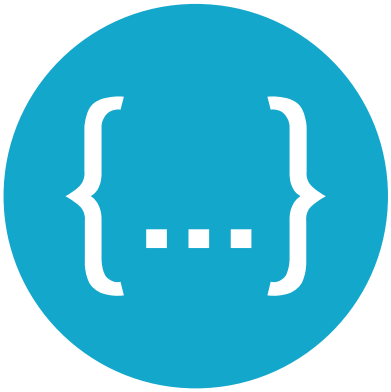
Draft

Volkswagen Infotainment Web Interface

viwi - protocol definition



version: 1.6.0
author: Dr. Patrick Bartsch < <patrick.bartsch@volkswagen.de>
release: 2016/09/19

Table of contents

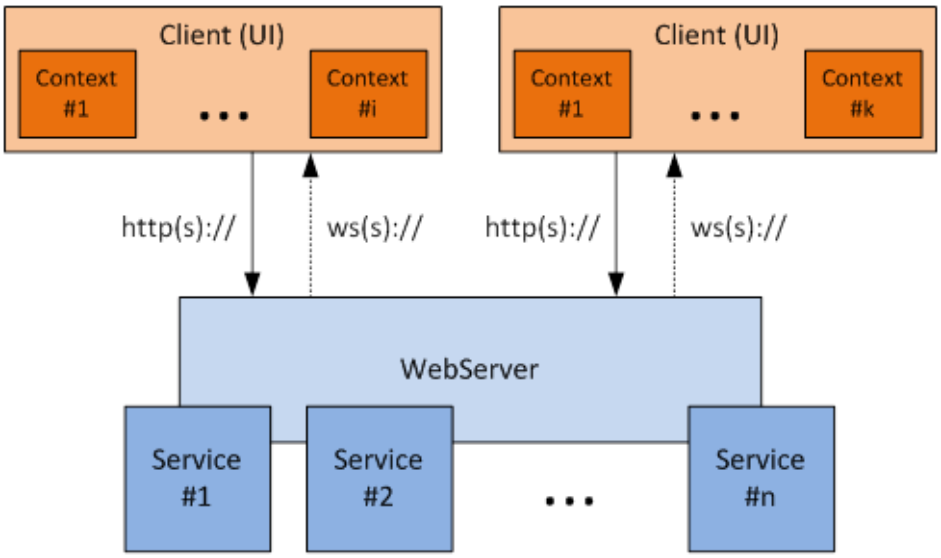
Draft

1 Introduction

Draft

1.1 Purpose of this document

This document provides an overview over web services on the WebInfotainment platform. All services provide mainly HTTP_GET, HTTP_POST, HTTP_PUT, HTTP_DELETE (herein called GET, POST, PUT, DELETE) and WebSocket interfaces. The response is always of contentType: `application/json` (or `application/vnd.viwi.v<major>.<minor>.<patch>+json` for a dedicated version), this also applies for the WebSocket interface. The general approach is a RESTful API.



The general architecture of a system using the herein defined interfaces is split into WebServer and Client (Figure 1: Example of system architecture). While the any client can have different contexts, the web server can have different services providing a number of resources that hold a number of elements. The connection between client and server is always a TCP/IP connection that allows http(s):// and ws(s):// protocols on top.

1.2 Paradigms

The API described herein follows some general 'design rules' or 'paradigms'. Deviation of actual interface definition must be minimized in a reasonable way.

1. The depth of the URI tree is limited to 3, i.e. `/<service>/<resource>/<element>`. While `<service>` and `<resource>` are predefined in this document, `<element>` will always be an identifier or the keyword 'spec'. `<resource>` is always a collection (e.g stations) and thus has to be a noun plural form, `<resource>` identifiers must not carry verbs like `getAccount` or `createTrack`. This applies also to `<resource>` that provide only a single element.
2. An event is considered 'on change' if two subsequent GET requests on the corresponding URI would result in different responses (only the response objects own properties are relevant). If an elements property 'name' is changed, an 'on change' event is fired (Publish Subscribe). If an element is added or removed from a list of objects (on resource level), an 'on change' event is fired (Publish Subscribe). If a client wants to get notified on nested properties or structures, the corresponding event has to be subscribed individually.
3. To keep the overall network traffic at a minimum, every resource and element access supports filtering. The response filtering concepts "fields", "paging" and "expand" are generally available.
4. Every object inherits from XObject. I.e. every Object has three mandatory properties: id, uri and name. These can not be filtered and will be present in every response object.
    1. Every type of XObject has its own endpoint.
    2. An XObject consists only of primitives and references to other XObject or lists of those.
5. RESTful HTTP calls are the main interface. A client does not necessarily have to register for events, it can also

use plain HTTP polling. A polling client will not be automatically updated with the server state. API-Calls that change the server state are responded with a StatusObject only. The altered state has to be accessed via HTTP_GET or by event subscription.

1.3 HTTP status codes

All web services have to follow the W3C/IANA HTTP status code specification (HTTP/1.1 status codes, 2012, RFC2616).

This section extends each Status-Code is extended with implication and handling information for a client. The general meaning of status codes remains untouched. The client has to be to work with any response with a status code defined in RFC2616. In addition to the general RFC2616 status code definitions, some status codes are generally not expected by the client, those codes are marked as *not applicable* in viwi context. Each status code may have a domain specific meaning, which will be described in the according domain section separately. Please note that each service domain may overwrite the **implication** and **client-side treatment**.

Code	name	Implication	Client-side treatment
100	Continue	Not applicable.	None
101	Switching Protocols	Only used for establishing websockets, see RFC6455 section 4.2.2.	None
200	OK	Used for successful HTTP Requests. This code is used to acknowledge successful change of the resource or element.	None
201	Created	Used for successfully creating new entities.	None
202	Accepted	The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon. There is no guarantee that the request will be fullfilled.	Subscribe to affected entity, if the actual outcome is of interest.
203	Non-Authoritative Information	Not applicable.	None
204	No Content	Not applicable.	None
205	Reset Content	Not applicable.	None
206	Partial Content	Not applicable.	None
300	Multiple Choices	Used if multiple services match the request criteria (service registry only)	Client has to select and re-request

Code	name	Implication	Client-side treatment
301	Moved Permanently	Not applicable, see service registry .	None
302	Found	Not applicable.	None
303	See Other	Not applicable.	None
304	Not Modified	If the entity is not modified, this status code can be sent. This indicates that the client should look for this entity in its cache.	None
305	Use Proxy	Not applicable.	None
306	Unused	Not applicable, see status code 306.	None
307	Temporary Redirect	The requested resource has moved. The new location is send as an absolute URL in the response HTTP-Header `Location` field.	Client should submit a new HTTP request.
400	Bad Request	The client submitted a malformed request, repeating the request will not help.	Client has to make sure to send a valid request.
401	Unauthorized	This request requires authentication.	If client is not authenticated, client has to authenticate.
402	Payment Required	Not applicable.	None
403	Forbidden	The client has insufficient rights to obtain the requested information or has submitted an HTTP request with POST method to a property, which is `read-only` or not allowed to be set by the client(e.g. `POST` on `/car/info` property `vehicleIdentication`).	If client do not have access rights, client has to acquire access rights.
404	Not Found	The service cannot find the entity, this may be a permanent or temporary condition. Do not expect elements to be generally available. This status code is treated as a valid response. **Use-case specific error handling is required.**	

Code	name	Implication	Client-side treatment
405	Method Not Allowed	Not applicable, because all possible HTTP-Methods are defined by the the viwi document and missing privileges are singnaled by status code 403.	None
406	Not Acceptable	Not applicable.	None
407	Proxy Authentication Required	Not applicable.	None
408	Request Time-out	Not applicable. The request sent to the server took longer than the server was prepared to wait. In other words, client connection with server "timed out". Server configuration should rule out this error.	None
409	Conflict	It is not possible to establish the required state. This might be the case if a client wants to set an objects property that is not writable or that currently can not be set. The `message` property of StatusObject may have information for the client to recognize the source of the conflict. This information is only for debugging and development purposes.	Resource specific treatment is necessary (e.g. media audioSource might be disabled during navigation announcements, the clients tries to un-mute media). See status code 409 for more information.
410	Gone	Not applicable.	None
411	Length Required	Not applicable, see status code 411.	None
412	Precondition Failed	Not applicable.	None
413	Request Entity Too Large	Not applicable, see paging.	None
414	Request-URI Too Large	Not applicable, see RFC2616 section 3.2.1.	None
415	Unsupported Media Type	Not applicable.	None

Code	name	Implication	Client-side treatment
416	Requested range not satisfiable	Not applicable.	None
417	Expectation Failed	Not applicable.	None
500	Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.	There is no client-side fix/solution for this kind of errors.
501	Not Implemented	The server does not support the functionality required to fulfill the request. Unless specified otherwise in service specific status code handling, this error indicates a not-permitted error.	Not applicable.
502	Bad Gateway	Not applicable.	None
503	Service Unavailable	Service is currently unable to handle the request. This implies a temporary problem which will be solved after a given delay. If known, the delay **may** be indicated in a `Retry-After` header.	Client should retry to submit the request after delay.
504	Gateway Time-out	The request can not be fulfilled because server acts as a proxy and did not receive a timely response from remote component (e.g. ECU of driver assistance).	There is no client-side fix/solution for this kind of error. If the remote component expected to be reachable after a delay in this power-cycle then server sends a response with status code 503.
505	HTTP Version not supported	Not applicable.	None

1.3.1 Handling service responses with respecting status codes

To communicate with the services a client submits an HTTP request and waits for the response. A response is always sent with a status code. The service uses the pre-defined status codes when sending a response. Each client interpret the response with the respective status code.

Each status code is categorized as successful (2xx), redirection (3xx), client error (4xx) or server error (5xx). If status code is not successful, then the response must be handled by the client as following:

- If an error can be treated by a client, it is categorized as **permitted errors**. For each permitted error a treatment is defined.
- If an error can not be treated by the client, it is categorized as **not permitted error**.
- Each service domain may overwrite **permitted** and **not permitted** errors.

1.4 RESTful API

1.4.1 General information

The REST architectural style describes six constraints:

- uniform interface (resource based, self descriptive, ...)
- statelessness (necessary state to handle the request is contained within the request itself)
- cacheability (clients can cache responses)
- client-server architecture (uniform interface separates client and server)
- layered system (client cannot ordinarily tell whether it is connected directly or intermediary)
- code on demand optional (Servers are able to temporarily extend or customize the functionality of a client by transferring logic to it)

Conforming to the REST architectural style, will enable any kind of distributed system to have desirable emergent properties, such as performance, scalability, simplicity, modifiability, visibility, portability and reliability.

These constraints, applied to the architecture, were originally communicated by Roy Fielding in his doctoral dissertation and define the basis of RESTful-style.

Source: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.html

Uniform interface

The uniform interface constraint defines the interface between clients and servers. It simplifies and decouples the architecture, which enables each part to evolve independently. The four guiding principles of the uniform interface are:

Resource-based

Individual resources are identified in requests using URIs as resource identifiers. The resources themselves are conceptually separate from the representations that are returned to the client. For example, the server does not send its database, but rather, some HTML, XML or JSON that represents some database records expressed, for instance, in Finnish and encoded in UTF-8, depending on the details of the request and the server implementation.

Manipulation of resources through representations

When a client holds a representation of a resource, including any meta data attached, it has enough information to modify or delete the resource on the server, provided it has permission to do so.

Self-descriptive messages

Each message includes enough information to describe how to process the message. For example, which parser to invoke may be specified by an Internet media type (previously known as a MIME type). Responses also explicitly indicate their cache-ability.

Hypermedia as the engine of application state (HATEOAS)

Clients deliver state via body contents, query-string parameters, request headers and the requested URI (the resource name). Services deliver state to clients via body content, response codes, and response headers. This is technically referred-to as hypermedia (or hyperlinks within hypertext). Aside from the description above, HATEOS also means that, where necessary, links are contained in the returned body (or headers) to supply the URI for retrieval of the object itself or related objects. We'll talk about this in more detail later.

The uniform interface that any REST services must provide is fundamental to its design.

Statelessness

As REST is an acronym for REpresentational State Transfer, statelessness is key. Essentially, what this means is that the necessary state to handle the request is contained within the request itself, whether as part of the URI, query-string parameters, body, or headers. The URI uniquely identifies the resource and the body contains the state (or state change) of that resource. Then after the server does it's processing, the appropriate state, or the piece(s) of state that matter, are communicated back to the client via headers, status and response body. Most of us who have been in the industry for a while are accustomed to programming within a container which provides us with the concept of "session" which maintains state across multiple HTTP requests. In REST, the client must include all information for the server to fulfill the request, resending state as necessary if that state must span multiple requests. Statelessness

enables greater scalability since the server does not have to maintain, update or communicate that session state. Additionally, load balancers don't have to worry about session affinity for stateless systems. So what's the difference between state and a resource? State, or application state, is that which the server cares about to fulfill a request-data necessary for the current session or request. A resource, or resource state, is the data that defines the resource representation — the data stored in the database, for instance. Consider application state to be data that could vary by client, and per request. Resource state, on the other hand, is constant across every client who requests it. Ever had back-button issues with a web application where it went AWOL at a certain point because it expected you to do things in a certain order? That's because it violated the statelessness principle. There are cases that don't honor the statelessness principle, such as three-legged OAuth, API call rate limiting, etc. However, make every effort to ensure that application state doesn't span multiple requests of your service(s).

Cacheability

As on the World Wide Web, clients can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable, or not, to prevent clients reusing stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client-server interactions, further improving scalability and performance.

Client-server architecture

The uniform interface separates clients from servers. This separation of concerns means that, for example, clients are not concerned with data storage, which remains internal to each server, so that the portability of client code is improved. Servers are not concerned with the user interface or user state, so that servers can be simpler and more scalable. Servers and clients may also be replaced and developed independently, as long as the interface is not altered.

Layered system

A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load-balancing and by providing shared caches. Layers may also enforce security policies.

Code-on-demand optional

Servers are able to temporarily extend or customize the functionality of a client by transferring logic to it that it can execute. Examples of this may include compiled components such as Java applets and client-side scripts such as JavaScript.

1.4.2 Application to viwi

The interfaces described inhere follow the RESTful (REST: Representational State Transfer) principles. The main concept in REST is the existence of resources (sources of specific information or services), each of which is referenced with a uri as global identifier. The RESTful API is used to retrieve information for the client (request) from the server (response), while Events provide a channel for communication in server to client direction (push information). The supported HTTP request methods are GET, POST, PUT and DELETE. The following table explains the main principle of interface definition for all services defined hereafter:

	HTTP	HTTP	HTTP	HTTP	WebSocket
type	*GET*	*POST*	*PUT*	*DELETE*	*subscribe*

	HTTP	HTTP	HTTP	HTTP	WebSocket
root URI, such as `/` or `/api/v1/`	**List** the URIs and perhaps other details of the service known to the system.	n/a	**Register new service** with the system. The new service's URI is assigned automatically and is returned by the operation.	n/a	Get updated on **collection changes** (e.g. add, delete of elements)
service URI, such as `/<service>/`	**List** the URIs and perhaps other details of the service resources.	**Create new resource** within the service. The new resource's URI is assigned automatically and is returned by the operation.	n/a	**De-Register new service** with the system. `<service>` has to be accessed by its uuid. (special permissions needed)	Get updated on **collection changes** (e.g. add, delete of elements)
Resource URI, such as `/<service>/<resource>/`	**List** the URIs and perhaps other details of the collection's members.	**Create new element** in collection. The new element's URI is assigned automatically and is returned by the operation.	n/a	n/a	Get updated on **collection changes** (e.g. add, delete of elements)
Element URI, such as `/<service>/<resource>/<element>`	**Retrieve** a representation of the addressed **member** of the collection, expressed in an appropriate Internet media type.	**Update element**. The updated element's URI is assigned automatically and is returned by the operation.	**Create new element** in collection by sending the entire entity. The new element's URI is assigned accordingly.	**Delete** the referred entity of the collection (or its attributes)	Get updated on **element changes** (e.g. playing tracks offset on media player)

HEAD

The `HEAD` method is identical to `GET` except that the server **MUST NOT** return a message-body in the response. The meta information contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining meta information about the entity implied by the request without transferring the entity-body itself. The method is often used to obtain information about expiry or existence, specially in cases where cross origin resource sharing (CORS) is needed.

DELETE

A client can either delete entire `<element>`s or just properties on the `<element>`s properties specified in the `$fields` query parameters. The mandatory fields `id`, `name` and `uri` can not be deleted. In case of a writable `name` property on an `<element>`, a client can reset the name to `""` by POSTing.

Example element deletion

request:

```
DELETE medialibrary/tracks/01ACEB4B-002D-4060-A8EB-81868BF0BC37 HTTP/1.1
Host: 127.0.0.1:1337
Connection: keep-alive
Accept: application/json
User-Agent: Chrome/34.0.1847.137 Safari/537.36
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8,de;q=0.6
```

response:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Vary: Accept-Encoding
Content-Type: application/json; charset=utf-8
ETag: "-32550834"
Content-Encoding: gzip
Date: Tue, 13 Jun 2014 19:47:27 GMT
Connection: keep-alive
Transfer-Encoding: chunked


{
  "status" : "ok"
}
```

Example deletion of element properties artists and rating

request:

```
DELETE medialibrary/tracks/01ACEB4B-002D-4060-A8EB-81868BF0BC37?$fields=artists,rating HTTP/1.1
Host: 127.0.0.1:1337
Connection: keep-alive
Accept: application/json
User-Agent: Chrome/34.0.1847.137 Safari/537.36
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8,de;q=0.6
```

response:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Vary: Accept-Encoding
Content-Type: application/json; charset=utf-8
ETag: "-32550834"
Content-Encoding: gzip
Date: Tue, 13 Jun 2014 19:47:27 GMT
Connection: keep-alive
Transfer-Encoding: chunked


{
  "status" : "ok"
}
```

POST

Creating or changing an entity should always be done in the most condensed way (one request) possible, except the client explicitly wants to execute a sequence.

Creating e.g. an element is possible by providing no additional information in some cases and later in time modifying the newly created element to the desired state is possible. Nevertheless, providing all known information with the initial request ensures object integrity and minimum network traffic at the same time.

Changing an element's properties subsequently may cause unwanted effects like flickering in the maprenderer case or jumps in the mediarenderer case.

The protocol is based the **last-wins** principal, i.e. last update arriving at the server determines the final state, disregarding the actual time gap between arrival of two subsequent requests.

1.4.3 Caching

Cache control is established by using HTTP headers `ETag` and `If-None-Match` (cmp. RFC 7232-2.3). Mainly used for web cache validation, a client can make conditional requests based on the response. This allows caches to be more efficient, saves bandwidth and avoid cache-sync problems as a service does not need to send a full response if the content has not changed and a client can use the cached information. This also applies to service-service communication, especially when service `A` references element owned by service `B`. `A` will act as a client and use its own cached information if no changes where detected by `B`.

The basic sequence is:

1. ask for an entity request:

```
GET cdn/images/foo.png HTTP/1.1
Host: 127.0.0.1:1337
Connection: keep-alive
Accept: image/*
User-Agent: Chrome/34.0.1847.137 Safari/537.36
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8,de;q=0.6
```

response:

```
HTTP/1.1 200 OK
Content-Type: image/png
ETag: "641abf"
Content-Encoding: gzip
Date: Tue, 13 Jun 2014 19:47:27 GMT
Connection: keep-alive
Transfer-Encoding: chunked
Content-Length: 15360


<base64encodedfile>
```

The client uses the content delivered with the response, stores the `ETag` info with the entity in its cache.

1. later ask for the same entity together with `If-None-Match`

```
GET cdn/images/foo.png HTTP/1.1
Host: 127.0.0.1:1337
Connection: keep-alive
If-None-Match: "641abf"
Accept: image/*
User-Agent: Chrome/34.0.1847.137 Safari/537.36
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8,de;q=0.6
```

response:

```
HTTP/1.1 304 Not Modified
Content-Type: image/png
ETag: "641abf"
Content-Encoding: gzip
Date: Tue, 13 Jun 2014 21:47:27 GMT
Connection: keep-alive
Transfer-Encoding: chunked
Content-Length: 0
```

The client uses the information from its cache, 0 bytes of payload were transfered.

1.5 AND vs. OR queries

Queries are treated as OR queries per query parameter, i.e. `GET /api/v1/<service>/<resource>/?name=foo,bar` will retrieve all elements whose names equals 'foo' **OR** 'bar'. Using multiple query parameters on the other hand is treated as an **AND** query, e.g. `GET /api/v1/<service>/<resource>/?name=foo&type=bar` retrieves all elements named 'foo', being of type 'bar'. To obtain elements by OR queries over multiple query parameters, multiple queries are needed to be made and being combined by the client. This also applies to `$expand`, `$fields` etc.
`GET /api/v1/<service>/<resource>/?$expand=foo,bar` reads like 'please expand the properties whose name is either foo **OR** bar'. `GET /api/v1/<service>/<resource>/?$expand=foo,bar` reads like 'give me only those properties whose name is either foo **OR** bar'.

1.6 Adressing aspects

1.6.1 unified resource identifier (`uri`)

The system uses `uri`s to reference to entities from and to each other. There are to types of `uri`s, absolute (e.g. `https://127.0.0.1:1337/cdn/images/image001.jpg`) and relative (e.g. `/cdn/images/image001.jpg`). All `uri`s are absolute by default, only if a service is referencing to information that available under the same host and port, the `uri` will be

relative. All clients have to send the HTTP `Host` header when accessing a service. The server will use the information provided by the HTTP `Host` header to build the absolute `uri`s. This means that an external client will get `uri`s based on the external IP address or hostname and port number of the service it is talking to.

1.7 Cross-origin resource sharing (CORS)

Cross-origin resource sharing (CORS) is a mechanism that allows restricted resources (e.g. fonts, JavaScript, etc.) on a web page to be requested from another domain outside the domain from which the resource originated.

A web page may freely embed images, stylesheets, scripts, iframes, videos and some plugin content from any other domain. However embedded web fonts and AJAX(XMLHttpRequest) requests have traditionally been limited to accessing the same domain as the parent web page (as per the same-origin security policy). "Cross-domain" AJAX requests are forbidden by default because of their ability to perform advanced requests (POST, PUT, DELETE and other types of HTTP requests, along with specifying custom HTTP headers) that introduce many cross-site scripting security issues.

CORS defines a way in which a browser and server can interact to safely determine whether or not to allow the cross-origin request. It allows for more freedom and functionality than purely same-origin requests, but is more secure than simply allowing all cross-origin requests. It is a recommended standard of the W3C.

Source: https://en.wikipedia.org/wiki/Cross-origin_resource_sharing Web Server requirements:

- The server shall be configured to allow CORS (http://www.w3.org/TR/cors/) during development phase
- The server shall send the CORS header Access-Control-Allow-Origin * during development phase
- The server shall send the CORS header Access-Control-Expose-Headers 'location' during development phase
- The server shall send the CORS header Access-Control-Allow-Credentials during development phase

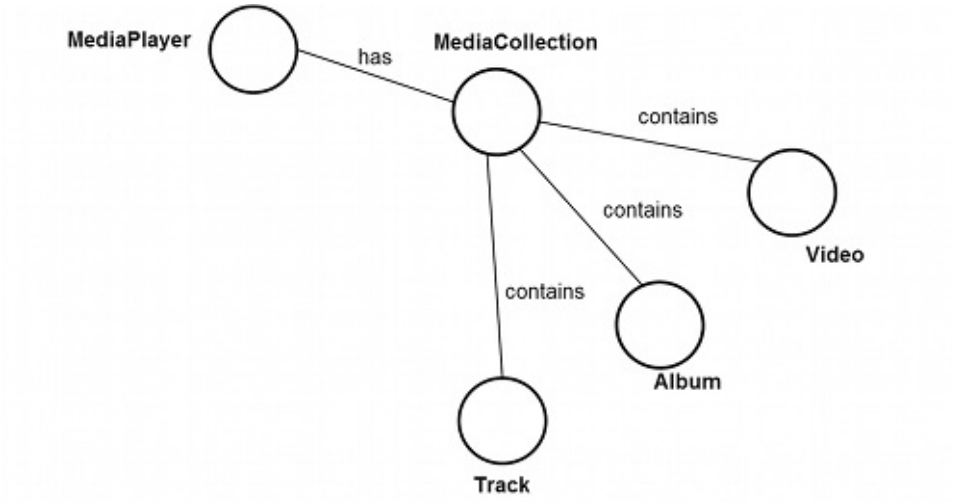
1.8 Request response content

The response is always of contentType: `application/json` (or `application/vnd.viwi.v<major>.<minor>.<patch>+json`) with `charset=UTF-8`, expect for binary content. Binary content uses regular HTTP headers according to its MIME-type.

POST requests only accept payload in the body, no form encoded (HEADER) data is accepted.

1.9 Graph interface

A fundamental concept of the API is the graph. All objects can get into relation with each other. The objects can be understood as nodes in a graph while the relations between them are the edges in this graph. This relation is expressed by referencing from one object to another in the object structure. For example, a 'media renderer' (node) may 'have' (edge) a 'mediaCollection' (node). The 'mediaCollection' (node) itself may 'contain' (edge) multiple items like a 'track' (node) or a 'video' (node).



1.10 Response filtering

1.10.1 Reserved query parameters

An API-query or subscription can contain none, one or multiple parameters that represent the desire to filter a query. All parameters that do not point to an object property, but have a general nature, are prefixed with $. The following paragraphs make use of this rule and give examples.

1.10.2 Resource search

A search can be performed on any resource, as either freetext search, parameter search or a combination of both. The parameter search is performed by using request parameters according to the properties of the resources object. The request parameters name has to resemble the objects property name, while its value is the search key. The character "%" (URL encoded: %25) is used as wildcard and can be used anywhere in the search key. It is possible to combine multiple request parameters in a single search, while combining multiple search keys in a single request is not supported. Thus, a search query is always returns the result of a `AND` query. `http://127.0.0.1:1337/medialibrary/tracks/?$q=5&artists=CB4E5462-1DDF-46C1-9B9D-45D6008A1989` reads as 'fetch all tracks from the medialibrary that have any property set to 5 AND the the artists property contains to CB4E5462-1DDF-46C1-9B9D-45D6008A1989'. To perform an `OR` query, multiple query have to be combined on client side.

Property search

request:

```
GET medialibrary/tracks/?rating=5 HTTP/1.1
Host: 127.0.0.1:1337
Connection: keep-alive
Accept: application/json
User-Agent: Chrome/34.0.1847.137 Safari/537.36
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8,de;q=0.6
```

response:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Vary: Accept-Encoding
Content-Type: application/json; charset=utf-8
ETag: "-32550834"
Content-Encoding: gzip
Date: Tue, 13 Jun 2014 19:47:27 GMT
Connection: keep-alive
Transfer-Encoding: chunked


{
  "status" : "ok",
  "data":[
    {
    "id" : "1ebe63c0-b528-11e3-a5e2-0800200c9a66",
    "name" : "me and my empty wallet",
    "duration":42,
    "artists":[
      {
        "id" : "bb3372f0-b527-11e3-a5e2-0800200c9a66",
        "name" : "ich",
        "uri" : "http://127.0.0.1:1337/medialibrary/artists/bb3372f0-b527-11e3-a5e2-0800200c9a66"
      }
    ],
    "albums":[
      {
        "id" : "5088aaa0-b528-11e3-a5e2-0800200c9a66",
        "name" : "where is my car",
        "uri" : "http://127.0.0.1:1337/medialibrary/albums/5088aaa0-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "image" : "http://127.0.0.1:1337/cdn/images/FT1hZIPBHX.png",
    "genres":[
      {
        "id" : "92884410-b528-11e3-a5e2-0800200c9a66",
        "name" : "Rock",
        "uri" : "http://127.0.0.1:1337/medialibrary/genres/92884410-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "rating":5,
    "uri" : "http://127.0.0.1:1337/medialibrary/tracks/1ebe63c0-b528-11e3-a5e2-0800200c9a66"
    }
  ]
}
```

A freetext search is performed by using the request parameter `$q`. The search key will be tested against any property value. Freetext search also supports wildcard character "%" (URL encoded: %25).

Example freetext

request:

```
GET medialibrary/tracks/?$q=me%20and%20my%20empty%20wallet HTTP/1.1
Host: 127.0.0.1:1337
Connection: keep-alive
Accept: application/json
User-Agent: Chrome/34.0.1847.137 Safari/537.36
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8,de;q=0.6
```

response:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Vary: Accept-Encoding
Content-Type: application/json; charset=utf-8
ETag: "-32550834"
Content-Encoding: gzip
Date: Tue, 13 Jun 2014 19:47:27 GMT
Connection: keep-alive
Transfer-Encoding: chunked


{
  "status" : "ok",
  "data":[
    {
    "id" : "1ebe63c0-b528-11e3-a5e2-0800200c9a66",
    "name" : "me and my empty wallet",
    "duration":42,
    "artists":[
      {
        "id" : "bb3372f0-b527-11e3-a5e2-0800200c9a66",
        "name" : "ich",
        "uri" : "http://127.0.0.1:1337/medialibrary/artists/bb3372f0-b527-11e3-a5e2-0800200c9a66"
      }
    ],
    "albums":[
      {
        "id" : "5088aaa0-b528-11e3-a5e2-0800200c9a66",
        "name" : "where is my car",
        "uri" : "http://127.0.0.1:1337/medialibrary/albums/5088aaa0-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "image" : "http://127.0.0.1:1337/cdn/images/FT1hZIPBHX.gif",
    "genres":[
      {
        "id" : "92884410-b528-11e3-a5e2-0800200c9a66",
        "name" : "Rock",
        "uri" : "http://127.0.0.1:1337/medialibrary/genres/92884410-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "rating" : "5",
    "uri" : "http://127.0.0.1:1337/medialibrary/tracks/1ebe63c0-b528-11e3-a5e2-0800200c9a66"
    }
  ]
}
```

Referenced object

To search for a linked object, use its `<uuid>` as search value on the search key of its property name.

request:


```
GET medialibrary/tracks/?artists=bb3372f0-b527-11e3-a5e2-0800200c9a66 HTTP/1.1
Host: 127.0.0.1:1337
Connection: keep-alive
Accept: application/json
User-Agent: Chrome/34.0.1847.137 Safari/537.36
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8,de;q=0.6
```

response:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Vary: Accept-Encoding
Content-Type: application/json; charset=utf-8
ETag: "-32550834"
Content-Encoding: gzip
Date: Tue, 13 Jun 2014 19:47:27 GMT
Connection: keep-alive
Transfer-Encoding: chunked


{
  "status" : "ok",
  "data":[
    {
    "id" : "1ebe63c0-b528-11e3-a5e2-0800200c9a66",
    "name" : "me and my empty wallet",
    "duration": 42,
    "artists": [
      {
        "id" : "bb3372f0-b527-11e3-a5e2-0800200c9a66",
        "name" : "ich",
        "uri" : "http://127.0.0.1:1337/medialibrary/artists/bb3372f0-b527-11e3-a5e2-0800200c9a66"
      }
    ],
    "albums": [
      {
        "id" : "5088aaa0-b528-11e3-a5e2-0800200c9a66",
        "name" : "where is my car",
        "uri" : "http://127.0.0.1:1337/medialibrary/albums/5088aaa0-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "image" : "http://127.0.0.1:1337/cdn/images/FT1hZIPBHX.png",
    "genres": [
      {
        "id" : "92884410-b528-11e3-a5e2-0800200c9a66",
        "name" : "Rock",
        "uri" : "http://127.0.0.1:1337/medialibrary/genres/92884410-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "rating": 5,
    "uri" : "http://127.0.0.1:1337/medialibrary/tracks/1ebe63c0-b528-11e3-a5e2-0800200c9a66"
    }
  ]
}
```

1.10.3 Fields

Every `GET` request is filterable by using the request parameter `$fields`. The parameter accepts a comma separated list of attribute names. For event subscriptions, the `$fields` parameter can be set as well. The fields defined by XObject are mandatory, i.e. these are always part of the response. If no 'field' parameter is given, the client gets an unfiltered response.

1.10.4 Sorting

Similar to searching, a generic parameter `$sortby` can be used for `GET` requests on resource level to describe sorting rules. Accommodate complex sorting requirements by letting the sort parameter take in a list of comma separated fields, each with a possible unary negative to imply descending sort order.

Every `<resource>` has a fixed default ordering, i.e. statelessness is given by defining a default sort behavior per `<resources>`, the `$sortby` parameter just over-rules the default behavior.

```
http://127.0.0.1:1337/medialibrary/tracks/?$sortby=rating
```

or

```
http://127.0.0.1:1337/medialibrary/tracks/?$sortby=-rating
```

Sorting can be combined with searching by adding both request parameters to the query:

```
http://127.0.0.1:1337/medialibrary/tracks/?$sortby=rating&name=me%20an%my%20empty%20wallet
```

Ordering by primitives

Ordering by primitive value properties is possible alphabetical an numerical.

Ordering by complex objects

The default ordering by references to complex object is defined by the mandatory `name` member of the complex object. A service might implement a different behavior that has to be specified per service, if different to the default.

Ordering by list properties

Ordering by list of primitive or list of complex object is solved by comparison similar to string comparison. Instead of comparing characters, list elements are compared. Lists of complex values follow the default behavior defined for ordering by complex objects regarding comparison.

```
[] < ["a","b","c","d"] < ["a","b","d"]
```

1.11 The expansion concept

Some of the services defined in this document deliver objects, that have reference to each other. The media library for example delivers a list of tracks, where every track can have a reference to an album. An album has a list of tracks itself. To avoid having to many circular references, the concept of resolve level is introduced. The default expansion level is 0 and might be 3 at maximum. Every data structure that potentially contains XObjects supports the expansion level request parameter `$expand`. A client can also request the expansion of certain object references by providing a list of property name on a request.

Expansion does only work on JSON payload. If an object references to binary data, like images, expansion does NOT apply. Of course binary data can NOT be embedded into JSON payload and thus can not be expanded.

1.11.1 No expansion

request:

```
GET medialibrary/albums/6149c270-b528-11e3-a5e2-0800200c9a66 HTTP/1.1
Host: 127.0.0.1:1337
Connection: keep-alive
Accept: application/json
User-Agent: Chrome/34.0.1847.137 Safari/537.36
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8,de;q=0.6
```


response:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Vary: Accept-Encoding
Content-Type: application/json; charset=utf-8
ETag: "-32550834"
Content-Encoding: gzip
Date: Tue, 13 Jun 2014 19:47:27 GMT
Connection: keep-alive
Transfer-Encoding: chunked
```

```
{
  "status" : "ok",
  "data": {
    "id" : "6149c270-b528-11e3-a5e2-0800200c9a66",
    "name" : "its in my pocket",
    "genres": [
      {
        "id" : "92884410-b528-11e3-a5e2-0800200c9a66",
        "name" : "Rock",
        "uri" : "http://127.0.0.1:1337/medialibrary/genres/92884410-b528-11e3-a5e2-0800200c9a66"
      },
      {
        "id" : "81c816a0-b528-11e3-a5e2-0800200c9a66",
        "name" : "Pop",
        "uri" : "http://127.0.0.1:1337/medialibrary/genres/81c816a0-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "artists": [
      {
        "id" : "bb3372f0-b527-11e3-a5e2-0800200c9a66",
        "name" : "ich",
        "uri" : "http://127.0.0.1:1337/medialibrary/artists/bb3372f0-b527-11e3-a5e2-0800200c9a66"
      },
      {
        "id" : "bb3372f0-b500-11e3-a5e2-0800200c9a66",
        "name" : "du",
        "uri" : "http://127.0.0.1:1337/medialibrary/artists/bb3372f0-b500-11e3-a5e2-0800200c9a66"
      }
    ],
    "duration": 42,
    "rating": 2,
    "tracks": [
      {
        "id" : "6ec6abc0-b528-11e3-a5e2-0800200c9a66",
        "name" : "coin",
        "uri" : "http://127.0.0.1:1337/medialibrary/tracks/6ec6abc0-b528-11e3-a5e2-0800200c9a66"
      },
      {
        "id" : "9df7f840-b528-11e3-a5e2-0800200c9a66",
        "name" : "wumpel",
        "uri" : "http://127.0.0.1:1337/medialibrary/tracks/9df7f840-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "discs": 2,
    "image" : "http://127.0.0.1:1337/cdn/images/image09720.png",
    "uri" : "http://127.0.0.1:1337/medialibrary/albums/6149c270-b528-11e3-a5e2-0800200c9a66"
  }
}
```

1.11.2 Single property expansion

A client might want to expand just a single property on a given element. To achieve single property expansion a client adds an `$expand` parameter followed by a comma separated list of property names.

request:

```
GET medialibrary/albums/6149c270-b528-11e3-a5e2-0800200c9a66?$expand=artists HTTP/1.1
Host: 127.0.0.1:1337
Connection: keep-alive
Accept: application/json
User-Agent: Chrome/34.0.1847.137 Safari/537.36
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8,de;q=0.6
```

response:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Vary: Accept-Encoding
Content-Type: application/json; charset=utf-8
ETag: "-32550834"
Content-Encoding: gzip
Date: Tue, 13 Jun 2014 19:47:27 GMT
Connection: keep-alive
Transfer-Encoding: chunked


{
  "status" : "ok",
  "data": {
    "id" : "6149c270-b528-11e3-a5e2-0800200c9a66",
    "name" : "its in my pocket",
    "genres": [
      {
        "id" : "92884410-b528-11e3-a5e2-0800200c9a66",
        "name" : "Rock",
        "uri" : "http://127.0.0.1:1337/medialibrary/genres/92884410-b528-11e3-a5e2-0800200c9a66"
      },
      {
        "id" : "81c816a0-b528-11e3-a5e2-0800200c9a66",
        "name" : "Pop",
        "uri" : "http://127.0.0.1:1337/medialibrary/genres/81c816a0-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "artists": [
      {
        "id" : "bb3372f0-b527-11e3-a5e2-0800200c9a66",
        "name" : "ich",
        "genres": [
          {
            "id" : "92884410-b528-11e3-a5e2-0800200c9a66",
            "name" : "Rock",
            "uri" : "http://127.0.0.1:1337/medialibrary/genres/92884410-b528-11e3-a5e2-0800200c9a66"
          }
        ],
        "rating": 5,
        "tracks": [
          {
            "id" : "1ebe63c0-b528-11e3-a5e2-0800200c9a66",
```

```
        "name" : "me and my empty wallet",
        "uri" : "http://127.0.0.1:1337/medialibrary/tracks/1ebe63c0-b528-11e3-a5e2-0800200c9a66"
      },
      {
        "id" : "9df7f840-b528-11e3-a5e2-0800200c9a66",
        "name" : "wumpel",
        "uri" : "http://127.0.0.1:1337/medialibrary/tracks/9df7f840-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "albums": [
      {
        "id" : "5088aaa0-b528-11e3-a5e2-0800200c9a66",
        "name" : "where is my car",
        "uri" : "http://127.0.0.1:1337/medialibrary/albums/5088aaa0-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "image" : "http://127.0.0.1:1337/cdn/images/image837943.jpg",
    "uri" : "http://127.0.0.1:1337/medialibrary/artists/bb3372f0-b527-11e3-a5e2-0800200c9a66"
  },
  {
    "id" : "bb3372f0-b500-11e3-a5e2-0800200c9a66",
    "name" : "du",
    "genres": [
      {
        "id" : "81c816a0-b528-11e3-a5e2-0800200c9a66",
        "name" : "Pop",
        "uri" : "http://127.0.0.1:1337/medialibrary/genres/81c816a0-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "rating": 4,
    "tracks": [
      {
        "id" : "6ec6abc0-b528-11e3-a5e2-0800200c9a66",
        "name" : "coin",
        "uri" : "http://127.0.0.1:1337/medialibrary/tracks/6ec6abc0-b528-11e3-a5e2-0800200c9a66"
      },
      {
        "id" : "9df7f840-b528-11e3-a5e2-0800200c9a66",
        "name" : "wumpel",
        "uri" : "http://127.0.0.1:1337/medialibrary/tracks/9df7f840-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "albums": [
      {
        "id" : "5088aaa0-b528-11e3-a5e2-0800200c9a66",
        "name" : "where is my car",
        "uri" : "http://127.0.0.1:1337/medialibrary/albums/5088aaa0-b528-11e3-a5e2-0800200c9a66"
      },
      {
        "id" : "6149c270-b528-11e3-a5e2-0800200c9a66",
        "name" : "its in my pocket",
        "uri" : "http://127.0.0.1:1337/medialibrary/albums/6149c270-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "image" : ".dev.portrait2",
    "uri" : "http://127.0.0.1:1337/medialibrary/artists/bb3372f0-b500-11e3-a5e2-0800200c9a66"
  }
],
"duration": 42,
"rating": 3,
"tracks": [
```

```
    "tracks" : [
      {
        "id" : "6ec6abc0-b528-11e3-a5e2-0800200c9a66",
        "name" : "coin",
        "uri" : "http://127.0.0.1:1337/medialibrary/tracks/6ec6abc0-b528-11e3-a5e2-0800200c9a66"
      },
      {
        "id" : "9df7f840-b528-11e3-a5e2-0800200c9a66",
        "name" : "wumpel",
        "uri" : "http://127.0.0.1:1337/medialibrary/tracks/9df7f840-b528-11e3-a5e2-0800200c9a66"
      }
    ],
    "discs": 2,
    "image" : "http://127.0.0.1:1337/cdn/images/image834543.jpg",
    "uri" : "http://127.0.0.1:1337/medialibrary/albums/6149c270-b528-11e3-a5e2-0800200c9a66"
  }
}
```


1.11.3 level expansion

A client might want to expand all properties on a certain level for a given element. To achieve level expansion, a client adds an `$expand` parameter followed a number specifying the expansion level (0-3).

request:

```
GET medialibrary/albums/6149c270-b528-11e3-a5e2-0800200c9a66?$expand=1 HTTP/1.1
Host: 127.0.0.1:1337
Connection: keep-alive
Accept: application/json
User-Agent: Chrome/34.0.1847.137 Safari/537.36
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8,de;q=0.6
```

response:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Vary: Accept-Encoding
Content-Type: application/json; charset=utf-8
ETag: "-32550834"
Content-Encoding: gzip
Date: Tue, 13 Jun 2014 19:47:27 GMT
Connection: keep-alive
Transfer-Encoding: chunked
```


```
{
  "status" : "ok",
  "data": {
    "id" : "6149c270-b528-11e3-a5e2-0800200c9a66",
    "name" : "its in my pocket",
    "genres": [
      {
        "id" : "92884410-b528-11e3-a5e2-0800200c9a66",
        "name" : "Rock",
        "rating": 3,
        "uri" : "http://127.0.0.1:1337/medialibrary/genres/92884410-b528-11e3-a5e2-0800200c9a66"
      },
      {
        "id" : "81c816a0-b528-11e3-a5e2-0800200c9a66",
```

```
      "name" : "Pop",
      "rating": 2,
      "uri" : "http://127.0.0.1:1337/medialibrary/genres/81c816a0-b528-11e3-a5e2-0800200c9a66"
    }
  ],
  "artists": [
    {
      "id" : "bb3372f0-b527-11e3-a5e2-0800200c9a66",
      "name" : "ich",
      "genres": [
        {
          "id" : "92884410-b528-11e3-a5e2-0800200c9a66",
          "name" : "Rock",
          "uri" : "http://127.0.0.1:1337/medialibrary/genres/92884410-b528-11e3-a5e2-0800200c9a66"
        }
      ],
      "rating": 5,
      "tracks": [
        {
          "id" : "1ebe63c0-b528-11e3-a5e2-0800200c9a66",
          "name" : "me and my empty wallet",
          "uri" : "http://127.0.0.1:1337/medialibrary/tracks/1ebe63c0-b528-11e3-a5e2-0800200c9a66"
        },
        {
          "id" : "9df7f840-b528-11e3-a5e2-0800200c9a66",
          "name" : "wumpel",
          "uri" : "http://127.0.0.1:1337/medialibrary/tracks/9df7f840-b528-11e3-a5e2-0800200c9a66"
        }
      ],
      "albums": [
        {
          "id" : "5088aaa0-b528-11e3-a5e2-0800200c9a66",
          "name" : "where is my car",
          "uri" : "http://127.0.0.1:1337/medialibrary/albums/5088aaa0-b528-11e3-a5e2-0800200c9a66"
        }
      ],
      "image" : "http://127.0.0.1:1337/cdn/images/image65476.jpg",
      "uri" : "http://127.0.0.1:1337/medialibrary/artists/bb3372f0-b527-11e3-a5e2-0800200c9a66"
    },
    {
      "id" : "bb3372f0-b500-11e3-a5e2-0800200c9a66",
      "name" : "du",
      "genres": [
        {
          "id" : "81c816a0-b528-11e3-a5e2-0800200c9a66",
          "name" : "Pop",
          "uri" : "http://127.0.0.1:1337/medialibrary/genres/81c816a0-b528-11e3-a5e2-0800200c9a66"
        }
      ],
      "rating": 4,
      "tracks": [
        {
          "id" : "6ec6abc0-b528-11e3-a5e2-0800200c9a66",
          "name" : "coin",
          "uri" : "http://127.0.0.1:1337/medialibrary/tracks/6ec6abc0-b528-11e3-a5e2-0800200c9a66"
        },
        {
          "id" : "9df7f840-b528-11e3-a5e2-0800200c9a66",
          "name" : "wumpel",
          "uri" : "http://127.0.0.1:1337/medialibrary/tracks/9df7f840-b528-11e3-a5e2-0800200c9a66"
```

```
        }
      ],
      "albums": [
        {
          "id" : "5088aaa0-b528-11e3-a5e2-0800200c9a66",
          "name" : "where is my car",
          "uri" : "http://127.0.0.1:1337/medialibrary/albums/5088aaa0-b528-11e3-a5e2-0800200c9a66"
        },
        {
          "id" : "6149c270-b528-11e3-a5e2-0800200c9a66",
          "name" : "its in my pocket",
          "uri" : "http://127.0.0.1:1337/medialibrary/albums/6149c270-b528-11e3-a5e2-0800200c9a66"
        }
      ],
      "image" : "http://127.0.0.1:1337/cdn/images/image837943.jpg",
      "uri" : "http://127.0.0.1:1337/medialibrary/artists/bb3372f0-b500-11e3-a5e2-0800200c9a66"
    }
  ],
  "duration": 42,
  "rating": 2,
  "tracks": [
    {
      "id" : "6ec6abc0-b528-11e3-a5e2-0800200c9a66",
      "name" : "coin",
      "duration": 8,
      "artists": [
        {
          "id" : "bb3372f0-b500-11e3-a5e2-0800200c9a66",
          "name" : "du",
          "uri" : "http://127.0.0.1:1337/medialibrary/artists/bb3372f0-b500-11e3-a5e2-0800200c9a66"
        }
      ],
      "albums": [
        {
          "id" : "5088aaa0-b528-11e3-a5e2-0800200c9a66",
          "name" : "where is my car",
          "uri" : "http://127.0.0.1:1337/medialibrary/albums/5088aaa0-b528-11e3-a5e2-0800200c9a66"
        },
        {
          "id" : "6149c270-b528-11e3-a5e2-0800200c9a66",
          "name" : "its in my pocket",
          "uri" : "http://127.0.0.1:1337/medialibrary/albums/6149c270-b528-11e3-a5e2-0800200c9a66"
        }
      ],
      "image" : "http://127.0.0.1:1337/cdn/images/We9Hv47YeG.jpg",
      "genres": [
        {
          "id" : "81c816a0-b528-11e3-a5e2-0800200c9a66",
          "name" : "Pop",
          "uri" : "http://127.0.0.1:1337/medialibrary/genres/81c816a0-b528-11e3-a5e2-0800200c9a66"
        }
      ],
      "disc": 1,
      "rating": 1,
      "uri" : "http://127.0.0.1:1337/medialibrary/tracks/6ec6abc0-b528-11e3-a5e2-0800200c9a66"
    },
    {
      "id" : "9df7f840-b528-11e3-a5e2-0800200c9a66",
      "name" : "wumpel",
      "duration": 13,
      "artists": [
```

```
        {
          "id" : "bb3372f0-b527-11e3-a5e2-0800200c9a66",
          "name" : "ich",
          "uri" : "http://127.0.0.1:1337/medialibrary/artists/bb3372f0-b527-11e3-a5e2-0800200c9a66"
        },
        {
          "id" : "bb3372f0-b500-11e3-a5e2-0800200c9a66",
          "name" : "du",
          "uri" : "http://127.0.0.1:1337/medialibrary/artists/bb3372f0-b500-11e3-a5e2-0800200c9a66"
        }
      ],
      "albums": [
        {
          "id" : "6149c270-b528-11e3-a5e2-0800200c9a66",
          "name" : "its in my pocket",
          "uri" : "http://127.0.0.1:1337/medialibrary/albums/6149c270-b528-11e3-a5e2-0800200c9a66"
        }
      ],
      "image" : "http://127.0.0.1:1337/cdn/images/ZYnb9UpJxU.png",
      "genres": [
        {
          "id" : "81c816a0-b528-11e3-a5e2-0800200c9a66",
          "name" : "Pop",
          "uri" : "http://127.0.0.1:1337/medialibrary/genres/81c816a0-b528-11e3-a5e2-0800200c9a66"
        },
        {
          "id" : "92884410-b528-11e3-a5e2-0800200c9a66",
          "name" : "Rock",
          "uri" : "http://127.0.0.1:1337/medialibrary/genres/92884410-b528-11e3-a5e2-0800200c9a66"
        }
      ],
      "disc": 1,
      "rating": 1,
      "uri" : "http://127.0.0.1:1337/medialibrary/tracks/9df7f840-b528-11e3-a5e2-0800200c9a66"
    }
  ],
  "discs": 2,
  "image" : "http://127.0.0.1:1337/cdn/images/af7643.jpg",
  "uri" : "http://127.0.0.1:1337/medialibrary/albums/6149c270-b528-11e3-a5e2-0800200c9a66"
 }
}
```


1.11.4 Paging

Every GET request is filterable by using the request parameter `$offset` and `$limit`. Using these parameters, a list response can be offset by `$offset` and limited to a total length of `$limit`. E.g. `GET /tuner/stations?$offset=5&$limit=10` returns a list of 10 elements in total, starting with the 6th (lists start with index 0) element, closing with the 15th element. Both parameters are applicable for events and requests. The response and event payload may contain a `paging` property to help the client determining the previous and next page of results.

`$offset` can be either an integer value or an `uuid`. E.g. `GET /tuner/stations?$offset=932e5b1e-1848-11e5-b60b-1697f925ec7b&$limit=10` returns a list of 10 elements in total, starting with the element `932e5b1e-1848-11e5-b60b-1697f925ec7b`.

`$limit` can be any positive or negative integer. Negative values will return a list of elements before the `$offset`, `$offset` being the last element, positive values let the returned list start at `$offset`.

Retrieve from the end of a list

To retrieve a list backwards a client might use a query with negative `$limit` AND negative `$offset`. The index -1 marks the last element of a list. E.g. `GET /tuner/stations?$offset=-1&$limit=-10` will return the last 10 list, `$offset=-1` marks the last element.

Paging is not only available on resource level but also on nested lists, even though paging nested lists does not tell the client about the previous and next page. Paging nested lists is only possible together with `$fields` filtering by adding `($offset:<offset>,$limit:<limit>)` to the expanded or filtered property name in the query string, link in `/addressbook/contacts/?$fields=emails($offset:0,$limit:2)`.

Service initiated paging

In case of queries that can not be answered at once, a service may send a partial result with paging information attached. E.g. if a client queries `/navigation/pois` (without any filters), the result list might be to big to transfer, so the service will initiate the paging of the result itself, by sending a certain number of results and setting the paging properties accordingly.

1.11.5 Timestamping

All responses can carry an optional relative `timestamp`. The `timestamp` is defined as Integer and expresses the time difference in milliseconds between system boot and message creation. The regular interval is limited to 10 milliseconds so that valid values for `timestamp` always comply to

```
timestamp mod 10 = 0
```

1.12 Publish-subscribe

There is only one WebSocket connection established per client. The WebSockets endpoint is the same as the regular root uri (e.g. `/` or `/api/v2/`). The same port number as for regular http requests shall be used.

Note: WebSockets payload may get concatenated if two messages shall be sent within a frame of 50ms (according to W3C specification). Thus, payload objects always shall have a trailing '\n'. for client side separation.

Definition: An event is considered 'on change' if two subsequent GET requests on the corresponding url would result in different responses on the first level ($expand=0) of the response object. E.g. if an element object is added or removed from a list of objects an 'on change' event is fired or if the name property of a member element object is changed.

ATTENTION Subscriptions on levels other than `<element>` do only support expand level 0.

An event message is defined as object serialized to JSON that is transmitted via WebSocket. The term `<event>` specifies the event `uri` a client wants to subscribe to, unsubscribe from, receive or emit messages for.

The `<event>` follows the the similar syntax as a regular GET request, including query parameters

```
/<service>/<resource>/<element>?<query-params>#<uniqueid-per-session>
```

While `/<service>/<resource>/<element>` describes the actual event `uri`, or `/<service>/<resource>/` describes the resource name, the `<uniqueid-per-session>` parameter allows multiple subscription for the same event with different fields of interest, update rate or updatelimit on client side. Therefore an *optional* `<uniqueid-per-session>` is generated on client side. The server does not care about the `<uniqueid-per-session>` as it is pure client side information. The server **must not** alter or remove `<uniqueid-per-session>` from the event uri. The *optional* `<query-params>` section of the subscription contains the `GET` parameters, a regular `GET` query would have in polling mode.

There are two mandatory properties for every message sent via WebSocket that are `type` and `uri` (=event). The first describes the intention (e.g. 'subscribe') of the message sent, while the later identifies the endpoint itself.

1.12.1 Subscribe

To subscribe to an `<event>`, the client has to send the following JSON stringified object to the server:

```
{
  "type" : "subscribe",
  "event" : "/<service>/<resource>/<element>?<query-params>#<uniqueid-per-session>",
  "interval": <timeStepInMs>,
  "updatelimit": <timeStepInMs>,
  "Authorization": <token>
}
```

The fields a client is interested in and also the number of items defined by `$offset` and `$limit` parameters have to be send in the *optional* `<query-params>` section of the subscription. In case a client subscribes with `$offset` and `$limit` set, a list window is subscribed.

The response to a `subscribe` must follow the structure if it succeeded or trigger an error message if it fails:

```
{
  "type" : "subscribe",
  "event" : "/<service>/<resource>/<element>?<query-params>#<uniqueid-per-session>",
  "status" : "ok"
}
```

1.12.2 Filtering

The subscription takes an optional list of `<attrName>` strings named *fields* that specify the attributes the client wants to subscribe to.

Periodic

The optional *interval* attribute specifies the update frequency in milliseconds for periodic updates, while the optional *updatelimit* attribute specifies the maximum update rate in milliseconds for ‘on change’ notification. If *interval* is set, *updatelimit* is always overruled.

On change

If *interval* is not set, the notification interval is defined ‘on change’ and can be limited by specifying an *updatelimit*. If an ‘on change’ occours before *updatelimit* elapsed, an event will be sent as soon as *updatelimit* elapsed. If there are multiple changes before the next possible update, only the last one know state is sent after *updatelimit* elapsed.

1.12.3 Unsubscribe

To unsubscribe from an `event`, the client has to send the following JSON-serialized object to the server, regardless of the query parameters `?<query-params>` used to subscribe:

```
{
  "type" : "unsubscribe",
  "event" : "/<service>/<resource>/<element>#<uniqueid-per-session>"
}
```

The response to an `unsubscribe` shall follow the structure noted below if unsubscribing succeeds:

```
{
  "type" : "unsubscribe",
  "event" : "/<service>/<resource>/<element>#<uniqueid-per-session>",
  "status" : "ok"
}
```

1.12.4 reauthorize

In case of expiring access tokens (cmp. Authorization), a subscription has to be re-authorized. To reauthorize a subscription for an `<event>`, the client has to send the following JSON stringified object to the server:

```
{
  "type" : "reauthorize",
  "event" : "/<service>/<resource>/<element>?<query-params>#<uniqueid-per-session>",
  "Authorization": <token>
}
```

The subscription parameters do not change with `reauthorize`. The client will receive the same fields, with the same rates a for the original subscription.

The response to a `reauthorize` must follow the structure if it succeeded or trigger an error message if it fails:

```
{
  "type" : "reauthorize",
  "event" : "/<service>/<resource>/<element>?<query-params>#<uniqueid-per-session>",
  "status" : "ok"
}
```

1.12.5 Data

For now, only the server is able to emit `data` events. The emitted data is expected to be a JSON formatted object:

```
{
  "type" : "data",
  "event" : "/<service>/<resource>/<element>?<query-params>#<uniqueid-per-session>",
  "data": <payload>,
  "paging" : {
    "total": Integer,
    "totalPages": Integer
  },
  "timestamp": Integer
}
```

or

```
{
  "type" : "data",
  "event" : "/<service>/<resource>?<query-params>#<uniqueid-per-session>",
  "data": <payload>,
  "paging" : {
    "previous" : "/<service>/<resource>?$limit=<limit>&$offset=<previousoffset>#<uniqueid-per-session>",
    "next" : "/<service>/<resource>?$limit=<limit>&$offset=<nextoffset>#<uniqueid-per-session>",
    "total": Integer,
    "totalPages": Integer
  },
  "timestamp": Integer
}
```

The `<payload>` is defined per event. The server manages the event distribution for all connected clients as defined above, i.e. parsing the event and transmitting it to all registered clients with the expected timing and filtering applied.

In case of paged data response, the `paging` property contains the `previous` and `next` pages a client can query. If the data does not have a `previous` or `next` page, the corresponding property will be undefined. In case of paged results that reach boundaries (either the end or the beginning) of a list, the paging section will contain `previous` and `next` properties that contain appropriate links respecting boundaries like beginning or end.

In addition to those pointers, `total` and `totalPages` are defined to represent the `total` number of items available on the server and the number of `totalPages` resulting in chunking them in `$limit` elements per page. In case of unknown or uncountable items, the properties might be undefined.

A query with `?$limit=0` (ref. Paging) gets the total number of items without actually retrieving a list of items in `data`, which will be an empty list.

request:

```
GET medialibrary/albums?$limit=0 HTTP/1.1
Host: 127.0.0.1:1337
Connection: keep-alive
Accept: application/json
User-Agent: Chrome/34.0.1847.137 Safari/537.36
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8,de;q=0.6
```

response:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Vary: Accept-Encoding
Content-Type: application/json; charset=utf-8
ETag: "-32550834"
Content-Encoding: gzip
Date: Tue, 13 Jun 2014 19:47:27 GMT
Connection: keep-alive
Transfer-Encoding: chunked


{
  "status" : "ok",
  "data": [],
  "paging": {
    "total": 314159
  }
}
```

1.12.6 Error

If any error on the server side occurs an object with type ‘error’ is sent. The following JSON format shall be used:

```
{
  "type" : "error",
  "code": <identifier>,
  "event": /<service>/<resource>/<element>#<uniqueid-per-session>,
  "data": <errormessage>
}
```

The content in `<errormessage>` must be a describing string for the error, while code contains an identifier from the error codes table.

Note: The request leading to this error is NOT processed, i.e. neither subscription, nor unsubscription is processed!

In case of an error during subscription, an immediate response (e.g. code 403) is sent.

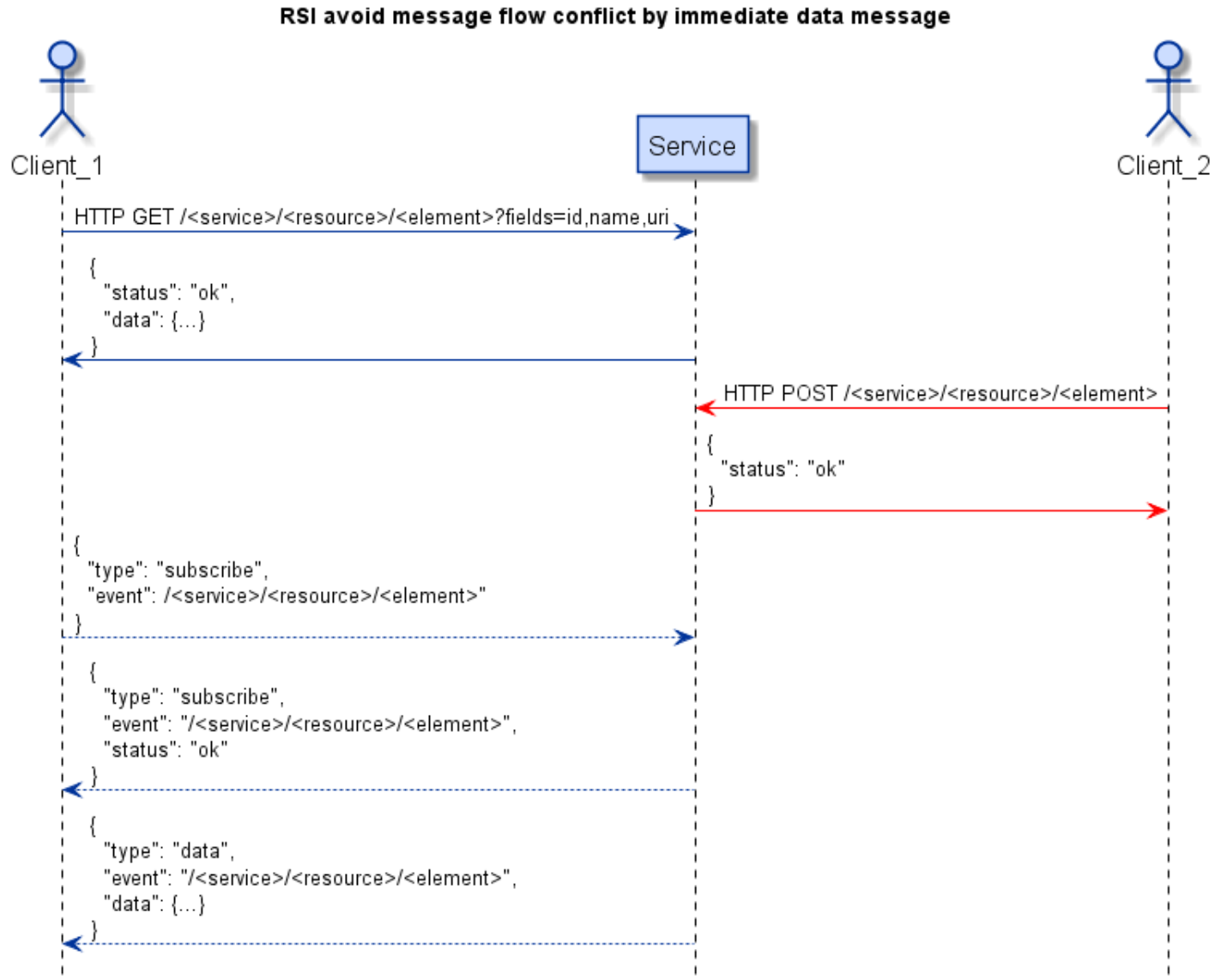
1.12.7 Multi client behavior

The system and therefore the API is designed to be multi client capable. If two clients send subsequent requests to

the server, there is no guaranty that a client achieves the desired system state. E.g. Client A sends a request to tune to a frequency X, Client B may send a request to tune to frequency Y just before tuning to frequency X is finished and clients are notified. The final state will be 'tuned to frequency Y'. The system works in a FIFO fashion not considering latencies in communicating to different clients. To avoid race conditions, clients MUST not force the system to desired state by any kind of cheating like continuous requesting.

1.12.8 Initial data response

Immediately after sending the subscription response, a server shall send the first data message, to avoid race conditions by handing multiple clients. This leads to the messaging sequence shown below:



In order to minimize network traffic a client might send the initial `GET` request filtered for just the mandatory `id`, `uri` and `name` fields by using the `fields` query parameter.

1.12.9 Lifecycle

All subscription are strictly bound to the WebSocket connection, if the connection is lost or is terminated in a regular way, subscription become invalid. The client listening for events has to resubscribe.

1.13 Error codes

The following Error-Codes are defined in addition to regular HTTP error codes that are also used in the event error code. The codes are be transferred within the JSON response, not on HTTP level.

code	meaning
42	syntax error
403	access denied (token invalid or expired)
1337	event name unknown
1895	maximum number of event subscriptions reached
31415	at least one field name unknown

1.14 Security

All information exchanged between any client and server in the network shall be secured by using https and wss protocols. The exploration feature (1.14) has to be limited to only the endpoints that a client is authorized to access (e.g. public endpoints).

There are different well known methods of authentication for web interfaces such as certificate exchange between client and server, username/password exchange or identifier/token transmission.

To expose the API to clients with limited access to those APIs, a user and privileges management service is designed. The details of this process are described in an external `ServiceRegistry_MSC_*` document (also see service registry.

In general, a client registers it self as user with the system. To obtain privileges for a certain `<service>` or `<resource>`, a client has to `POST` a request. There will be a master client that has administrative rights, e.g. the main unit UI, that can can grant or deny the privileges request based on the information provided with the request.

1.15 REST is not RPC

While RPC APIs expose procedures to perform the necessary steps to get from one state to another on the servers side, the REST API has to be understood in a OO (object oriented) way. The server is providing an interface to its objects/models. The client requests changes of the servers models/objects properties.

There is no mechanism defined to use RPC via the interfaces described in here for a reason. There is no need to, because the server models/objects are defined accordingly.

1.16 Exploration

The server provides access its own interfaces by an explore mechanism via GET.

The client can start exploring a server from its root (e.g. `/` or `/api/v2/`) endpoint, which returns a list of available services.

request:

```
GET /api/v2/ HTTP/1.1
Host: 127.0.0.1:1337
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
Accept: application/json;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.
86 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,de;q=0.6
```

response:


```
HTTP/1.1 200 OK
X-Powered-By: Express
Vary: Accept-Encoding
Content-Type: application/json; charset=utf-8
ETag: "-32550834"
Content-Encoding: gzip
Date: Thu, 04 Dec 2015 19:47:27 GMT
Connection: keep-alive
Transfer-Encoding: chunked


{
"status": "ok",
"data": [
  {
    "name": "cdn",
    "id": "cdn",
    "uri": "/api/v1/cdn/",
    "description": "The content delivery network is implemented as a service on the headunit to deliver static content like imag
es, music, video etc. It can be understood as the ‘external media interface’."
  },
  {
    "name": "radio",
    "id": "radio",
    "uri": "/api/v1/radio/",
    "description": ""
  }
  ]
}
```

Querying the next level `/<service>/` returns a list of available resources of the service. Both listings use the following schema:

```
{
  "properties": {
    "type" : "array",
    "items": {
      "type" : "object",
      "properties": {
        "id": {
          "type" :"string",
          "format" : "uuid"
        },
        "name":{
          "type" :"string"
        },
        "uri": {
          "type" : "string",
          "format" : "uri"
        },
        "description": {
          "type" : "string"
        }
      }
    }
  }
}
```

Finally a resource will list all of its elements under `/<service>/<resource>/`.

1.16.1 Reserved keywords

The protocol knows dedicated keywords that a described below.

spec

A request to `/<service>/<resource>/spec` will return a schema of the resources interface according to the viwi schema specification (refer to the actual viwi object definition). The resource describes its own object shape this way.

id

A request with the keyword `id` on any level will return a plain string, the `id`. In the case of a request on root level, the id is a unique identifier for the system. On all other levels, the id of the element of interest will be return in the same plain way.

Example: Request the system id request:

```
DELETE /id HTTP/1.1
Host: 127.0.0.1:1337
Accept: application/json;q=0.8
```

response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8


d2cb3c92-ad8f-496f-b463-7a86973c677a
```

Example: Request a `<service>` s id request:

```
DELETE /<service>/id HTTP/1.1
Host: 127.0.0.1:1337
Accept: application/json;q=0.8
```

response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8


864d79f9-a1ea-405a-82a3-2a62c3745f25
```

2 Global JSON objects

Draft

2.1 General

General object structures are introduced below. All keys in request and response (including event payload) JSON have to be treated with regard of their casing, to avoid conflicts reading or writing values from or to a JSON object. I.e keys like `RouteCalculationProgress` is different to `routecalculationprogress`.

2.1.1 ResponseObject

All viwi objects are JSON (objects), that are encapsulated in a general response structure, except binary data such as images, videos etc.. This response structure is different between HTTP payload and Publish-Subscribe messages, while the `data` property in both cases is identical and is considered being the actual *payload*. If a request was successfully processed (HTTP 200), an object in the following notation is returned:

```
{
  "status" : "ok",
  "data": <Response>,
  "paging" : {
    "total": Integer,
    "totalPages": Integer,
    "previous": String,
    "next": String
  },
  "timestamp": Integer
}
```

In case of paged data response, the `paging` property contains the `previous` and `next` pages a client can query. Of course, paging only applies to `/<service>/<resource>` queries. If the data does not have a `previous` or `next` page, the corresponding property will be undefined.

The `<response>` contains the actual JSON payload that is defined separately for each REST endpoint. If a request failed, status is set to ‘error’ and an additional errorMessage is attached to the response as follows:

```
{
  "status" : "error",
  "data": <Response>,
  "message": <String>,
  "code": <Identifier>
}
```

For responses on `<service>` level (e.g. `/api/v1/myService`), a ResponseObject contains an additional `service` field to represent itself as a serviceObject:

```
{
  "status" : "ok",
  "data": <Response>,
  "paging" : {
    "total": Integer,
    "totalPages": Integer,
    "previous": String,
    "next": String
  },
  "timestamp": Integer,
  "service": serviceObject
}
```

2.1.2 StatusObject

A StatusObject is a general object without data (payload) attribute. For HTTP response code 201 Created, the HTTP header 'Location' is the one of the new element which was created by the request (HTTP/1.1 status codes, 2012, RFC2616). Possible StatusObjects are shown below:

OK

```
{
  "status" : "ok"
}
```

Error

```
{
  "status" : "error",
  "message": <String>,
  "code": <identifier>
}
```

2.2 XObject

The XObject is the general object used to derive all detailed objects from. Thus, every object exchanged through the API defined inhere has at least all XObject properties (mandatory). Optional properties for any object are only sent if applicable. A missing property is treated as being `undefined` and thus not processable for the client.

Property	Description	Type
id	identifier	uuid
name	object name	String
uri	object uri	URI

The `uri` of an XObject can be either relative or absolute. If the XObject refers to an element available on the same host and port, the `uri` has to be relative, else if the referred element is stored on a different host or port, the `uri` must be absolute, to keep the amount of redundant data and thus traffic at a minimum.

2.3 binary content

There are endpoints like the cdn service that delivery binary data. This data is transmitted with regular HTTP headers according to its MIME-type. Subscription on binary data like images, video, certificates etc. is not possible. Binary data is considered being `static`.

2.4 Schema

All object properties make use JSON Schema definitions RFC7159. In addition to the formats defined in RFC7159 the `date-time`, `date` and `time` formats are derived but not identical to the definition in RFC3339. APIs may also use the following (proprietary) formats:

```
{
  "uuid": {
    "description": "unique identifier",
    "regex": "^[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fAF]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}$",
    "example": "388d3a25-663b-11e3-949a-0800200c9a66"
  }
```

```
},
"geoposition": {
  "description": "latitude followed by longitude and altitude in meters separated by ';'",
  "regex": "^[-+]?[0-9]*\.?[0-9]*;[-+]?[0-9]*\.?[0-9]*;[-+]?[0-9]*\.?[0-9]*$",
  "example": "37.772323;-122.214897;0"
},
"e164telephonenumber": {
  "description": "E.164 encoded Telephone number",
  "regex": "^\+?\d{4,23}$",
  "example": "+49536190"
},
"macaddress": {
  "description": "device mac address",
  "regex": "^([0-9A-F]{2}[:-]){5}([0-9A-F]{2})$",
  "example": "a3:3E:ff:e3:01:fe"
},
"rgba": {
  "description": "rgba color with alpha",
  "regex": "^rgba\((\d{1,3}),\s*(\d{1,3}),\s*(\d{1,3})(?:,\s*(\d(?:\.\d+)?))?\)$",
  "example": "rgba(25,9,19,0.4)"
},
"language": {
  "description": "HTTP1.1 compatible language tag",
  "regex": "^(\w{2})(-\w*)?$",
  "example": "en-gb"
},
"servicecategory": {
  "description": "plain string category name",
  "regex": "^[a-z]$",
  "example": "car"
},
"duration": {
  "description": "iso8601 duration",
  "regex": "^P\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}$",
  "example": "P0003-06-04T12:30:05"
},
"temperature": {
  "description": "temperature measurement unit",
  "regex": "^[KCF]{1}$",
  "example": "C"
},
"distance": {
  "description": "distance unit",
  "regex": "^mi|km|yd|m|ft$",
  "example": "km"
},
"speed": {
  "description": "speed unit",
  "regex": "^kmh|mph$",
  "example": "kmh"
},
"date-time": {
  "description": "date-time (based on RFC3339 5.6) detailed fraction",
  "regex": "^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d{3})?((Z|(\+|\-)\d{2}:\d{2}))$",
  "example": "2005-11-12T12:01:42.123+01:00"
},
"time": {
  "description": "time (based on RFC3339 5.6) detailed fraction",
  "regex": "^\d{2}:\d{2}:\d{2}(\.\d{3})?((Z|(\+|\-)\d{2}:\d{2}))$",
  "example": "12:01:42.123+01:00"
},
```

```
  "date": {
    "description": "date (RFC3339 5.6)",
    "regex": "^\d{4}-\d{2}-\d{2}$",
    "example": "2005-11-12"
  },
  "uri": {
    "description": "uri with schema http(s) or w/o Authority",
    "regex": "^(https?:(\/\/([a-z0-9\-._~%]+|\[[a-f0-9:.]+\])(:[0-9]+)?))?(\/[a-z0-9\-._~%!@include("assets/formats.json")'()*+,;=:@]
+)*\/?(\?[a-z0-9\-._~%!@include("assets/formats.json")'()*+,;=:@\/?]*)?(\#[a-z0-9\-._~%!@include("assets/formats.json")'()*+,;
=:@\/?]*)?$",
    "example": "http://www.example.com/aDocument?foo#bar"
  },
  "ical": {
    "description":"iCal format defined in https://tools.ietf.org/html/rfc5545",
    "regex": "^BEGIN:VCALENDAR[\S\s]+END:VCALENDAR$",
    "example": "BEGIN:VCALENDAR  END:VCALENDAR"
  }
}
```

It is also allowed to define a dedicated regular expression pattern as format on property definition level.

2.4.1 number vs integer

JSON does not allow non-numbers like `NaN`, nor does it make any distinction between integer and floating pint. If the format `integer` is used in this document or the object definitions, the meaning is that a service will send and expect integer values. The parsing must be compatible with parsing a number into an integer, because there is no distinction during transfer (`on the wire`). If the format is defined as `number`, neither client nor service may parse it as an integer.

2.4.2 Language tags

The specified JSON format `language` allows the following tags:

```
[
  "ar",     // arabic
  "az",     // azerbaijanian
  "bg",     // bulgarian
  "bs",     // bosnian
  "cs",     // czech
  "da",     // danish
  "de",     // german
  "el",     // greek
  "en-gb",  // english UK
  "en-us",  // english US
  "es-es",  // spanish
  "es-mx",  // spanish US
  "et",     // estonian
  "fi",     // finnish
  "fr-ca",  // frensh cnadian
  "fr-fr",  // french
  "hi",     // hindi
  "hr",     // croatian
  "hu",     // hungarian
  "id",     // indonesian
  "it",     // italian
  "ja",     // japanese
  "ko",     // korean
  "lt",     // lithuanian
  "lv",     // latvian
  "ms",     // malaysian
  "nl",     // dutch
  "no",     // norwegian
  "pl",     // polish
  "pt-br",  // portuguese brazil
  "pt-pt",  // portuguese
  "pt-pt",  // portuguese US
  "ro",     // romanian
  "ru",     // russian
  "sk",     // slovak
  "sl",     // slowenian
  "sr",     // serbian
  "sv",     // swedish
  "th",     // thai
  "tr",     // turkish
  "uk"      // ukranian
  "zh-cn",  // chinese mandarin
  "zh-hk",  // chinese cantonese
  "zh-tw",  // chinese tradional
]
```

2.4.3 Service category names

Services are categorized to let clients register and find similar services. Services are found via `GET` on root level, using the query parameter `?servicecategory=<name>`, wherein `<name>` is one of the following category names:

```
[
  "addresses",        // address related information
  "auth",             // authentication and authorization
  "car",              // vehicle states, vehicle configurations
  "cdn",              // static content
  "charging",         // charging management
  "ev",               // electric vehicle information, states and configurations
  "search",           // dedicated searches that aggregate general searching
  "hybrid",           // HEV related information, states and configurations
  "map",              // digital map
  "maprendering",     // visual maps
  "inputdevices",     // input devices like optical sensors, hardkeys, capacitive keys
  "media",            // media related like rendering, queues(collections)
  "medialibrary",     // services hosting media items like tracks, albums or videos
  "mixer",            // audio management
  "navigation"        // route calculation and navigation general
  "communication",    // phone, messaging etc.
  "system",           // system related, like performance, window management or registration
  "radio",            // radio
  "speech"            // speech dialogs
]
```

2.4.4 Language

Every request may contain an `Accept-Language` Header field, which allows the client to let the service know about the accepted languages (cmp. RFC2616#14.4).

In case of language depended content (e.g. city names), the service shall add the `Content-Language` Header to the response representing the actual language used to generate the content.

3 uuid generation

The viwi protocol uses `uuid`s for element identification on different levels. A `uuid` is a 128bit number. In its canonical form, a UUID is represented by 32 lowercase hexadecimal digits, displayed in five groups separated by hyphens, in the form `8-4-4-4-12` for a total of 36 characters (32 alphanumeric characters and four hyphens). For example:

```
5967E93F-40F9-4F39-893E-CC0DA890DB2E
```

In the canonical representation, `xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx`, the most significant bits of N indicates the variant (depending on the variant; one, two, or three bits are used). The variant covered by the UUID specification is indicated by the two most significant bits of `N` being 1 0 (i.e., the hexadecimal `N` will always be 8, 9, A, or B).

UUID specification describes five versions. The four bits of `M` indicate the UUID `version` (i.e., the hexadecimal `M` will be either 1, 2, 3, 4, or 5).

Each `version` uses different information to generate the `uuid` and thus some may be more appropriate than the others in specific use cases. According to the specification, Version 1 UUIDs are generated from date-time and MAC address, Version 2 UUIDs are generated from group or user id and date-time, Version 3 & 5 produces deterministic UUIDs generated from a user-specified namespace and user-supplied data, and Version 4 is generated from pseudo-random number.

For the ids generated by services in the context of this protocol, the use of Version 4 and Version 5 a recommended.

3.1 Namespaces

Namespaces are, themselves, `uuid`s. While the UUID specification gives example UUIDs for namespaces corresponding to fully qualified domain names (DNS), URLs, ISO OIDs, and X.500 DNs, any UUID can be used as the namespace when generating Version 5 `uuid`s. Thus, nesting of namespaces is possible. A good practice is to use `<service>` and `<resource>` names for namespacing.

3.2 Version 4 (random)

Version 4 UUIDs use a scheme relying only on random number generations. This algorithm sets the version number (4 bits) as well as two reserved bits. All other bits (the remaining 122 bits) are set using a random or pseudorandom data source. Version 4 UUIDs have the form

```
xxxxxxxx-xxxx-4xxx-Yxxx-xxxxxxxxxxxx
```

where `x` is any hexadecimal digit and `Y` is one of 8, 9, A or B.

3.3 Version 5 (SHA-1 hash & namespace)

Version 5 UUIDs use a scheme with `SHA-1` hashing. Note that the 160 bit SHA-1 hash is truncated to 128 bits to make the length work out. An erratum addresses the example in appendix B of RFC 4122.

3.3.1 uniqueness criteria

In order to generate deterministic `uuid`s for entities that need to be recognized being identical on different systems, Version 5 shall be used. The characteristics used to determine similarity shall be used as inputs for the hashing function. An example for such an entity is a FM radio station, which is exactly characterized by its `ecc` and `pi` codes. The generation rule in this case would be

```
uuid = sha1(namespace.toBytes()+eec.toBytes()+pi.toBytes()).substr(0,32);
```

Thus, the same broadcasting station will get the same `uuid` on every system. Transferring p[illegible]ts based on stati[illegible] ids or finding the correct station icon becomes very easy when using this approach.

The uniqueness criteria and thus the generation rule have to be defined on a per resource b[illegible].

4 Interface design patterns

4.1 Settings

When ever a service needs general settings which also may include a reset to factory defaults, a `settings` resource is used. The individual settings are evaluated by their name and value/state.

4.2 Naturally delayed responses

There are cases, where a delayed and incremental results to a query can generally be expected. In case of Bluetooth device discovery for example a server side device scan has to be triggered. In these cases an immediate response to a GET request is neither possible nor practical for client side usage. The general pattern for these kind of applications/services should follow the principle :

A `POST` request on resource level is used to create a new object. The client can subscribe to `on change` events for the newly created object or poll (`GET`) it continuously. The newly created object will change over time until completion. This pattern is applicable to:

- route calculation (/navigation/routes)
- device discovery (/bluetooth/scans)
- search auto completion (/deezer/searches)

4.3 Server vs. client decision

Instances that receive direct user input, like buttons, graphical user interfaces or key word activation shall be clients rather than servers. Servers have to serve multiple clients at once.

4.4 Endpoint naming

In addition to utilizing the HTTP verbs appropriately, `<service>` and `<resource>` naming is arguably the most debated and most important concept to grasp when creating an understandable, easily leveraged web service API. When resources are named well, an API is intuitive and easy to use. Done poorly, that same API can feel klutzy and be difficult to use and understand. The following rules are to follow for a well defined API syntax:

- for `<service>` names use the most specific noun possible to group its `<resource>` s
- `<resource>` has to be a plural noun to express that it holds a list of `<element>` s
- APIs should be easily readable and understandable like `GET /media/renderers` or `DELETE /medialibrary/albums/ED757E27-BA8B-4A67-9798-360F78AF6445?$fields=releasedate`

4.5 Property naming

JSON properties are treated as keys in a map/dictionary. Thus, the property names have are case-sensitive. In order to make the API more readable and consistent, property names shall be camelCased starting with a lower case character. Properties on the same object must not only be distinguishable by their casing, such a situation is considered a collision without technically being one.

4.6 Do not use inline objects

Following the design rule that every object in the system can be referenced to by its own `uri` , any kind of inline object definition is forbidden. If optimized data structures (like structs in OO languages) are needed, first ask, if some other component may have an interest in the complex structure of question. If this is not he case and will never be the case

in future, consider moving sub-properties of structure in question one level up to be regular properties of the object itself. If that does not work, e.g. because you need a list of more complex structures, define a regex format (schemes) and use strings instead.

5 API versioning

Building APIs requires more thought up front than agile developer might be used to. Paths, data structure, meta-data and extensibility are important and would be best considered up front. Once those decisions have been made, changes to the structure would potentially result in breaking older versions of the API. Thus, API design and setting some rules for future consistency is important.

The versioning discussed in here only applies to the interface definition and does not tell anything about the service or client versions that implement the interface.

5.1 Semantic versioning

The versioning is based on SemVer 2.0.0. The multi digit version number, separated by "." consists of the three:

- major: the API is entirely reworked
- minor: the API has breaking changes to other versions
- patch: the API has compatible changes to other versions that compile to the version number `major.minor.patch`

The `major` version can also be considered as the generation which can be part of the root level uri seen in the document (e.g. `/api/v1/`).

Reasons for changing the major version

In case of a total rework of the API, a new major version has to be used. The following actions are considered being a total rework:

- rearrangement/restructure of `<service>`s and their `<resource>`s, i.e. new `<service>`s are created, older disappear, resources are assigned to a different `<service>`
- new general information and access methods are added that are incompatible with older versions in general

5.1.1 Reasons for changing the minor version

Changing the minor version indicates a breaking change or a noteworthy extension/enhancement of the API or general mechanisms. The following reasons are being considered noteworthy or breaking:

- deletion or renaming of properties
- changing semantics of properties (e.g. split into multiple properties)
- changing limits on the value range of a writable property
- add required properties for POST or PUT
- remove restrictions in the query result (e.g. a special order of a list)

5.1.2 Reasons for changing the patch version

The API will need continuous extension, so the versioning level `patch` is used to mark exactly those compatible changes. The following changes to the API are considered being compatible:

- add properties on object level
- add `<service>`s
- add `<resource>`s
- add general optional query parameters
- add general optional meta information to response or status objects
- add new formats
- Changing comments on property level and descriptions on any level, if these changes do NOT imply a semantic change of the API
- mark properties a being deprecated in the API comments (not their name)

5.1.3 Deprecation

Entities on all level may be deprecated at some point in time. Following the versioning rules, an API can only be marked as `deprecated` on `patch` level, being removed earliest on `minor` level. Marking means putting a marker `@deprecated:` in front of the description of the entity in question. I case of a replacement, an additional marking `@replacedby:` is strongly recommended to be placed into the description too.

5.1.4 Pre-release tagging

Any API version can be pre-released mainly for review purposes, following the pre-release rules of SemVer (§9 @ SemVer2.0.0).

version ranges To express the support of a version range, a service might register with range notation. Only the SemVer (§9 @ SemVer2.0.0) notation `~` is supported and expresses a range:

```
1.4.1 <= ~1.4.1 < 1.5.0
```

The `^` notation is NOT supported.

5.1.5 Accessing the interface (certain version)

To access a specific API version, the client has to provide the desired version indication in the Accept-Header. If no such information is provided, the latest available version is used.

To indicate the desired version, the vendor tree following http://tools.ietf.org/html/rfc4288#section-3.2 is used:

```
Accept: application/vnd.viwi.v<MAJOR>.<MINOR>.<PATCH>+json
```

To access the API version 1.4.2 the Accept Header would be:

```
Accept: application/vnd.viwi.v1.4.2+json
```

The response has to be at least at the requested API version `patch` level and **must NOT** leave the requested `major` or `minor` level.

5.1.6 Per service versioning

Every `<service>` can work with its own API version, while the resources provided by this `<service>` have to be consistent with the `<service>` version.

There is **NO** package versioning that groups `<service>` versions into an overall API (global) version.

Every `<service>` will tell its supported versions by setting the `versions` property of the `<service>` object to a list of supported versions.

6 Service registry

A service will register itself by `PUT` ing itself into the root level providing information about it self with the serviceObject. It can unregister itself by using `DELETE`.

A detailed explanation of the registration process can be found in an external `ServiceRegistry_MSC_*` document.

6.1 serviceObject

In order to register a service with the service registry, a special object called `serviceObject`, inherited from xObject, definition is available.

```
{
  "id": {
    "type": "string",
    "description": "service id (if sent with registration request, it will be ignored by the registry)",
    "format": "uuid"
  },
  "name": {
    "type": "string",
    "description": "service name"
  },
  "uri": {
    "type": "string",
    "description": "service uri (the desired <service_path> relative to the root of the service registry)",
    "format": "uri"
  },
  "description": {
    "type": "string",
    "description": "human readable description of the micro service"
  },
  "port": {
    "type": "integer",
    "description": "TCP port the service is running on"
  },
  "serviceCategories": {
    "type": "array",
    "items": {
      "type": "string",
      "description": "predefined key words"
    }
  },
  "privileges":  {
    "type": "array",
    "items": {
      "type": "string",
      "description": "relative path to the service or resource"
    }
  },
  "versions":  {
    "type": "array",
    "items": {
      "type": "string"
      "description": "supported version in semVer notation"
    }
  }
}
```

`privileges` lists the obtainable privileges for a service or resource. `serviceCategories` lists the categories a service is assigned to. The supported API versions are listed as strings in semVer format (major.minor.patch) under `versions`.

6.2 Registration

The service registers itself by providing the name it wants to be accessible under, relative to the root level uri (e.g. `/` or `/api/v1`).

The registering service needs to provide privileges as an array containing the Micro Services access rights structure (e.g. /car/settings, /car/units, /tuner), i.e. privileges contains either a `<service>` list or a `/<service>/<resource>` list. `<service>` s and `/<service>/<resource>` may be mixed in this list.

The service registry generates an id (uuid) for the service and responds with the newly registered services id in the `Location` header like in the following example:

```
Location: https://<IP-address_of_service_registry>:<port_of_service_registry>/api/v1/<uuid_of_service>
```

or

```
Location: /<uuid_of_service>
```

Refer to version ranges to determine the necessary versions notation.

registration example:

request:

```
PUT /<service_path> HTTP/1.1
Host: 127.0.0.1:1337
Accept: application/json;q=0.8


<serviceObject>
```

response:

```
HTTP/1.1 201 Created
Content-Type: application/json; charset=utf-8
Location: <URI of created service>


{
  "status": "ok"
}
```

6.3 Unregistration

Unregistration is only allowed on `id` level, i.e. by sending `DELETE`. `DELETE` requests on other accessing paths will result in a status code 400 Bad request

request:

```
DELETE /<service_uuid> HTTP/1.1
Host: 127.0.0.1:1337
Accept: application/json;q=0.8
```

response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8


{
  "status": "ok"
}
```


Only the user (TLS user identity) which has registered the service is allowed to unregister the service. If the wrong user (client or service) tries to delete the service, the Service Registry will deny the `DELETE` request by sending the HTTP 403 Forbidden status code.

6.4 valid status codes

The Service Registry may respond with different status codes, depending on the requested action. The `30X` range differs from regular viwi.

code	content type	meaning
200	content type	request successful, regular content follows
201	content type	creation succeeded, Location header set
300	list of potential redirect uris	Multiple matches found, client has to decide
307	no payload, just redirect	Exact match found

7 User Authentication and Authorization

In order to secure API access and guarantee delivery of authorized content only, a token mechanism (cmp. OAuth 2.0 - RFC6749, JWT - RFC7519) is used. The token mechanism can be understood as a dedicated valet key for valet services. This key often only allows the car to be driven a particular distance, and typically does not unlock the trunk and glove box. OAuth is often referred to as a valet key for the web in that it grants an application access to protected data only for specific uses and often for a limited amount of time. Also, authentication via tokens never requires actually passing user credentials from one client or service to another. Once the authentication and validation process begins, the user is driven to a secure URL where credentials are requested, but those credentials are never shared outside of that secure URL.

The client logging to the system may receive two different tokens upon successful login, the `accessToken` and the `refreshToken`. While the `accessToken` can be handed between clients and services, clients and clients as well as services and services, the `refreshToken` is a secret that shall be kept by the client that logged in to the system by any authorization mechanism. The `refreshToken` is used to obtain a new `accessToken` once it becomes invalid or expires. To avoid latency, a client shall check the expiration of the `accessToken` and request a new `accessToken` before the actual expiration.

HTTP provides a dedicated `Authorization` Header for transferring the `accessToken` with each request. In case of indirect request, i.e. request a service to deliver information that rely on another services information, the `accessToken` is handed from one to the other service.

WebSockets multiplex messages that might be affected by authorization, so the `subscribe` message (Publish Subscribe) contains the optional `Authorization` property. Once an `accessToken` expires, the corresponding subscription(s) will send an `error` message with the error code 403. This is the indication that a new subscription with a valid token has to be sent or a reauthorization of the subscription with a new, valid `accessToken` is needed.

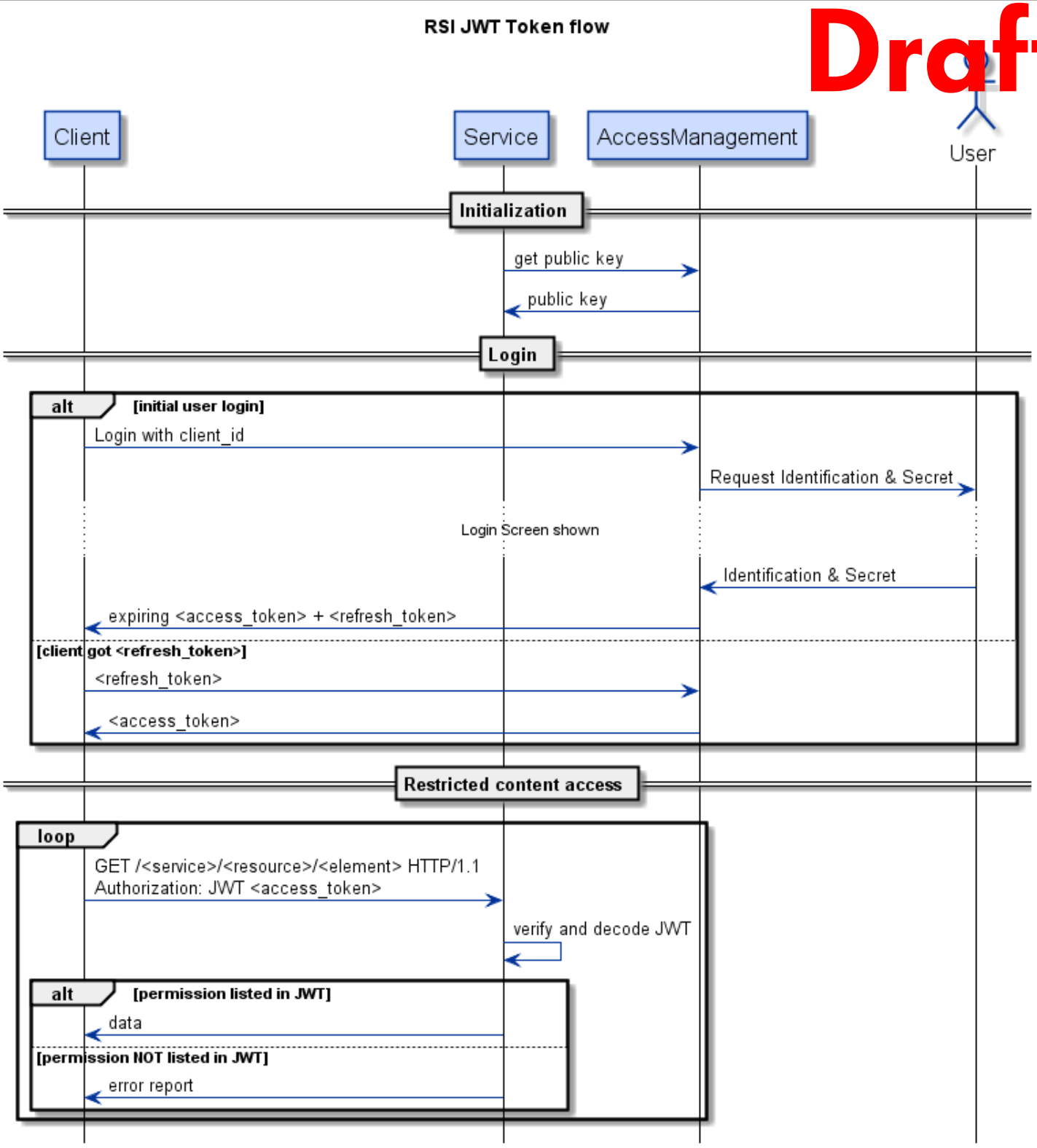
7.1 Token types

While the protocol itself is agnostic to the type of token used for accessing restricted resources, the following section shall give a short overview of technologies that best fit the needs of the protocol. The type of token to be used in an actual implementation may vary from `<service>` to `<service>`, if not otherwise specified in an overall system specification. The protocol itself is neither limited nor does it benefit from choosing the one or the other alternative.

7.1.1 JWT

JSON Web Token (JWT) is an open standard RFC 7519 that defines a compact and **self-contained** way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can generally be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA.

For the application to the viwi protocol, the Token issuer has to provide access to its public key for Token verification. The payload contains all the required information about the user, avoiding the need to query the database more than once, thus they are called **self-contained**, so verification is improtant and shall be implemented by all service handling sensitive information.



JWT tokens will be sent in the `Authorization` header, following with term `JWT` and a space character.

Example that assumes `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiSm9obiBEb2UifQ.xuEv8qrfXu424LZk8bVgr9MQJUIrp1rHcPyZw_KSsds` is the actual token:

```
GET /<service>/<resource> HTTP/1.1
Host: 127.0.0.1:1337
Authorization: JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiSm9obiBEb2UifQ.xuEv8qrfXu424LZk8bVgr9MQ
JUIrp1rHcPyZw_KSsds
```

Expiry

JWTs are stateless keys to information that needs to protected. Therefore a token has to be *revokable*. As the

underlying concept of self-containing tokens does not allow revocation by a central compone[illegible] a request by request basis, JWTs shall expiry quickly. The makes the possibly time window of using a tok[illegible] wi[illegible]out permissio[illegible] small. A new token can be requested early be using the `refresh_token`.

Identification

Every JWT shall contain a `jti` claim in its header to ensure blacklisting possibilities and to allow one time use. Every JWT shall contain a `aud` claim in its header to allow identification of the audience a token was issued to. This is especially of interest when tokens are handed between `<service>` s.

Trust decisions

The contents of a JWT cannot be relied upon in a trust decision unless its contents have been cryptographically secured and bound to the context necessary for the trust decision. In particular, the key(s) used to sign and/or encrypt the JWT will typically need to verifiable be under the control of the party identified as the issuer of the JWT.

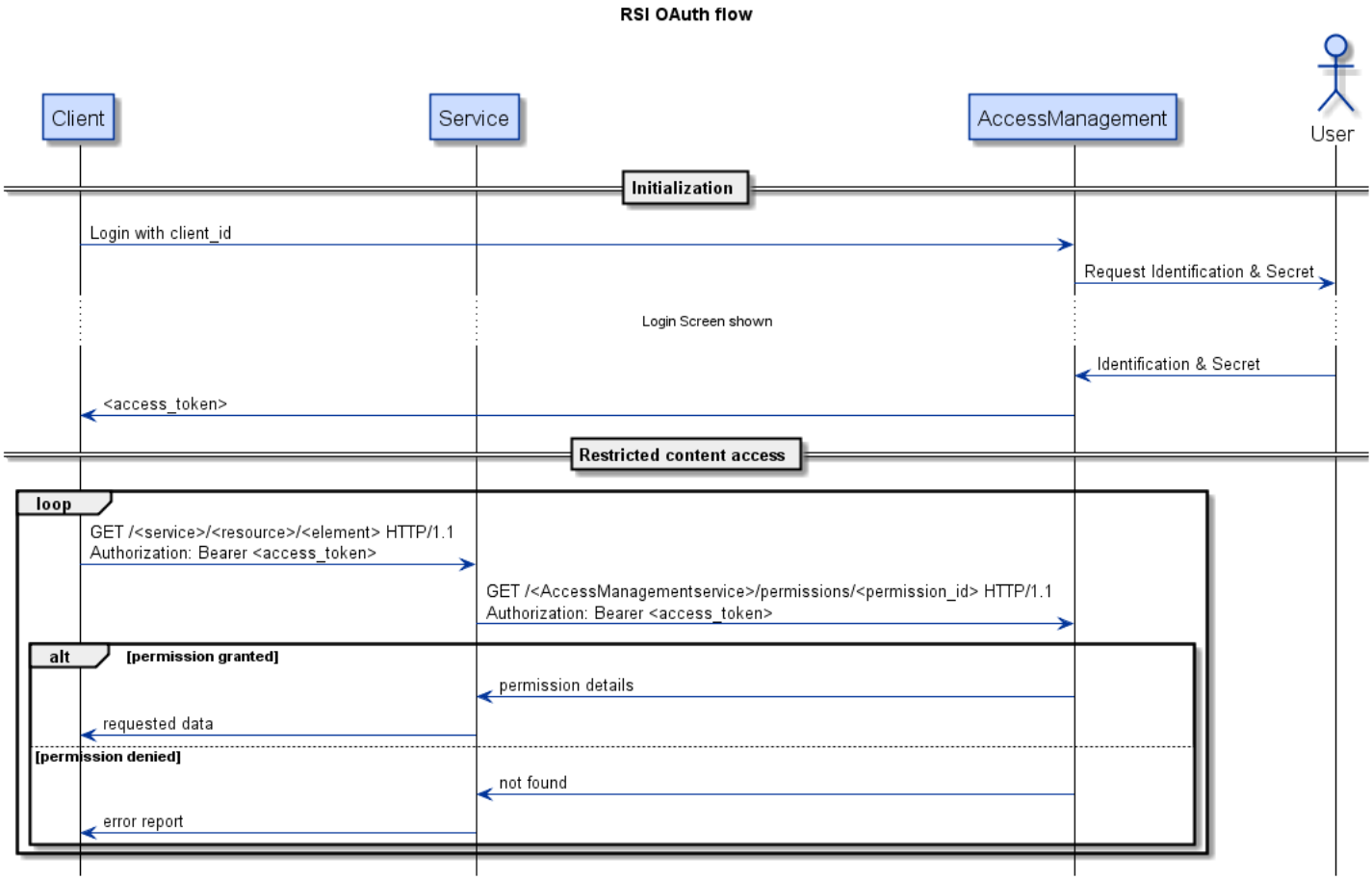
Every JWT shall contain an `iss` claim in its header to identify the principal that issued the token by its absolute `uri`.

Privacy considerations

A JWT may contain privacy-sensitive information. When this is the case, measures must be taken to prevent disclosure of this information to unintended parties. One way to achieve this is to use an encrypted JWT. Another way is to ensure that JWTs containing unencrypted privacy-sensitive information are only transmitted over encrypted channels or protocols, such as TLS. Those Tokens must not be shared between services.

7.1.2 OAuth

Other than the JWT, an OAuth Token does not carry any information about the user, the privileges etc. The OAuth Token is therefore more compact. In order to obtain information about privileges, a services needs to ask the issuer of the token for its validity before responding. The big advantage is that the Token does not have to expire, because it can be revoked between requests easily. The down sid is of course the number of requests send to and from the token issuer.



OAuth tokens will be sent in the `Authorization` header, following with term `Bearer` and a space character.

Example that assumes `mF_9.B5f-4.1JqM` is the actual token:

```
GET /<service>/<resource> HTTP/1.1
Host: 127.0.0.1:1337
Authorization: Bearer mF_9.B5f-4.1JqM
```

8 Changelog

Draft

1.6.0 (2016-07-13)

Bug Fixes

- **globalsJSONObjects:** add mising description for using regex as format (0196d58)
- **patterns:** add missing patterns content - was lost during migration (18fbc4c)
- **POST:** add 'last-update-wins' information at POST (ebc1fec)
- **ResponseObject:** The repsonseObject will only add service information on service level requests, (535b5d2)

Features

- **auth:** add more detailled explaination of JWT vs. OAuth Token usage (94102e3)
- add id and uri to serviceObject to let it be an xObject (2b87095)
- **formats:** add ical format (e47da41)
- **formats:** add URI schema (07c4e85)
- **formats:** separately deliver formats.json to allow automated workflows (694493e)
- **global.formats:** add time date and date-time formats explicitly (26c313a)
- **intro:** add reserved keyword id for direct access of an entities id (dbd36af)
- **intro:** explain the difference between OR and AND query on viwi level (d026220)
- **publishSubscribe:** add reauthorize action to websockets to allow refreshing tokens of an existing s (450de58)
- **statusCodes:** allow 300 response code for service registry responses (c6b69c5)

BREAKING CHANGES

- ResponseObject: The ResponseObject embeds a ServiceObject instead of dedicated properties

1.5.2 (2016-05-10)

Bug Fixes

- **404:** fix confusing explanation (4f872ff)
- **501:** mark status code 501 as not applicable (ab5613c)
- **general:** add missing dependencies (1f68027)
- **Schema:** ad missing comma (cdc4cc6)
- **versioning:** better wording for major version reasons (f0ce9be)
- **versioning:** remove conflicting statement regarding service addition. Addintion leads to patc (482cab5)

Features

- **addressing:** add addressing aspects section (353fbea)
- **Authentication:** add more detailed description of token concept (4c6f951)
- **design patterns:** add factory reset recommendation (4db0cd0)
- **expand:** make clear that referenced binary data can not be expanded (8fd0af4)
- **general:** add uuid generation section (2184f87)
- **general:** describe binary payload exceptions (fe393e4)
- **general:** initial commit based on viwi 1.5.1 (extraction of the original viwi 1.5.1) (26fc333)
- **images:** embed images into md (11cf081)
- **languages:** add content- and accept-language notes (91f7790)
- **PublishSubscribe:** add a note that an immediate response shall be sent on subscription errors (8181ac3)
- **RESTfulAPI:** add a section about caching (ff3c5a3)
- **schema:** add distance and speed formats (c1b8d59)
- **schema:** reference to the actual RFCs for formats (1de7ba4)

- **sorting:** add a note to make sure that a default sort order has to be defined per <res(e6eb0bb)