# Proposed changes to TMF630 REST API Design Guidelines Part 1, R17.5.1

Contents

[Proposed changes to TMF630 REST API Design Guidelines Part 1, R17.5.1 1](#_Toc11859395)

[1.1. Chapter 1, section “Request and Response HTTP headers – client side” 2](#_Toc11859396)

[1.1.1. Proposed change 2](#_Toc11859397)

[1.1.2. Current TMF630 text 2](#_Toc11859398)

[1.1.3. Proposed text 2](#_Toc11859399)

[1.2. Chapter 1, section “Request and Response HTTP headers – server side” 5](#_Toc11859400)

[1.2.1. Proposed change 5](#_Toc11859401)

[1.2.2. Current TMF630 text 5](#_Toc11859402)

[1.2.3. Proposed text 5](#_Toc11859403)

[1.3. Chapter 1, section “General HTTP headers” 7](#_Toc11859404)

[1.3.1. Proposed change 7](#_Toc11859405)

[1.4. Chapter 3, section “API Response status and exception codes” 8](#_Toc11859406)

[1.4.1. Proposed change 8](#_Toc11859407)

[1.4.2. Current TMF630 text 8](#_Toc11859408)

[1.4.3. Proposed text 8](#_Toc11859409)

[1.5. Chapter 3, section “API Media types” 13](#_Toc11859410)

[1.5.1. Proposed change 13](#_Toc11859411)

[1.5.2. Current TMF630 text 13](#_Toc11859412)

[1.5.3. Proposed text 13](#_Toc11859413)

[1.6. Chapter 3, section User, Application and Extended Error Codes 14](#_Toc11859414)

[1.6.1. Proposed change 14](#_Toc11859415)

[1.6.2. Current TMF630 text 14](#_Toc11859416)

[1.6.3. Proposed text 14](#_Toc11859417)

[1.7. Add section “Client Guidance” 15](#_Toc11859418)

[1.7.1. Proposed change 15](#_Toc11859419)

[1.7.2. Proposed text 15](#_Toc11859420)

## Chapter 1, section “Request and Response HTTP headers – client side”

### Proposed change

Replace this section with the below text to:

* align with the replacement of RFC2616 currently in the specification with the newer RFC7230, RFC7231, RFC7232, RFC7233, 7234 and RFC7235
* to make the section easier to read and apply

Is the Replacement the correctly proposal ?

### Current TMF630 text

The negotiation of the media type required by the client and provided by the server requires a number of headers in both the request and the response.

* The client MAY include ANY HTTP Header as specified in [RFC2616].
* The client MUST expect any HTTP header as specified in [RFC2616].

The media type is the form of the response payload that the client would like to receive; this is specified using the standard HTTP header: Accept

* The client MUST use the Accept HTTP header to specify the media type.
* The Accept HTTP header MUST have a value matching [RFC2616].
* The client MUST support “application/json” by default.

If a media type is specified that is not supported by the server then the response must be returned using the default, application/json, i.e. as if no media type was specified.

* The client MAY specify the locale of the response in the request header.
* The client MUST use the Accept-Language HTTP header to specify the locale.
* The Accept-Language http header MUST have a value matching the template:
  + Locale: ISO-639, ("\_", ISO3166-Alpha2)?
* The client MUST expect “en\_GB” by default.

The Accept-Language header uses the [ISO639] language code and [ISO3166] country code standards separated by an underscore “\_”, with the second half of the format being optional. If a language is specified that is not supported by the server then the response payload must be returned as “en\_GB”.

### Proposed text

All header values MUST follow the syntax rules set forth in the specification where the header field is defined. Many HTTP headers are defined in [RFC 7231](https://tools.ietf.org/html/rfc7231), however a complete list of approved headers can be found in the "[IANA Header Registry](http://www.iana.org/assignments/message-headers/message-headers.xhtml)".

The table of request headers below SHOULD be used by services. Using these headers is not mandated, but if used they MUST be used consistently.

|  |  |
| --- | --- |
| Header | Description |
| Authorization | Authorization header for the request |
| Date | Timestamp of the request, based on the client's clock, in [RFC 5322](https://tools.ietf.org/html/rfc5322) date and time format. The server SHOULD NOT make any assumptions about the accuracy of the client's clock. This header MAY be included in the request, but MUST be in this format when supplied.  Greenwich Mean Time (GMT) MUST be used as the time zone reference for this header when it is provided. For example: Wed, 24 Aug 2016 18:41:30 GMT. Note that GMT is exactly equal to UTC (Coordinated Universal Time) for this purpose. |
| Accept | The requested content type for the response such as:   * application/json * application/xml * text/xml * text/javascript (for JSONP)   [Per the HTTP guidelines](https://tools.ietf.org/html/rfc2616#section-14.1), this is just a hint and responses MAY have a different content type, such as a blob fetch where a successful response will just be the blob stream as the payload. If not set by the client, the server SHOULD default to application/json.  Client and server MUST support application/json by default. |
| Accept-Charset | Default is UTF-8, but services SHOULD be able to handle ISO 8859 1. |
| Accept-Language | This header MAY be used, but if set, it MUST be compliant to [RFC 5646](https://tools.ietf.org/html/rfc5646) (please note the TMF guideline is slightly incorrect in this regard) and have a value matching the template  [?](https://mywiki.telekom.de/pages/viewpage.action?pageId=930329062)   |  | | --- | | Locale: <ISO-639-1>[-<[ISO3166-Alpha2>] |   See also [ISO-639-1](https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes) and [ISO3166-Alpha2](https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2). Examples: de-DE, en-US.  The client MUST expect "en-GB" by default (according to TMF). |
| *Cache control headers* | Cache control headers SHOULD be implemented for resource underlying slow or no change at all or requiring caching for other reasons. This enabled the HTTP infrastructure, including the API manager, to cache request results. See [HTTP Caching](https://mywiki.telekom.de/pages/viewpage.action?pageId=930329062#BasicGuidelines-HTTP_Caching). |
| Content-Type | The MIME type of request body MUST be defined by the client if a body is sent with the request (for POST/PUT/ PATCH). It MUST comply to [RFC 2046](https://tools.ietf.org/html/rfc2046) and contain the charset used to encode the body. Example: application/json; charset=utf-8 |
| Prefer | If the return=minimal preference is specified, services SHOULD return an empty body in response to a successful insert or update. If return=representation is specified, services SHOULD return the created or updated resource in the response. Services SHOULD support this header if they have scenarios where clients would sometimes benefit from responses, but sometimes the response would impose too much of a hit on bandwidth. |
| X-Business-Context | The business context is a general classification for a larger number of requests (types) usually set by the originating consumer.  Any service receiving this header parameter MUST forward it unmodified to other services it invokes. Non-REST technologies (e.g. SOABP) might use a different identifier for this attribute. |
| X-Correlation-ID | Value for identifying a single transaction across multiple services up to the backend. Usually set by the originating consumer.  Any service receiving this header parameter MUST forward it unmodified to other services it invokes. Non-REST technologies (e.g. SOABP) might use a different identifier for this attribute.  If this ID is not available (not supplied, empty or at origin), the service MUST create it as a UUID. The length of the value MUST be between 36 and 256 characters (a standard UUID has 36 characters). The creator MAY prefix the UUID with an application and/or instance defined string. |
| X-Request-ID | Value to identify an individual request across consuming application, middleware and service provider. This ID is to be generated for every single request and not to be passed down (as X-Correlation-ID) to sub-services.  Every consuming application MUST create this value as UUID. The length MUST be between 36 and 256 characters (a standard UUID has 36 characters). The creator MAY prefix the UUID with an application and/or instance defined string. |

## Chapter 1, section “Request and Response HTTP headers – server side”

### Proposed change

Replace this section with the below text to:

* align with the replacement of RFC2616 currently in the specification with the newer RFC7230, RFC7231, RFC7232, RFC7233, 7234 and RFC7235
* to make the section easier to read and apply

### Current TMF630 text

As part of the content negotiation, just as the client can request particular behavior from the server, the server needs to respond back saying; “this is what you’re getting”.

* The server MAY include any HTTP header as specified in [RFC2616]
* The server MUST expect any HTTP header as specified in [RFC2616].
* The server SHOULD specify the media type of the response in the response header
* The server MUST use the Content-Type HTTP header to specify media type
* The Content–Type HTTP header MUST have a value matching [RFC2616].
* The server MUST support “application/json” by default.

The above rules work in the same way as the Accept; “application/json” is supported as the default media type of the server. This is returned either when requested or when no Accept header is present. The server SHOULD provide this header back to the client for the sake of clarity and so that the client does not have to “detect” the actual media type of the response. However, it is not required and the client SHOULD assume that either the requested media type or the default is being returned.

* The server SHOULD specify the Content-Length HTTP header. Client can thus know whether it has read the correct number of bytes from the connection and can make a HEAD request to find out how large the entitiy-body is, without downloading it.
* The server SHOULD use Last-Modified HTTP header in the responses to specify the time at which the resources where created.
* The server SHOULD use ETag in responses. The entity tag may be any string value, so long as it changes along with the resource’s representation.
* The server MUST use Location HTTP header to specify the URI of a newly created resource and may be used to direct clients to the operational status of an asynchronous controller resource. This is the preferred method to implement asynchronous behavior.

The following specifies the locale within the response; this matches the Accept-Language HTTP header in the request message.

* The server MAY specify the locale of the response in the response header.
* The server MUST use the Content-Language HTTP header to specify the locale.
* The Content-Language HTTP header MUST have a value matching the template:
  + Locale: ISO-639, ("\_", ISO3166-Alpha2)?
* The server MUST support “en\_GB” by default.
* The supported locales MUST be documented.
* The server SHOULD specify Content-Length HTTP header.

e.g.

HTTP/1.1 200 OK

Content-Type: application/json

Content-Language: en\_GB

Content-Length: 1024

ETag: “x234dff”

### Proposed text

All header values MUST follow the syntax rules set forth in the specification where the header field is defined. Many HTTP headers are defined in [RFC 7231](https://tools.ietf.org/html/rfc7231), however a complete list of approved headers can be found in the "[IANA Header Registry](http://www.iana.org/assignments/message-headers/message-headers.xhtml)".

Services SHOULD return the following response headers, except where noted in the "required" column.

| **Response Header** | **Required** | **Description** |
| --- | --- | --- |
| **Response Header** | **Required** | **Description** |
| Date | All responses | Timestamp the response was processed, based on the server's clock, in [RFC 5322](https://tools.ietf.org/html/rfc5322) date and time format. This header MUST be included in the response. Greenwich Mean Time (GMT) MUST be used as the time zone reference for this header. For example: Wed, 24 Aug 2016 18:41:30 GMT. Note that GMT is exactly equal to UTC (Coordinated Universal Time) for this purpose. |
| Cache control headers | GET method | If the server implements caching, it MUST specify cache control headers according to 3.8 HTTP Caching. |
| Content-Length | All responses | The server SHOULD specify the length |
| Content-Type | All responses | The content type of the response body MUST be specified if a body is returned. It MUST comply to [RFC 2046](https://tools.ietf.org/html/rfc2046) and contain the charset used to encode the body.  Both server and client MUST support application/json by default. |
| Preference-Applied | When specified in request | Whether a preference indicated in the Prefer request header was applied |

## Chapter 1, section “General HTTP headers”

### Proposed change

Add reference to RFC7234 as the most established HTTP caching standard and the message that caching should only be used after careful consideration whether it is beneficial for a certain resource or service.

## Chapter 3, section “API Response status and exception codes”

### Proposed change

Replace the current text in the section with the proposed below to limit the list of HTTP status codes to RFC7231 and RFC6585 and make the selection of the error codes to be used easier.

### Current TMF630 text

THE REST APIs MUST use the exception and response codes documented at <http://www.iana.org/assignments/http-status-codes/http-status-codes.xml>.

In particular, the following response codes must be used:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Status**  **Code** | **Rule** |  |
| **2xx** | **Success**  **Indicates that the client’s request was accepted successfully.** |
| 200 | OK - SHOULD be used to indicate nonspecific success. Must **not** be used to communicate errors in the response  bodyM |
| 201 | Created - MUST be used to indicate successful resource creation.  Return message SHOULD contain a resource representation and a  Location header with the created resource’s URI |
| 202 | Accepted - MUST be used to indicate successful start of an  asynchronous action |
| 204 | No Content - SHOULD be used when the response body is  intentionally empty |
| 206 | Partial Content – MUST be used for Partial resource returned in response (with pagination) |
| **3xx** | **Redirection**  **Indicates that the client must take some additional action in order to complete their request.** |
| 301 | Moved Permanently - SHOULD be used to relocate resources |
| 302 | Found - SHOULD not be used |
| 303 | See Other - SHOULD be used to refer the client to a different URI –  can be used with a Location header containing the URI of a resource that shows the outcome of an asynchronous task. |
| 304 | Not Modified - SHOULD be used to preserve bandwidth |
| 307 | Temporary Redirect - SHOULD be used to tell clients to resubmit the  request to another URI |
| **4xx** | **Client Error**  **This category of error status codes points the finger at clients.** |
| 400 | Bad Request - MAY be used to indicate nonspecific failure.  The request could not be understood by the server. The client  SHOULD NOT repeat the request without modifications |
| 401 | Unauthorized - MUST be used when there is a problem with the  client’s credentials |
| 403 | Forbidden - SHOULD be used to forbid access regardless of authorization state. *For example, a client may be authorized to*  *interact with some, but not all of a REST API’s resources. If the client attempts a resource interaction that is outside of its permitted*  *scope, the REST API should respond with 403.* |
| 404 | Not Found - MUST be used when a client’s provided URI cannot be  mapped to a resource URI |
| 405 | Method Not Allowed - MUST be used when the HTTP method is not  Supported |
|  | 406 | No Acceptable - The requested resource is capable of generating only content not acceptable according to the Accept headers sent in the request |  |
|  | 409 | Conflict -The request could not be completed due to a conflict with the current state of the target resource. |  |
|  | 410 | Gone - The requested resource is no longer available at the server and no forwarding address is known. |  |
|  | 411 | Length required-The server refuses to accept the request without a defined Content- Length. |  |
|  | 412 | Precondition Failed - The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server |  |
|  | 413 | Request Entity Too Large- The server is refusing to process a request because the request entity is larger than the server is willing or able to process |  |
|  | 414 | Request-URI Too Long- The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret |  |
|  | 415 | Unsupported Media Type - The request entity has a media type which the server or resource does not support. For example, the client uploads an image as image/svg+xml, but the server requires that images use a different format. |  |
|  | 422 | Unprocessable Entity - The request was well-formed but was unable to be followed due to semantic errors. |  |
|  | 429 | Too Many Requests – SHOULD be used to indicate that the client has sent to many requests in a given amount of time (“rate limiting”). |  |
| 5 | **5xx** | **Server Error**  **This category of error status codes points the finger at servers.** |  |
| 400 | 500 | Internal Server Error – a generic error message, given when an unexpected condition was encountered and no more specific message is suitable. |  |
|  | 501 | Not implemented- the server either does not recognize the request method, or it lacks the ability to fulfil the request. Usually this implies future availability (e.g., a new feature of a web-service API). |  |
| 503 | 503 | Service unavailable - The server is currently unavailable (because it is overloaded or down for maintenance). Generally, this is a temporary state. |  |

### Proposed text

The HTTP protocol has a range of predefined status and error codes. The codes generally have the following meaning:

* 2xx = Success
* 3xx = Retry/Redirect
* 4xx = Consumer-side Error
* 5xx = Server-side Error

A service MUST limit the use of HTTP status codes to the standards [RFC 7231](https://tools.ietf.org/html/rfc7231#section-6) and [RFC 6585](https://tools.ietf.org/html/rfc6585). These codes MUST be applied according to their intended uses.

The HTTP status and error codes SHOULD be used as detailed out in the chapters for method application (e.g. [Application of the POST method](https://mywiki.telekom.de/pages/viewpage.action?pageId=930329062#BasicGuidelines-Resource_Post)). In addition, the following statuses can be returned:

| **HTTP Status** | **Meaning** | **Methods** |
| --- | --- | --- |
| **HTTP Status** | **Meaning** | **Methods** |
| 200 OK | This is the standard success response. If a resource was created, use 201 instead. | GET, PUT, PATCH, DELETE |
| 202 Accepted | The request was successful and will be processed asynchronously. | POST, PUT, PATCH, DELETE |
| 301 Moved Permanently | This and all future requests should be directed to the given URI. | All |
| 303 See Other | Should be used with a Location header to refer the client to a different URI, e.g. to query the outcome of an asynchronous operation. | POST, PUT, PATCH, DELETE |
| 400 Bad Request | Generic / unknown error. Should also be delivered in case of input payload fails business logic validation. | All |
| 401 Unauthorized | The user must log in (this usually means "Unauthenticated"). | All |
| 403 Forbidden | The user is authenticated, but not authorized to use this resource. | All |
| 404 Not Found | The resource was not found. | GET, PUT, PATCH, DELETE |
| 406 Not Acceptable | Resource can only generate content not acceptable according to the Accept headers sent in the request. | All |
| 408 Request Timeout | The server times out waiting for the resource. A retry might be possible (SHOULD be indicated in the OpenAPI Spec). | All |
| 410 Gone | Resource does not exist any longer, e.g. when accessing a resource that has intentionally been deleted. | GET, PUT, PATCH, DELETE |
| 411 Length Required | The server refuses a request without the Content-Length header set | All |
| 412 Precondition Failed | Returned for conditional requests, e.g. If-Match if the condition failed. Used for optimistic locking. | PUT, PATCH, DELETE |
| 414 URI Too Long | A server that receives a request-target longer than any URI it wishes to parse MUST respond with status 414. | All |
| 415 Unsupported Media Type | Clients sent request body without content type or specified a content type that cannot be processed by the service. | All |
| 429 Too Many Requests | SHOULD be used to indicate that the client has sent to many requests in a given amount of time. See [Rate limits](https://mywiki.telekom.de/pages/viewpage.action?pageId=930329062#BasicGuidelines-Rate_Limits). |  |
| 500 Internal Server Error | A generic error indication for an unexpected server execution problem (here, client retry may be sensible). | All |
| 501 Not Implemented | Server cannot fulfill the request (usually implies future availability, e.g. new feature). | All |
| 503 Service Unavailable | Server is (temporarily) not available (e.g. due to overload)  —  client retry may be sensible. | All |

**Application of the POST method**

The POST method is usually used to create a single new entity. The service SHOULD return one of the following possible responses:

|  |  |
| --- | --- |
| **HTTP Status** | **Details** |
| 201 Created | On success, the HTTP status code 201 "Created" MUST be returned. It is RECOMMENDED to return the full data structure of the created entity (including attributes determined by the service, such as timestamps or calculated properties) in the body, but the service also MAY return an empty body. The location of the resource MUST be returned as HTTP location header – it MAY be returned as well in the body (in this case [HAL](http://stateless.co/hal_specification.html) is RECOMMENDED). |
| 409 Conflict | The request cannot be completed due to conflict, e.g. when two clients try to create the same resource or if a duplicate check has failed. |
| *In addition, the generic statuses listed in "*[*HTTP status and error codes*](https://mywiki.telekom.de/pages/viewpage.action?pageId=930329062#BasicGuidelines-HTTP_Status_and_Error_Codes)*" can be returned* | |

**Application of the GET method**

Get usually is used to either retrieve a list of resources (e.g. GET …/customers) or a specific entity (e.g. GET …/customers/12). The service SHOULD return one of the following possible responses:

|  |  |
| --- | --- |
| **HTTP Status** | **Details** |
| 200 OK | The GET was successful. Data provided in response body. |
| 304 Not modified | If the client has used the If-Modified-Since header, the service MAY respond with "304 Not modified". See [HTTP Caching](https://mywiki.telekom.de/pages/viewpage.action?pageId=930329062#BasicGuidelines-HTTP_Caching). |
| *In addition, the generic statuses listed in "*[*HTTP status and error codes*](https://mywiki.telekom.de/pages/viewpage.action?pageId=930329062#BasicGuidelines-HTTP_Status_and_Error_Codes)*" can be returned* | |

**Application of the PUT and PATCH methods**

**PUT:** The PUT method is usually used to update an existing resource. When accessing the PUT method, the entire data of the respective resource SHOULD be transferred even if the client/consumer only wants to change part of the data.

**PUT/create:** If the resource identifier is determinable by the client before creation, the PUT method MAY be used to create a resource. In this case, on success, the HTTP status code 201 "Created" MUST be returned. A service SHOULD NOT offer both POST and PUT to create a resource. Usually, POST if preferable.

**PATCH:** The PATCH method MAY be provided by a service to facilitate a partial update without submitting the full data structure of the record. Implementation of the PATCH method SHOULD be based on either

* The application/json-patch+query media type as defined in [RFC 7386](https://tools.ietf.org/html/rfc7386) (simpler, but has limitations)
* The application/json-patch+json media type as defined in [RFC 6902](https://tools.ietf.org/html/rfc6902) (more powerful, but also more complex)

**Returned body:** If the update or creation was successful, it is RECOMMENDED to return the full data structure of the created record (including attributes determined by the service, such as timestamps or calculated attributes) in the body, but the service also MAY return an empty body. The location of the resource MUST be returned as HTTP location header – it MAY be returned as well in the body as href attribute. The service SHOULD return one of the following possible responses:

|  |  |
| --- | --- |
| **HTTP Status** | **Details** |
| 200 OK | The update was successful. |
| 201 Created | Only to be used in exceptional cases for PUT/create, see above. |
| 409 Conflict | The request cannot be completed due to conflict, e.g. when two clients try to update the same resource. |
| *In addition, the generic statuses listed in "*[*HTTP status and error codes*](https://mywiki.telekom.de/pages/viewpage.action?pageId=930329062#BasicGuidelines-HTTP_Status_and_Error_Codes)*" can be returned* | |

**Application of the DELETE method**

The DELETE method is usually provided to remove a resource from a collection. The service SHOULD return one of the following possible responses:

|  |  |
| --- | --- |
| **HTTP Status** | **Details** |
| 204 No content | This SHOULD be the default response. The DELETE was successful. Response body is empty. |
| 200 OK | The DELETE was successful. Response body contains data of the deleted resource. |
| *In addition, the generic statuses listed in "*[*HTTP status and error codes*](https://mywiki.telekom.de/pages/viewpage.action?pageId=930329062#BasicGuidelines-HTTP_Status_and_Error_Codes)*" can be returned* | |

## Chapter 3, section “API Media types”

### Proposed change

Replace the current text with the below to reference RFC2046 and RFC4288 instead of RFC7386

### Current TMF630 text

When defining methods for REST services, we can further specify the types of data a given method can process. For example, a GET method may be able to transfer a Trouble Ticket representation in XML or JSON. Each is represented by its own media type.

* REST APIs MUST support the “application/json” media type by default.
* In case of PATCH if application/json is provided then the default rule will be to apply the same rules as for JSON merge.
* In case of JSON Patch [RFC6902] for partial updates “application/json-patch+json” media type MUST be used.
* In case of JSON PATCH Query, “application/json-patch+query” media type MUST be used.
* In case of PATCH as per https://tools.ietf.org/html/rfc7386 “application/merge-patch+json” MUST be used
* The default for resource representation MUST be JSON.
* An API MUST only use the ACCEPT HEADER and CONTENT-TYPE (POST) to control the representation media types. Other mechanisms SHOULD not be supported.

### Proposed text

The representation of resources that are exchanged when URIs are accessed can have different formats. The consumer of a service can specify which types of content it accepts in its request via the header parameter "Accept." Responses of the provider but also requests that transport content (such as PUT and POST) indicate in which representation the content is actually transferred using the "Content-Type" parameter.

1. Any service MUST support application/json (both for requests and responses). If additional media types are offered, they MUST be documented in the OpenAPI Specification of the API.
2. Media types MUST always be coded in accordance with [RFC 2046](https://www.ietf.org/rfc/rfc2046.txt).
3. For derived, user-defined media types, the prefix vnd vnd  = stands for vendor MUST be used in accordance with [RFC 4288](https://tools.ietf.org/html/rfc4288) in the following form:

|  |
| --- |
| application/vnd.<companyname>.<mediatypename>\[.<versionnumber>\]+json and  application/vnd.<companyname>.< mediatypename >\[.<versionnumber>\]+xml  Examples: application/vnd.dtag.customer.v1+jsonapplication/vnd.dtag.serviceorder.v3+json |

If these are provided, application/json should still be accepted and mapped to the latest backwards compatible version.

## Chapter 3, section User, Application and Extended Error Codes

### Proposed change

Replace current text with the below text to align with RFC7807 instead of TMF defined error response

### Current TMF630 text

The HTTP 4xx or 5xx standard error codes should always be used in the response header.

An API MAY support user and application specific error codes.

User and Application specific Error Codes MUST be represented in the Error Representation (Body) of the response.

Sub codes are possible (400-2) however they MUST be in the Error Representation (Body)

Error Representation in body of the response MUST have the following structure:

|  |  |  |
| --- | --- | --- |
| Property | Description | Mandatory |
| code | Application related code (as defined in the API or from a common list) | Mandatory |
| reason | Text that explains the reason for error. This can be shown to a client user. | Mandatory |
| message | Text that provide more details and corrective actions related to the error. This can be shown to a client user. | Optional |
| status | http error code extension like 400-2 | Optional |
| referenceError | url pointing to documentation describing the error | Optional |
| @type | The class type of a REST resource. | Optional |
| @schemaLocation | it provides a link to the schema describing a REST resource. | Optional |

Example:

|  |
| --- |
| 400 Bad Request  Content-type:application/json  {  "code": "ERR001",  "reason": "Missing mandatory field",  "message": "Please provide and Authorisation header"  } |

The error codes if supported SHOULD be defined in the API specification.

@type and @schemaLocation SHOULD be used for run time extension of the error.

### Proposed text

[RFC 7807](http://tools.ietf.org/html/rfc7807) defines a Problem JSON object and the media type **application/problem+json**. Operations should return it (together with a suitable status code) when any problem occurred during processing and you can give more details than the status code itself can supply, whether it be caused by the client or the server (i.e. both for 4xx or 5xx error codes).

A service returning an error status code MUST NOT return an HTML representation of the error in the body.

## Add section “Client Guidance”

### Proposed change

Add a section describing the patterns which are describing how a supplier/client should treat the information shared in the API to ensure upgradability and independence.

### Proposed text

This chapter is specific for consumers of services and applies to systems making an originating call (any kind of front-end), but also to services consuming other services.  
To ensure the best possible experience for clients talking to a REST service, clients SHOULD adhere to the following best practices.

**Ignore rule**

For loosely coupled clients where the exact shape of the data is not known before the call, if the server returns something the client wasn't expecting, the client MUST safely ignore it.  
Some services MAY add fields to responses without changing versions numbers (see also [Versioning](https://mywiki.telekom.de/pages/viewpage.action?pageId=930329887#Guidelinesforserviceimplementation-Versioning)). Services that do so MUST make this clear in their documentation and clients MUST ignore unknown fields.

**Variable order rule**

Clients MUST NOT rely on the order in which data appears in JSON service responses. For example, clients SHOULD be resilient to the reordering of fields within a JSON object. When supported by the service, clients MAY request that data be returned in a specific order. For example, services MAY support the use of the \_orderBy query string parameter to specify the order of elements within a JSON array. Services MAY also explicitly specify the ordering of some elements as part of the service contract. For example, a service MAY always return a JSON object's "type" information as the first field in an object to simplify response parsing on the client. Clients MAY rely on ordering behavior explicitly identified by the service.

**Silent fail rule**

Clients requesting OPTIONAL server functionality (such as optional headers) MUST be resilient to the server ignoring that particular functionality.

**Failure handling**

Failure has to be expected. Depending on services means these can fail at any time, either temporarily or fatally.  
The following guidelines for failure handling apply:

* Consumers of services MUST implement timeouts for every service invocation. The recommended length of the timeout SHOULD be provided in the services documentation as part of their OpenAPI Specification.
* Consumers of service SHOULD log error bodies returned by services to enable later analysis (see also [Error body](https://mywiki.telekom.de/pages/viewpage.action?pageId=930329062#BasicGuidelines-Error_body)).
* On redirects (usually indicated by 3xx HTTP statuses), the consumer SHOULD retry immediately with the URI specified by the service and potentially (e.g. on 301 "Moved permanently") update its reference(s).
* On client errors (usually indicated by 4xx HTTP statuses), the service SHOULD log sufficient information for the developer to analyze and fix the problem. Log contents might be selected based on the error code, e.g. on 414 "URI too long", only the request URL could be logged.
* On temporary failures (usually indicated by the 5xx HTTP statuses), the consumer SHOULD retry the operation, either immediately or after a short waiting time. Services SHOULD include recommendations for retry handling as part of their OpenAPI Specification.

**Circuit Breaking**

Usually, a [circuit breaker](https://en.wikipedia.org/wiki/Circuit_breaker_design_pattern) scheme is implement as part of the middleware (e.g. the API Gateway). This has the advantage that important parameters, like timeout, are controlled by the provider.  
A consumer MAY implement its own circuit breaking mechanism (for example, see [Hystrix](https://github.com/Netflix/Hystrix/wiki)), but MUST do this in close cooperation with each individual service provider, to prevent undesired effects.