynchronous response. If the server cannot generate a response within the wait time specified, then the response must be returned asynchronously.
11.1 A pattern for Service Provider Push

This asynchronous communication pattern, allows Service Consumer to receive an asynchronous response that is pushed from the Service Provider.

Application of this pattern may be limited to internal only. External use presents security management concerns, specifically the Service Consumer would be required to authenticate and authorize the Service Provider callback.

Use of this pattern must use the following systems interaction, described below.

**Figure 12: Asynchronous Service Provider Push Response Pattern**

1. The Service Consumer sends a request to a Service Provider.
   The request must include the following headers:
1. OAGi-MessageID header; the header identifies the request.

2. Prefer header with a field value of “respond-async”; the header informs the Service Provider to respond asynchronously.

3. Link header with a relation-type (i.e., “rel”) value of “oagi/callback”; the header specifies the URI that the Service Provider may use to push the request result and/or request result status.

Note: See the Request Headers section of the document for details on the Prefer and Link headers and see the Custom Headers section for details on the OAGi-MessageID headers.

2. For a successful request, the Service Provider must return a 202 Accepted status code in the response. For unsuccessful requests, see the Confirmation Management section for a list of possible error status codes.

3. Once the Service Consumer’s request has completed processing by the Service Provider, the Service Provider, using the call back URI, sends a POST request to the Service Consumer that includes the request result and may include request result status of the resource management operation.

The request must include the following headers:

- OAGi-CorrelationID header; the header identifies the original request to which the current request is related.

The request may include the following headers:

- OAGi-MessageID header; the header identifies the request.

4. Upon a successful request, the Server Consumer must return a 201 Created status code. For unsuccessful requests, see the Confirmation Management section for a list of possible error status codes.

5. Service Consumer is responsible for ensuring that the OAGi-CorrelationID value matches the originating OAGi-MessageID value.

6. Alternative to Steps 3-5: Once the Service Consumer’s request has completed processing by the Service Provider, the Service Provider, using the call back URI, sends a POST request to the Service Consumer that includes only request result status.

The request must include the following headers:

- OAGi-CorrelationID header; the header identifies the original request to which the current message is related.

- SOR header; the header identifies the receiver of the request.

- Link header with a relation-type (i.e., “rel”) value of “oagi/request-result”; the header specifies the URI that the Service Consumer may use to pull the request result.
The request may include the following headers:

- **OAGi-MessageID** header; the header identifies the request.

7. Upon a successful request, the Server Consumer must return a **200 OK** status code. For unsuccessful requests, see the Confirmation Management section for a list of possible error status codes.

8. Service Consumer is responsible for ensuring that the **OAGi-CorrelationID** value matches the originating **OAGi-MessageID** value.

**For any request in the asynchronous pattern for service provider push, the following rules apply:**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R282</td>
<td>The service consumer request <strong>MUST</strong> include an <strong>OAGi-MessageID</strong> header that specifies the request identifier.</td>
</tr>
<tr>
<td>R222</td>
<td>The service consumer request <strong>MUST</strong> include a <strong>Link</strong> header with a <strong>relation-type</strong> value of “/oagi/callback” that specifies the URI for the callback resource for communicating request results and/or request results status.</td>
</tr>
<tr>
<td>R222.1</td>
<td>The persistence and availability (i.e. time interval) of the URI, specified in the <strong>Link</strong> header, <strong>MUST</strong> be defined in the API Specification.</td>
</tr>
<tr>
<td>R223</td>
<td>The service provider processing the request <strong>MUST</strong> return a <strong>202 Accepted</strong> status.</td>
</tr>
<tr>
<td>R224</td>
<td>The service provider, upon request processing completion, <strong>MUST</strong> call back the service consumer including the request result and/or request result status using a <strong>POST</strong> request to the URI specified in the <strong>Link</strong> header of the initial request.</td>
</tr>
<tr>
<td>R224.1</td>
<td>The request result <strong>MUST</strong> include a resource representation (for output data).</td>
</tr>
<tr>
<td>R224.2</td>
<td>The request result status, if included, <strong>MUST</strong> be represented with a <strong>Confirm Message</strong>.</td>
</tr>
<tr>
<td>R224.3</td>
<td>If the callback request returns only the request result status and operation results are available, then the service provider <strong>MUST</strong> include a <strong>Link</strong> header with a <strong>relation-type</strong> value of “/oagi/request-result” that specifies the URI from which the service consumer can retrieve the request result.</td>
</tr>
<tr>
<td>R225</td>
<td>The service consumer, upon successfully processing the request result and/or request result status, returned in the callback request, <strong>MUST</strong> return a <strong>201 Created</strong> status.</td>
</tr>
</tbody>
</table>

**11.2 A Pattern for Service Consumer Pull**

This asynchronous communication pattern, allows clients to pull an asynchronous response from the server.
This pattern must use the following systems interaction, described below.

1. The Service Consumer’s first request, that specifies the resource management request, must include a *Prefer* header with a field value of "respond-async" to inform the Service Provider of the Service Consumer’s preference for an asynchronous response. The *Prefer* header and field value informs the Service Provider to respond asynchronously.

*Figure 13: Asynchronous Service Consumer Pull Response Pattern*
**Note:** See the Request Headers section of the document for details on the **Prefer** header.

2. Upon a successful request, the Service Provider must return a **202 Accepted** status, a **Location** header and a **Retry-After** header with a field value indicating the minimum time the Service Consumer should wait before attempting to retrieve the result of the request. The **Location** header specifies the URI that the Service Consumer may use to pull the request result. For unsuccessful requests, see the Confirmation Management section for a list of possible error status codes.

3. Once the time constraint in the **Retry-After** header has been satisfied, the Service Consumer sends a GET request using the URI specified in the **Location** header returned in the response to the initial request.

4. If the resource management operation of the initial request has not completed processing (i.e., and is still in process), the Service Provider must issue a **200 OK** status and a **Retry-After** header with a field value indicating the minimum time the Service Consumer should wait before another attempt to retrieve the result of the request. The Service Provider should include a **Confirm Message** entity with information on the operation status.

5. Once the time constraint in the **Retry-After** header has been satisfied, the Service Consumer sends another GET request using the URI specified in the **Location** header returned in the response to the initial request.

6. Upon a successful request, the Server Consumer must return a **200 OK** status code that includes the results of the initial resource management request. For unsuccessful requests, see the Confirmation Management section for a list of possible error status codes.

---

**For any initial request in the asynchronous pattern for service consumer pull, the following rules apply:**

R227 The service provider processing the request **MUST** return a **202 Accepted** status code with a **Location** header that specifies the URI for the request result and a **Retry-After** header indicating when the client may poll the service provider.

R227.1 The persistence and availability (i.e. time interval) of the URI, specified in the **Location** header, **MUST** be defined in the API Specification.

**For any GET request, pulling the request result in the asynchronous pattern for service consumer pull, the following rules apply:**

R228 The service consumer request **MUST** use the URI specified in the **Location** header of the response to the initial request.

R229 If the status of the request is in-progress, then the service provider **MUST** return a **200 OK** status code with a **Retry-After** header indicating when the client may poll the service provide.
11.3 A Pattern for Service Consumer Polling and Pull

This asynchronous communication pattern allows service consumers to poll request status and pulls an asynchronous response from the service provider.

This pattern must use the following systems interaction, described below.
Figure 14: Asynchronous Service Consumer Polling and Pull Response Pattern

1. The Service Consumer’s first request, that specifies the resource management request, must include a Prefer header with a field value of “respond-async”. The Prefer header and field value informs the Service Provider to respond asynchronously.

Note: See the Request Headers section of the document for details on the Prefer header.
2. For a successful request, the Service Provider must return a 202 Accepted status, a Link header with a relation-type value of "/oagi/processing-status" and a Retry-After header with a field value indicating the minimum time the Service Consumer should wait before attempting to retrieve the processing status of the request. The Link header specifies the URI that the Service Consumer may use to query the processing status of the request. For unsuccessful requests, see the Confirmation Management section for a list of possible error status codes.

3. Once the time constraint in the Retry-After header has been satisfied, the Service Consumer sends a GET request using the URI specified in the Link header returned in the response to the initial request.

4. Upon the successful query for request processing status, the Service Provider checks the request processing status and determines that the request is in-progress. The Service Provider must issue a 200 OK response and should include a Confirm Message entity with information on the request status. Optionally, a Retry-After header with a field value indicating the minimum time the Service Consumer should wait before making another attempt to retrieve the request processing status.

5. Upon the successful request for request processing status, the Service Provider checks the request processing status and determines that the request has completed. The Service Provider must issue a 303 See Other response with a Location header that provides a URI for the request result.

6. The Service Consumer sends a GET request using the URI specified in the Location returned in the previous response.

7. Upon a successful request, the Server Provider must return a 200 OK status code that includes the results of the initial resource management request. For unsuccessful requests, see the Confirmation Management section for a list of possible error status codes.

For any initial request in the asynchronous pattern for service consumer polling and pull, the following rules apply:

**R232** The service provider processing the request **MUST** return a 202 Accepted status code with a Link header with a relation-type value of "/oagi/processing-status" that specifies the URI for the request processing status and a Retry-After header indicating when the client may poll the service provider.

**R232.1** The persistence and availability (i.e. time interval) of the URI, specified in the Link header, **MUST** be defined in the API Specification.

For any GET request, polling request processing status in the asynchronous pattern for service consumer polling and pull, the following rules apply:

**R233** The service consumer request **MUST** use the URI specified in the Link header of the response to the initial request.
R234 If the processing status of the request is in-progress, then the service provider MUST return a **200 OK** status code with a **Retry-After** header indicating when the client may poll the service provider.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R234</strong></td>
<td>If the processing status of the request is in-progress, then the service provider MUST return a <strong>200 OK</strong> status code with a <strong>Retry-After</strong> header indicating when the client may poll the service provider.</td>
</tr>
</tbody>
</table>

R235 If the processing status of the request is completed, then the service provider **MUST** issue a **303 See Other** response with a **Location** header that provides a URI for the request result.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R235</strong></td>
<td>If the processing status of the request is completed, then the service provider <strong>MUST</strong> issue a <strong>303 See Other</strong> response with a <strong>Location</strong> header that provides a URI for the request result.</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R235.1</strong></td>
<td>The availability (time interval) of the URI, specified in the <strong>Location</strong> header, <strong>MUST</strong> be defined in the API Specification.</td>
</tr>
</tbody>
</table>

R287 Request processing status **MUST** include a Confirm Message (for request processing status).

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R287</strong></td>
<td>Request processing status <strong>MUST</strong> include a Confirm Message (for request processing status).</td>
</tr>
</tbody>
</table>

For any GET request, pulling the request result in the asynchronous pattern for service consumer polling and pull, the following rules apply:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R236</strong></td>
<td>The service consumer request <strong>MUST</strong> use the URI specified in the <strong>Location</strong> header of the response that indicating the operation status as completed.</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R237</strong></td>
<td>The request result <strong>MAY</strong> include a resource representation (for output data) and <strong>MAY</strong> include a Confirm Message (for request result status).</td>
</tr>
</tbody>
</table>
12 Patterns for Event Notifications

This section describes communication patterns in support of event notifications. Event notifications are message instances that are published by Service Providers and consumed by interested observers, the Service Consumers. Event notifications may be provided in two forms:

- **Limited**, where the content of the message includes the event identifier and other key metadata elements such as the event creation date.
- **Full**, where the content of the message includes not only the event identifier and all metadata elements but also the event’s detailed data.

Consider, for example, when an employee gets paid. A limited event notification says that an employee has gotten paid; this is a pay event. On the other hand, a full event notification provides information about the pay event including the gross pay, deductions, and net pay.

A single pattern is described for the Service Consumer to retrieve event notification messages. The pattern will leverage the server-push mechanism of long-polling.

12.1 A Pattern for Long Polling

This event notifications pattern, allows clients to receive event notifications from the server. In the standard HTTP model, a server can neither initiate a connection with a client nor send an unrequested response to a client. Therefore, in this model, a server cannot push asynchronous event notifications to clients requiring clients to poll the server for new event notifications (i.e. the “short polling” mechanism). The client polling frequency depends on the latency that the client can tolerate in retrieving the event notifications. Client “short polling” incurs additional processing and network bandwidth consumption (due to the inefficiency of polling request and response pairs when no new event notifications are available). The long polling server-push mechanism was developed to address the problems of server-client event notification latency and use of processing and network resources. [Loreto et al. (2011)]

Loreto [Loreto et al. (2011)] defines long polling as when “the server attempts to “hold open” (not immediately reply to) each HTTP request, responding only when there are events to deliver. In this way, there is always a pending request to which the server can reply for delivering events as they occur, thereby minimizing the latency in message delivery.”

This pattern must use the following systems interaction, described below.
1. The Service Consumer’s GET request for Event Notifications must include a Prefer header with a field value of “/oagi/long-polling”. The Prefer header and field value informs the Service Provider to use the long-polling push mechanism.

Note: See the Request Headers section of the document for details on the Prefer header.

2. If Event Notifications are available at the Service Provider (for the respective subscriber), the Service Provider must return a 200 OK status along with the event notifications in the message entity-body.

3. If no event notifications become available at the Service Provider (for the respective subscriber) before a designate timeout, the Service Provider must return a 200 OK status along with a Confirm Message in the message entity-body describing the timeout condition. For unsuccessful requests, see the Confirmation Management section for a list of possible error status codes.
For any initial GET request in the event notifications long-polling pattern, the following rules apply:

**R239** The service consumer request **MUST** include a **Prefer** header with a value of “/oagi/long-polling” that informs the server to use the long-polling push mechanism.

**R240** The service provider processing the request **MUST** return any Event Notifications for the respective subscriber that are available or become available before timeout (if any) has occurred.

**R240.1** The response **MUST** include a resource representation (for Event Notifications) or a **Confirm Message** (for request result status).

Best practice always suppresses caching in a long poll response [Loreto et al. (2011)].

**For any response in the event notifications long-polling pattern, the following rule applies:**

**R241** The service provider **MUST** suppress caching.

*Note:* Refer to the Message Headers subsection on Caching.
13 Special Cases

This section addresses special cases such as alternative approaches and unique use cases.

13.1 Media Type Selection

Media type selection should be specified in a request through the use of HTTP's **Accept** header. However, cases exist where user agents may not be capable of setting an **Accept** header (e.g., a browser implementation) and requires the use of simple links that support media type selection by way of an **accept** query parameter that emulates the **Accept** header.

**R247** Media type selection **MAY** be specified by an **accept** query parameter.

```
http://api.abc.com/hr/v1/associates?accept=application/pdf
```

13.2 Multipart Message Instances

The entity-body of a message may contain more than one body-part. In the case of the multipart message, the Content-Type message header must specify the message with the multipart media type and subtype. Each body-part within the message has its own HTTP and MIME headers: Content-Type and Content-Disposition.

**For any multipart message request or response, the following rules apply:**

**R248** The **Content-Type** entity header **MUST** be included in the message's message-headers.

**R248.1** The **type/subtype** field value **MUST** be limited to an element of the value domain:
- **multipart/mixed**
  - *Note:* See RFC 2046 [Freed (1996)]
- **multipart/related**
  - *Note:* See RFC 2387 [Levinson (1998)]
- **multipart/form-data**
  - *Note:* See RFC 7578 [Masinter (2015)]

**R248.1.1** The **multipart/mixed** field value **MUST** be used to indicate that the body-parts are independent and need to be bundled in a particular order. [Freed (1996)]

**R248.1.2** The **multipart/related** field value **MUST** be used to indicate that the body-parts are inter-related and form part of an aggregate whole (i.e., the body-parts only make sense in the aggregate). [Levinson (1998)]

**R248.1.3** The **multipart/form-data** field value **MUST** be used to indicate that the body-parts encode name-value pairs where the values contain data of arbitrary media types. [Masinter (2015)] [Allamaraju et al. (2010)]

**R248.2** The **boundary** parameter value **MUST** be assigned a value.
**R248.2.1** The `boundary` parameter value **SHOULD** adhere to the format and value domains as specified in IETF’s RFC 2046.

>Note: See RFC 2046 [Freed (1996)]; the RFC recommends that that the boundary parameter should be enclosed in quotes.

The following example illustrates a multipart message header.

```
Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08jU534c0p
```

**R248.3** A body-part **MUST** be preceded by a boundary delimiter, composed of two hyphen characters followed by the boundary parameter value.

```
"--" boundary
```

**R248.3.1** The boundary delimiter **SHOULD** adhere to the format and value domains as specified in IETF’s RFC 2046.

**R248.3.2** The boundary delimiter **MUST** occur at the beginning of a line following a CRLF (line break).

>Note: See RFC 2046 [Freed (1996)]

**R248.3.3** The boundary delimiter **MUST** be terminated by either another CRLF (line break) and the body-part headers or by two CRLFs (i.e., there are no headers).

**R248.4** A body-part **MUST** consist of a header area, a blank line, and a body area.

>Note: See RFC 2046 [Freed (1996)]

**R248.5** The last body-part **MUST** be succeeded by a close boundary delimiter, composed of two hyphen characters followed by the boundary parameter value followed by two hyphen characters.

```
"-- boundary --"
```

**R248.6** The **Content-Type** entity header **MAY** be included the message body-part header area.

>Note: See RFC 2046 [Freed (1996)]

**R248.6.1** The **Content-Type** header **MUST** be used to describe the media type, subtype and masking of a body-part.

```
"Content-Type": "type"/"subtype [";" "masked" ="true" | "false"]
```

**R248.6.2** In the absence of the **Content-Type** header, the content type value **MUST** default to “text/plain”.
R271 The Content-Disposition entity header MAY be included the message body-part header area.

"Content-Disposition" "type [";" disposition-parameter"

Note: If the message body-part Content-Type is “multipart/form” then the Content-Disposition entity header must be included in the message body-part header area. See RFC 7578 [Masinter (2015)]

R271.1 The type [";" disposition-parameter] field value MUST adhere to the format and value domain as specified in IETF’s RFC 7578.

R271.1.1 The Content-Disposition header MUST contain an additional parameter of "name" where the value of this parameter is the name of the name-value pair, communicated in the body-part.

R271.1.2 If the value of the name-value pair, communicated in the body-part, is the content of a file, then the name for the file SHOULD be provided using the "filename" parameter.
The following example illustrates a message body-part with multipart/mixed content.

```
--gc0p4Jq0M2Yt08jU534c0p
Content-Type: application/json
{
  ...json content
}
--gc0p4Jq0M2Yt08jU534c0p
Content-Disposition: attachment; filename="att-1111-1.png"
Content-Type: image/jpeg
  ... encoded content
--gc0p4Jq0M2Yt08jU534c0p--
```

The following example illustrates a message body-part with multipart/form-data content.

```
--aCZ51y
Content-Disposition: form-data; name="field1"
Content-Type: text/plain; charset=UTF-8
  ... text content
--aCZ51y--
```
14 Message Body Representations

14.1 Metadata Representation

For any response message metadata, the following rule applies:

R242 The metadata MUST be represented in an object class, named paginationResponse, in the message-body.

The example, below, illustrates the GET response metadata supporting pagination.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "paginationResponse": {
        "startSequenceNumber": 1,
        "returnedNumber": 10,
        "totalNumber": 25,
        "completeIndicator": false,
        "resourceSetID": "7001"
    }
}
```

14.2 Resource Representations

For any resource representation of the resource model, the following rule applies:

R243 A resource representation MAY contain a link to itself with the link relation (rel) set to self.

For any collection resource representation, the following rule applies:

R244 A collection (array of items) resource representation MUST be contained within an anonymous root object.

Note: Anonymous refers to an unnamed object.

In the example, below, the associates collection is contained in an anonymous object. The associate instance resource shows the link to self.

```
{
    "associates": [{
        "associateID": {
            "idValue": "12121212"
        }
        "links": [{
            "href": "hr/v1/associates/12121212",
            "rel": "self"
        }]
    }]
}
```
Resource identifiers should be included in the resource representation when the resource representation is included in the request or response (e.g. PUT request).
15 References

[Allamaraju et al. (2010)]

[Barth (2011)]
Barth, A. “HTTP State Management Mechanism”, RFC 6265, IETF, April 2011.

[Berglund et al. (2010)]
http://www.w3.org/TR/xpath20/

[Berners-Lee (2005)]
http://www.ietf.org/rfc/rfc3986.txt

[Braden, R. (1989)]

[Bryant (2009)]
http://redrata.com/restful-uri-design/

[Daigneau (2012)]

[Dusseault (2007)]

[Dusseault et al. (2010)]
[Erl et al. (2012)]

[Fielding et al. (1999)]
http://www.w3.org/Protocols/rfc2616/rfc2616.html

[Fielding (2000)]

[Fielding et al. (2014)]

[Fowler (2010)]
http://martinfowler.com/articles/richardsonMaturityModel.html

[Freed (1996)]
http://www.ietf.org/rfc/rfc2046.txt

[Goessner (2007)]
Goessner, Stefan. “JSON Path – XPath for JSON”.
http://goessner.net/articles/JsonPath/

[Gregorio et al. (2012)]

[Herzum (2000)]

[IANA]
Internet Assigned Numbers Authority. “MIME Media Types”.
http://www.iana.org/assignments/media-types

[IANA (2012)]
http://www.iana.org/assignments/http-status-codes/http-status-codes.xml

[IANA (2013a)]
http://www.iana.org/assignments/link-relations/link-relations.xml
[IANA (2013b)]
http://www.iana.org/assignments/message-headers/message-headers.xml

[IANA (2013c)]
https://www.iana.org/assignments/http-parameters/http-parameters.xhtml

[IANA (2013d)]
https://www.iana.org/assignments/character-sets/character-sets.xhtml

[ISO 11179 (2003)]

[JSON.org]
http://www.json.org/

[Levinson (1998)]
http://www.ietf.org/rfc/rfc2387.txt

[Longden (2012)]
Longden, Ben. “vnd.error”
https://github.com/blongden/vnd.error

[Loreto et al. (2011)]

[Marvin et al. (2010)]

[Masinter (2015)]

[Masse (2011)]
Masse, Mark. REST API Design Rulebook, O’Reilly, 2011.

[Nottingham (2010)]
[OASIS (2014a)]
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html

[OASIS (2014b)]
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html

[Orchard (2003)]


[Preson-Werner]
Preson-Werner, T. “Semantic Versioning 2.0.0-rc.1”.
http://semver.org/

[Raggett (1999)]
http://www.w3.org/TR/html4/

[RESTPatterns (2011)]
REST & WOA Wiki, REST Patterns. 2011
http://restpatterns.org/

[Richardson (2008)]

[Richardson (2013)]
Richardson, L., Amundsen, M. RESTful Web APIs, O’Reilly, 2013.

[RubyOnRails Org (2012)]
RubyOnRails Org. “Edge Rails: PATCH is the new primary HTTP method for updates“, February 2012.

[Spainhour (1996)]

[Snell (2014)]
16 Appendix A: Message Body Alternatives

Three alternatives are available for representing OAGIS messages in "RESTful" Web APIs. The alternatives vary in their RESTful "maturity level" [Richardson (2008), Fowler (2010)] (as described in the section: Introduction) and/or the extent to which they leverage the HTTP Message Architecture (as described in the section: Message Architecture).

In the first alternative, the message-body includes the full OAGIS Business Object Document (BOD): the ApplicationArea, the DataArea (comprising the Verb and Noun). This alternative aligns to a Level 1 maturity level where HTTP protocol is used for communication and Nouns are recognized and managed as resources.

In the second alternative, the message-body includes only the OAGIS Noun or a Component, MetaHeader and PaginationResponse elements. Instead of using the ApplicationArea (of the first alternative), the smaller and more compact MetaHeader element, in addition to the Custom HTTP Message Headers (as described in section: Custom Headers), are used. This alternative aligns to a Level 2 or 3 depending on whether or not the resource representation includes hypermedia controls.

In the third alternative, the message-body includes only the OAGIS Noun or a Component and PaginationResponse element. Instead of using either the ApplicationArea (of the first alternative) or the MetaHeader (of the second alternative), the Custom HTTP Message...
Headers are used. This alternative aligns to a Level 2 or 3 depending on whether or not the resource representation includes hypermedia controls.