INTERNATIONAL STANDARD

ISO 20078-2

Second edition
2021-11

Road vehicles — Extended vehicle (ExVe) web services —

Part 2: Access

Véhicules routiers — Web services du véhicule étendu (ExVe) —

Partie 2: Accès

Reference number
ISO 20078-2:2021(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 22, *Road vehicles,* Subcommittee SC 31, *Data communication*.

This second edition cancels and replaces the first edition (ISO 20078-2:2019), which has been technically revised.

The main changes are as follows:

— added definition of a push method, allowing the offering party to push resources to the accessing party according to subscription;

— added definition of reusable subscription profiles, used to store URI locations and authorization information from the accessing party and used by offering party to push subscribed resources;

— defined requirements for container management API;

— added Annex A and digital Annex B describing container management API;

— revised error format;

— redefined resource versioning.

A list of all parts in the ISO 20078 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Road vehicles — Extended vehicle (ExVe) web services —

Part 2:
Access

1 Scope

This document defines how to access resources on a web-services interface of an offering party using the Hypertext Transfer Protocol Secure (HTTPS). Resources can be accessed through request/reply and/or requested to be pushed.

The Representational State Transfer (REST) architectural pattern is chosen as a common way to format resource paths both for request/reply and push. Some specific extensions to this pattern are defined to allow for asynchronous resource requests, such as, for example, forcing readouts of data from a connected vehicle.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 20078-1, *Road vehicles — Extended vehicle (ExVe) web services — Part 1: Content and definitions*

ISO 8601 (all parts), *Date and time — Representations for information interchange*

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 20078-1 apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.2 Abbreviated terms

For the purposes of this document, the abbreviated terms given in ISO 20078-1 apply.

4 Representational state transfer-based interface

4.1 General

4.1.1 Introduction

There are three different ways to access resources for the accessing party, request/reply, push, and push with a subsequent request/reply. Request/reply is the recommended method, but push can be used when needed. In most cases only one method is used for a particular resource, but sometimes both

request/reply and push are needed. Latency, message size and frequency are examples of requirements to consider when selecting between request/reply, push, and push with subsequent request/reply.

4.1.2 Request/reply

Figure 1 shows an example of request/reply sequence, where the accessing party requests a resource using HTTP GET and the offering party returns it in the payload.

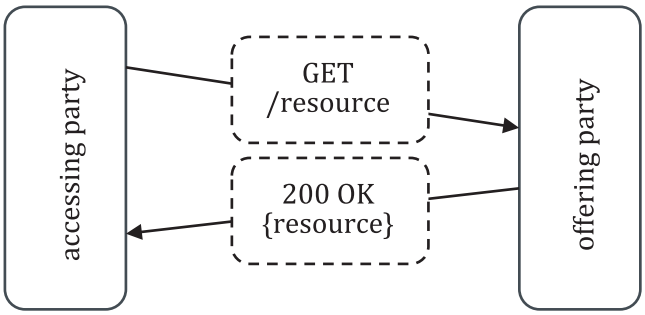

Figure 1 — Request/reply example

4.1.3 Push

Figure 2 shows an example of push sequence, where the accessing party initiates a push of a resource using HTTP POST. The offering party confirms the subscription and starts to push the resource using HTTP POST. Figure 2 shows an example where the push is done two times.

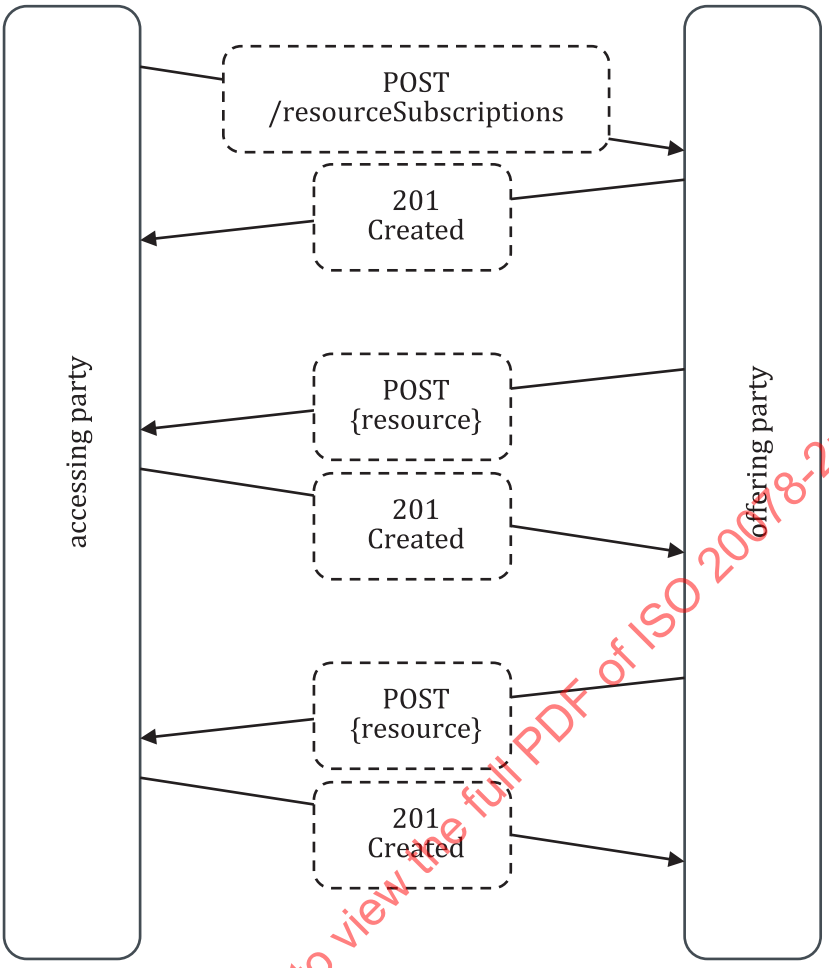

Figure 2 — Push example

Figure 3 shows an example of push sequence, where there is no explicit initiation of the push from the accessing party. It is instead initiated by for instance rules or configuration at the offering party. It is outside the scope of this document to describe how this is agreed between the accessing party and the offering party.

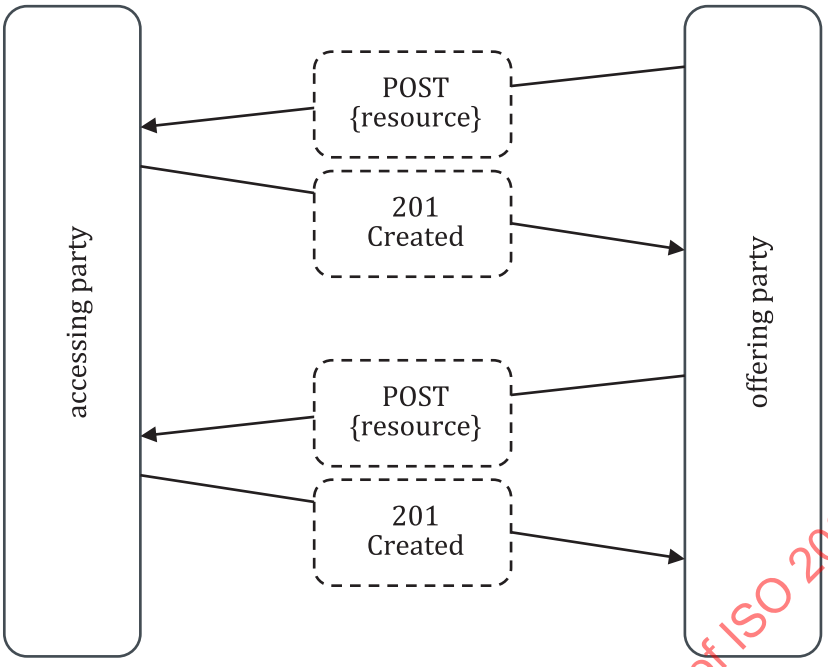

Figure 3 — Push example without explicit initiation

4.1.4 Push with subsequent request/reply

Figure 4 shows an example of push with subsequent request/reply, where the accessing party initiates a push of a resource using HTTP POST. The offering party confirms the subscription and starts to push the resource reference using HTTP POST. The accessing party uses the resource reference to request the resource from the offering party.

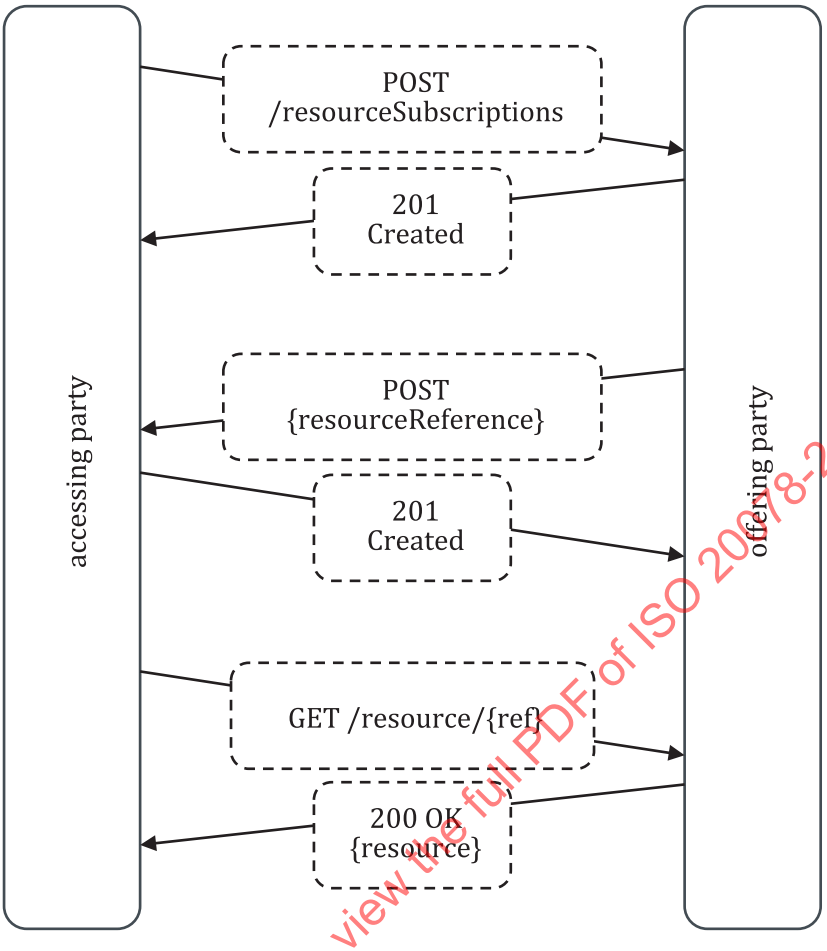

Figure 4 — Push with subsequent request/reply example

Figure 5 shows an example of push with subsequent request/reply, where there is no explicit initiation of the push from the accessing party. It is instead initiated by, for instance, rules or configuration at the offering party. It is outside the scope of this document to describe how this is agreed between the accessing and the offering party.

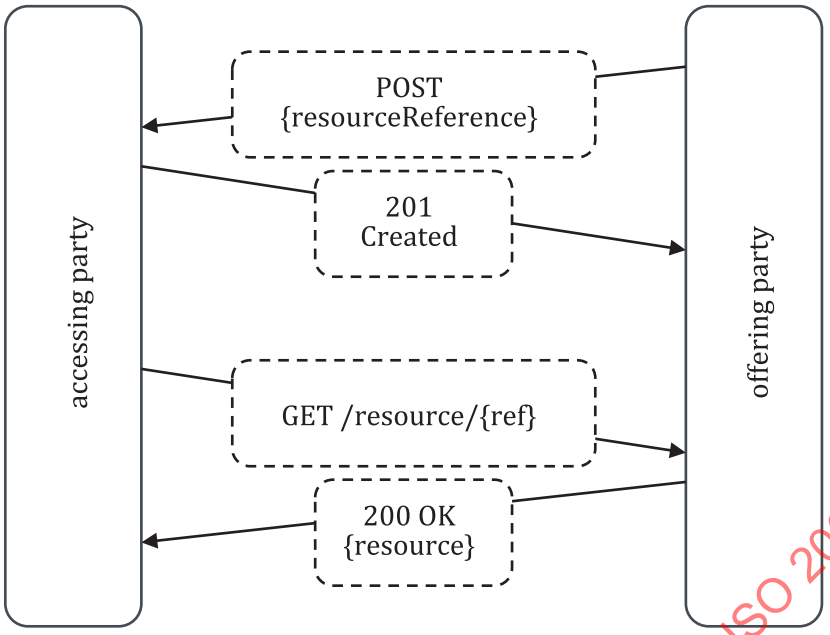

Figure 5 — Push with subsequent request/reply example without explicit initiation

4.1.5 Requirements

The following defines the requirements on a REST[9]-based web service interface using HTTPS[6][7] based on transport layer security (TLS) to give the accessing party secure access to resources provided by the offering party. The requirements in this document are valid both for request/reply and push unless otherwise stated.

REQ_04_01_01	The REST-based web services interface implementation shall use HTTPS as the transport protocol with TLS.

REQ_04_01_02	HTTP shall only be used with version 1.1 or higher compatible versions.

REQ_04_01_03	TLS shall only be used with version 1.2 or higher versions.

REQ_04_01_04	The request/reply REST web service shall be a strict client-server interaction, where the accessing party (client) sends a request and the offering party (server) sends a response.

NOTE Resources can be transferred both in the request and the response.

REQ_04_01_05	The request/reply REST implementation shall be stateless; i.e. the offering party server shall not maintain any accessing party client context or session information.

Due to REQ_04_01_05, each request-response pair is handled independently from one another. Each client request by the accessing party contains all information required by the server of the offering party to successfully respond to the request, including a representation of the client state when necessary.

REQ_04_01_06	A push initiation REST web service shall be a strict client-server interaction, where the accessing party (client) sends a request and the offering party (server) sends a response.

REQ_04_01_07	The push REST web service shall be a strict client-server interaction, where the offering party (client) sends a request and the accessing party (server) sends a response.

4.2 Resources

REQ_04_02_01	Information on the server shall be exposed as resources expressed as plural nouns.

NOTE 1 This holds true even when the resource is only available one time on a connected vehicle (e.g. "odometers").

REQ_04_02_02	The exposed resources shall be uniquely identified in the form of Uniform Resource Identifiers (URIs).

The resources, the resource groups or the containers, and how to apply those on a specific presentation or application layer of the accessing party are described in ISO 20078-1. How an accessing party authenticates and how it is authorized for access to resources is described in ISO 20078-3.

REQ_04_02_03	The offering party shall define the base URIs of the web services offered by the offering party.

Table 1 — Examples of possible ExVe-based URIs

Resource	Description
https://{example.com}/exve/	URI based on sub directory
https://exve.{example.com}/	URI based on sub domain

REQ_04_02_04	The offering party shall comply with the URI resource paths defined by specific ExVe standard applications.

Table 2 — Examples of possible ExVe resource URIs

Resource	Description
{base_URI}/{resourcePath}	Resources based on path
{base_URI}/vehicles	A list of the available vehicles as defined by authorization context
{base URI}/vehicles/{vehicleId}/dtcReadouts/	Read all Diagnostic Trouble Codes for a specific vehicle
{base URI}/vehicles/{vehicleId}/ecus/{ecuId}/dtcReadouts/	Read all Diagnostic Trouble Codes for a specific ECU of a specific vehicle

There are two primary elements defining an URI: entities and resources (Table 2). Entities are the fundamental objects representing, e.g. vehicles, ECUs, drivers and fleets. Resources are the actual data, aggregated information or functionalities associated with an entity and a specific use case.

REQ_04_02_05	Resources shall be named and described.

EXAMPLE Fuel level can be an example of a single data item resource, vehicle position can be an aggregated resource consisting of several data items (e.g. latitude, longitude, sample time) and lock and unlock the vehicle can be a functionality resource.

REQ_04_02_06	The offering party should have the possibility to extend resources, but shall not be able to reduce resources.

Thus, by REQ_04_02_06 it is not possible to remove data items from a resource, other than through an update of the underlying use case specification. It is however possible to add data items to a resource (i.e. versioning).

REQ_04_02_07	Aggregated resources shall only include or cross-reference necessary data items for the complete and correct operation of the related use case.

NOTE 2 REQ_04_02_07 ensures an accessing party only receives data items necessary for fulfilling the intended need and nothing else when accessing the resource (i.e. data economy).

See also ISO 20078-1:2021, 8.3, for further details.

REQ_04_02_08	A resource within an already existing use case should not be redefined.

An application may extend the recommended patterns below if none of them meets the needs of the use case.

REQ_04_02_09	Each use case shall define how the HTTP operations GET, POST, PUT, PATCH, and DELETE are supported for each defined resource and what the response for each operation is.

REQ_04_02_10	All URI elements shall be written in lower camel case notation.

NOTE 3 The {baseURI} in following patterns refers to the offering party root URI (see Table 3).

Table 3 — Examples of base URI expresses REQ_04_02_09 and REQ_04_02_01

Normative generic format	{baseURI}/{entities}
Example	{baseURI}/vehicles Request a list of all vehicles available to the accessing party.

REQ_04_02_11	Relations of entities shall be expressed using subresources.

NOTE 4 See Table 4.

Table 4 — Examples of sub-URI's expresses REQ_04_02_11 and REQ_04_02_01

Normative generic format	{baseURI}/{entities}/{ID}/{entities2}/{ID2}
Example	{baseURI}/fleets/12/vehicles/456 Request information on vehicle with id 456 of fleet with id 12.
	{baseURI}/vehicles/456/ecus/789 Request information on ECU with id 789 of vehicle with id 456.

REQ_04_02_12	A resource shall be placed after the entity to which it belongs in the URI.

NOTE 5 See Table 5.

Table 5 — Examples of descriptive URI's expresses REQ_04_02_12 and REQ_04_02_01

Normative generic format	{baseURI}/{entities}/{ID}/{resource}
Example	{baseURI}/vehicles/123/positions Request all positions for vehicle with id 123.
	{baseURI}/vehicles/456/odometers Request the odometer value for vehicle with id 456.
	{baseURI}/vehicles/456/tirePressures Request tire pressures of all wheels on the vehicle with id 456.

REQ_04_02_13	If filtering of the response is needed, query parameters shall be used.

NOTE 6 Several query parameters can be added to a request.

NOTE 7 See Table 6.

Table 6 — Examples of filtering responses expresses REQ_04_02_13 and REQ_04_02_01

Normative generic format	{baseURI}/{entities}?{filter}={filterValue}
	{baseURI}/{entities}?{filter}={filterValue}&{filter2}={filterValue2}
Example	{baseURI}/vehicles?ignitionState=on Request all vehicles with ignition on.
	{baseURI}/vehicles/123/positions?startDate=...&endDate=... Request the positions for vehicle with id 123 registered in given date span in the ISO 8601 series date-time format.

REQ_04_02_14	If sorting is needed, query parameters shall be used.

NOTE 8 See Table 7.

Table 7 — Examples of query parameters expresses REQ_04_02_14 and REQ_04_02_01

Normative generic format	{baseURI}/{entities}?{sorting}={sortingValue}
	{baseURI}/{entities}?{sorting}={sortingValue}&{sorting2}={sortingValue2}
Example	{baseURI}/vehicles?sortField=id&sortOrder=asc

REQ_04_02_15	If selection of subsets of resources is needed, query parameters shall be used.

NOTE 9 See Table 8.

Table 8 — Examples of query parameters for subsets expresses REQ_04_02_15 and REQ_04_02_01

Normative generic format	{baseURI}/{entities}?{id}={ID}
	{baseURI}/{entities}?{id}={ID}&{id2}={ID2}
Example	{baseURI}/vehicles?id=123&id=124
	{baseURI}/vehicles?id=YS2RX20001754836

Identifiers may come in multiple formats including, but not limited to, VIN or pseudonymized IDs.

REQ_04_02_16	Pseudonymized IDs may be simple numerical IDs, UUIDs or any other alphanumerical scheme.

NOTE 10 See Table 9.

Table 9 — Examples of pseudonymized IDs represented by numerical IDs expresses REQ_04_02_16 and REQ_04_02_01

Normative generic format	{baseURI}/{entities}?{id}={ID}
	{baseURI}/{entities}?{id}={ID}&{id2}={ID2}
Example	{baseURI}/vehicles?id=ce5d5e3d-28bc-475f-8ef7-b5cb9c8039d4&id=f95ce756-42fc-48b2-8873-86553f6df5cc
	{baseURI}/vehicles?id=456

REQ_04_02_17	For large resource responses pagination may be used.

NOTE 11 See Table 10.

Table 10 — Examples of pagination expresses REQ_04_02_17 and REQ_04_02_01

Normative generic format	{baseURI}/{entities}?start={value}&limit={count}
Example	{baseURI}/vehicles?start=20&limit=10

REQ_04_02_18	A GET request on the returned location may return the total amount of results. If used, it shall be part of the message body using the keyword “exveTotal”.

EXAMPLE The following example expresses REQ_04_02_18:

```
{
 "results": {
 "exveTotal": "150",
... }
}
```

REQ_04_02_19	If wildcards are used, a wildcard (*) shall access all subentities or resources of all parent entities.

NOTE 12 See Table 11.

Table 11 — Examples of wildcards in URIs expresses REQ_04_02_19 and REQ_04_02_01

Normative generic format	{baseURI}/{entities}/*/{entities2}
Example	{baseURI}/vehicles/*/positions Request all positions for all vehicles.
	{baseURI}/vehicles/*/ecus Request all ECU’s of all vehicles.

REQ_04_02_20	If wildcards are used, a wildcard (*) may be combined with other filtering, including IDs, to access a smaller number of entities.

NOTE 13 See Table 12.

Table 12 — Examples of wildcards in URIs while filtering by IDs expresses REQ_04_02_20 and REQ_04_02_01

Normative generic format	{baseURI}/{entities}/*/{resource}?{id}={ID}&{id2}={ID2}&{filter}={filterValue}&{filter2}={filterValue2}
Example	{baseURI}/vehicles/*/odometers?id=456&id=789& startDate=...&endDate=... Request all odometer values from vehicles with id 456 and 789 registered within the given time span in the ISO 8601 series date-time format.

REQ_04_02_21	For fully anonymized access, no entity IDs shall be used in the URIs.

NOTE 14 See Table 13.

Table 13 — Examples of an anonymized access expresses REQ_04_02_21 and REQ_04_02_01

Normative generic format	{baseURI}/{entities}/{entities2}
Example	{baseURI}/vehicles/hazardWarnings?isActive=true

REQ_04_02_22	A push resource shall be suffixed with subscriptions.

NOTE 15 See Table 14.

Table 14 — Examples of push resources regarding REQ_04_02_22

Normative generic format	{baseURI}/{resource}Subscriptions
Example	POST {baseURI}/positionSubscriptions?vehicleId=123 Subscribe to positions for vehicle with id 123.
	GET {baseURI}/positionSubscriptions Request a list of all position subscriptions.
	POST {baseURI}/vehicles/456/tirePressureSubscriptions?pressure=low Subscribe to low tire pressures of all wheels on the vehicle with id 456.

REQ_04_02_23	The accessing party shall define the base URIs of the web services offered by the accessing party. (Compare with REQ_04_02_03).

4.3 Subscription profiles

Push of subscribed resources requires URI locations and authorization information from the accessing party. This information is stored in subscription profiles. Subscription profiles can be created in a separate step before creating subscriptions or directly when creating a subscription. In both cases the subscription profile will be assigned an id and can be reused by any number of subscriptions.

REQ_04_03_01	A subscription profile shall contain: — profileId (assigned at creation), — callback baseURI, — token information according to REQ_04_03_02 or REQ_04_03_03.

REQ_04_03_02	A subscription profile for refresh token shall contain: — an OAuth2 refresh token, — the OAuth2 refresh token lifetime, — the OAuth2 endpoint for renewing access tokens.

NOTE 1 See Table 15.

REQ_04_03_03	A subscription profile for bearer token shall contain: — a bearer token, — the bearer token lifetime.

NOTE 2 It is the responsibility of the accessing party to renew the subscription profile before the bearer token expires.

NOTE 3 See Table 16.

REQ_04_03_04	A subscription shall contain a subscription profile.

REQ_04_03_05	The full subscription profile information can be provided when the subscription is created.

Table 15 — Examples of REQ_04_03_05 providing subscription profile information for refresh token

Normative generic format	POST {baseURI}/{resource}Subscriptions “profile:” “token_type”:”refresh_token” “token”:”{token}” “expires_in”:{lifetime in seconds for refresh token} “tokenEndpoint”:”{accessingPartyTokenEndpoint}” “callbackBaseURI”:”{accessingPartyBaseURI} Response: HTTP 201 Location: {URI of created subscription} Response payload: "profileId": "{Subscription profile id}"

Table 15 *(continued)*

Example	POST {baseURI}/positionSubscriptions?vehicleId=123
	{
	“profile”:
	{
	“token_type”:”refresh_token”,
	“token”:”VGhpcyBpcaCB0b2tlbiwg … cm1hdHRlZA==”,
	“expires_in”: 31536000,
	“tokenEndpoint”: “https://oauth2.example.com/token”,
	“callBackBaseURI”: “https://ap.example.com/exVe”
	}
	}
	
	Subscribe to positions for vehicle with id 123.
	
	Response:
	
	HTTP 201 (created) response returns location header of the new subscription resource (id 2233) and implicitly created subscription profile id in the response body:
	
	Location: {baseURI}/positionSubscriptions/2233
	Response payload: {"profileId": "1234"}

Table 16 — Examples of REQ_04_03_05 providing subscription profile information for bearer token

Normative generic format	POST {baseURI}/{resource}Subscriptions “profile:” “token_type”:”bearer_token” “token”:”{token}” “expires_in”:”{lifetime in seconds for bearer token}” “callbackBaseURI”:”{accessingPartyBaseURI} Response: HTTP 201 Location: {URI of created subscription} Response payload: "profileId": "{subscription profile id}"

Table 16 *(continued)*

Example of subscription containing authorization information and implicit creation of subscription profile	POST {baseURI}/positionSubscriptions?vehicleId=123
	{
	“profile”:
	{
	“token_type”:”bearer_token”,
	“token”:”VGhpcyBpcaCB0b2tlbiwg … cm1hdHRlZA==”,
	“expires_in”:3600,
	“callBackBaseURI”:”https://ap.example.com/exVe”
	}
	}
	
	Subscribe to positions for vehicle with id 123.
	
	Response:
	
	HTTP 201 (created) response returns location header of the new subscription resource (id 2234) and implicitly created subscription profile id in the response body:
	
	Location: {baseURI}/positionSubscriptions/2234
	Response payload: {"profileId": "1235"}

REQ_04_03_06	A subscription profile id (profileId) can be provided when the subscription is created.

NOTE 4 Subscription profiles can be managed by APIs or in web portals.

NOTE 5 See Table 17.

Table 17 — Examples of REQ_04_03_06 providing subscription profile id

Normative generic format	POST {baseURI}/{resource}Subscriptions “profileId”:{profileId}
Example	POST {baseURI}/positionSubscriptions?vehicleId=123 “profileId”:”2233” Subscribe to positions for vehicle with id 123, authorization information is provided in a profile with id 2233.

REQ_04_03_07	HTTP POST can be used by accessing party to create subscription profile for subscriptions.

NOTE 6 See Table 18 and Table 19.

Table 18 — Examples of creating subscription profile for subscriptions regarding REQ_04_03_02 and REQ_04_03_07

Normative generic format	POST {baseURI}/subscriptionProfiles “token_type”:”refresh_token” “token”:”{token}” “expires_in”:”{lifetime in seconds for refresh token}” “tokenEndpoint”:”{accessingPartyTokenEndpoint }” “callbackBaseURI”:”{accessingPartyBaseURI}
Example	POST {baseURI}/subscriptionProfiles Creates subscription profile for subscriptions Request payload: { “token-type”=”refresh_token”, “token”=”RThpcyBpcaCB0b2tlbiwg … cm1hdHRlZA==”, “expires_in”=31557600, “tokenEndpoint”=“https://oauth2.example.com/token”, “callBackBaseURI”=“https://ap.example.com/exVe” } HTTP 201 (created) response returns location header of the new resource: Location: {baseURI}/subscriptionProfiles/2233

Table 19 — Examples of creating subscription profile for subscriptions regarding REQ_04_03_03 and REQ_04_03_07

Normative generic format	POST {baseURI}/subscriptionProfiles “token_type”:”bearer_token” “token”:”{token}” “expires_in”:”{lifetime in seconds for bearer token}” “callbackBaseURI”:”{accessingPartyBaseURI}

Table 19 *(continued)*

Example	POST {baseURI}/subscriptionProfiles Creates subscription profile for subscriptions Request payload: { “token-type”=”bearer_token”, “token”=”BThpcyBpcaCB0b2tlbiwg … cm1hdHRlZZ==”, “expires_in”=3600, “callBackBaseURI”=“https://ap.example.com/exVe” } HTTP 201 (created) response returns location header of the new resource: Location: {baseURI}/subscriptionProfiles/3344

REQ_04_03_08	HTTP GET can be used by accessing party to request a list of subscription profiles.

NOTE 7 See Table 20.

Table 20 — Examples of requesting a list of subscription profiles regarding REQ_04_03_08

Normative generic format	GET {baseURI}/subscriptionProfiles
Example	GET {baseURI}/subscriptionProfiles Request all available subscription profiles for an accessing party. Response payload: {“profiles” : [{“profileId”=”2233”, “token-type”=”refresh_token”, “tokenExpTime”=1611999920, “tokenEndpoint”=“https://oauth2.example.com/token”, “callBackBaseURI”=“https://ap.example.com/exVe” }, {“profileId”=”3344”, “token-type”=”bearer_token”, “tokenExpTime”=1580417120, “callBackBaseURI”=“https://ap.example.com/exVe” }] } Where tokenExpTime = (subscription time + expires_in) as a unix timestamp [8].

REQ_04_03_09	HTTP DELETE can be used by accessing party to delete a subscription profile.

NOTE 8 See Table 21.

Table 21 — Examples of deleting subscription profile regarding REQ_04_03_09

Normative generic format	DELETE {baseURI}/subscriptionProfiles/{profileId}
Example	DELETE {baseURI}/subscriptionProfiles/2233 Delete subscription profile with id=2233.

REQ_04_03_10	The offering party shall inactivate the subscription when the provided authorization has expired.

REQ_04_03_11	The subscription shall be assigned a unique identifier called subscriptionID.

NOTE 9 See Table 22.

Table 22 — Example of subscriptionID regarding REQ_04_03_11

Normative generic format	subscriptionID={unique identifier}
Example	“subscriptionID”=”1eb16206-0179-4ade-b5ea-08b6e6b4ebd5”

REQ_04_03_12	If subscriptions are initiated explicitly, HTTP POST shall be used.

NOTE 10 See Table 23.

Table 23 — Example of subscription regarding REQ_04_03_12

Normative generic format	POST {baseURI}/{resource}Subscriptions
Example	POST {baseURI}/positionSubscriptions?vehicleId=123 Subscribe to positions for vehicle with id 123.

REQ_04_03_13	If subscriptions are initiated explicitly, updating a subscription shall be done using HTTP PUT.

NOTE 11 See Table 24.

Table 24 — Examples of updating a subscription regarding REQ_04_03_13

Normative generic format	PUT {baseURI}/{resource}Subscriptions/{subscriptionId}
Example	PUT {baseURI}/positionSubscriptions/789 Update the position subscription with subscription id 789.
	PUT {baseURI}/positionSubscriptions/567?addVehicleId=234 Extend the existing subscription (Subscription id 567) with positions for the vehicle with id 234 (using query parameters).

Table 24 *(continued)*

	PUT {baseURI}/positionSubscriptions/567 {“position” : [{“vehicleId”=”234”}, {“action”:”ADD”}] } Extend the existing subscription (Subscription id 567) with positions for the vehicle with id 234 (using JSON payload).

REQ_04_03_14	If subscriptions are initiated explicitly, deleting a subscription shall be done using HTTP DELETE.

NOTE 12 See Table 25.

Table 25 — Examples of deleting a subscription regarding REQ_04_03_14

Normative generic format	DELETE {baseURI}/{resource}Subscriptions/{subscriptionId}
Example	DELETE {baseURI}/positionSubscriptions/456 Delete the subscription for positions with subscription id 456.

REQ_04_03_15	The resource shall be pushed using HTTP POST to the baseURI decided by the accessing party.

NOTE 13 See Table 26.

Table 26 — Examples of pushing resources regarding REQ_04_03_15

Normative generic format	POST {accessingPartyBaseURI}/{resource}
Example	POST {accessingPartyBaseURI}/position Push vehicle position according to subscription.

REQ_04_03_16	A subscription shall have a status. The status can be: — ACTIVE (default), — INACTIVE (no resources will be pushed).

REQ_04_03_17	The accessing party should be able to deactivate an existing subscription in order to pause the push of related resources.

REQ_04_03_18	The accessing party shall be able to reactivate an existing subscription in order to resume the push of related resources.

REQ_04_03_19	A subscription can be deactivated by the offering party if the push request returns errors.

REQ_04_03_20	If the offering party has set a subscription to INACTIVE, — reason, — HTTP status code, and — timestamp in the ISO 8601 series of last push attempt shall be provided. Table 27 contains a list of defined reasons.

NOTE 14 See Table 27 for possible reasons.

Table 27 — Reason for deactivation of subscriptions regarding REQ_04_03_20

TOKEN_EXPIRED	Refresh token or bearer token provided in a subscription is expired.
AUTH_ERROR	Push request returns authorization error.
AP_SERVICE_NOT_AVAILABLE	Callback endpoint provided in a subscription is not reachable (after max number of retries).
PUSH_HTTP_STATUS_CODE	HTTP status code returned to push request.
RENEW_TOKEN_ERROR	Cannot get a new access token.
TIMEOUT	Push request takes too long to complete.

NOTE 15 The offering party can define additional reason values.

REQ_04_03_21	HTTP GET can be used by accessing party to request a list of subscriptions.

NOTE 16 See Table 28.

Table 28 — Examples of requesting a list of resource subscriptions regarding REQ_04_03_21

Normative generic format	GET {baseURI}/subscriptions

Table 28 *(continued)*

Example of requesting a list of resource subscriptions	GET {baseURI}/subscriptions Returns all available subscriptions for resources for an accessing party including corresponding profileId. Response payload: {“subscriptions” : [{“subscriptionId”=”1eb16206-0179-4ade-b5ea-08b6e6b4ebd5”, “resource”=”positionSubscriptions”, “profileId”=”2233”, “status”=”ACTIVE” }, {“subscriptionId”=”1eb16206-0179-4ade-b5ea-01b3e6b4ebd0”, “resource”=” odometerSubscriptions”, “profileId”=”2233”, “status”=”INACTIVE”, “reason”=”AUTH_ERROR”, “httpStatusCode”=”402”, “timestamp”=”2020-02-24T09:29:11Z” }] } Inactive subscription contains timestamp of last unsuccessful push request.

4.4 HTTP header fields

REQ_04_04_01	A client shall send a host header field in all HTTP/1.1 request messages, see RFC 7230[16].

NOTE 1 The server name is the same as the “Host” name as outlined in the extended vehicle URI format; see Table 1.

REQ_04_04_02	The HTTP request header fields “Authorization” shall be present in every client HTTP request.

REQ_04_04_03	The HTTP request header field “Accept” shall identify the media type the client accepts in the response from the server.

REQ_04_04_04	The HTTP response header field “Content-Type” shall be present in every server HTTP response with content.

REQ_04_04_05	The HTTP request header field “Authorization” shall use the “Bearer” authentication scheme to transmit a token.

NOTE 2 The generation, format and use of the token is further specified in ISO 20078-3.

4.5 Media types

REQ_04_05_01	The media type: application/json; utf-8 should be supported in HTTP request and response messages.

REQ_04_05_02	Media types that may be supported in HTTP request and response messages are: — text/plain; utf-8, — text/xml; utf-8, — application/octet-stream (for raw binary data).

NOTE 1 Binary data can also be embedded in JSON or XML messages, typically using Base64 encoding.

REQ_04_05_03	At least one media type shall be selected when requesting data.

4.6 Resource versioning

REQ_04_06_01	The resource versioning shall be done on the resource level.

REQ_04_06_02	The API versioning may be done in the URL.

REQ_04_06_03	The resource version shall be identified by a custom parameter exve-resource-version included in the request and response header fields “Accept” and “Content-Type”.

REQ_04_06_04	Syntax of the custom parameter exve-resourceversion shall be: {media type}; exve-resourceversion={usecase resource.}v{major.minor}

NOTE 1 Resource representation with higher major version number is not compatible with lower major version number. For example, resource version v2.0 is not compatible with v1.1.

NOTE 2 Minor version number indicates compatible changes in resource representation of same release. For example, resource version v2.0 is compatible with v2.1.

REQ_04_06_05	As character encoding, utf-8 shall be used in accept-charset or content HTTP headers.

NOTE 3 The use case resource with path is a unique path + name for the resource type.

NOTE 4 The resource version is the version of the resource that the client wants to be returned. It does not have to be the latest available on the server, but the latest the client can handle.

NOTE 5 The requested return format is the format the client wants to have the response in (e.g. JSON, XML), following REQ_04_05_02.

EXAMPLE Media types indicating resource versions, when versioning on resource level:

application/json; exve-resourceversion=resourcereadout.v1.0; charset=utf-8

application/xml; exve-resourceversion=resourcereadout.v1.0; charset=utf-8

REQ_04_06_06	Available versions for each resource shall be documented in the use case specific resources.

REQ_04_06_07	If no version is defined in the accept header, the latest version shall be returned.

4.7 Resources and web services

Resources are exposed through web services. In most cases there is a many-to-many relationship between resources and web services. This subclause outlines how web services and resources relate to each other and how granting of resources is reflected when exposing them through web services.

4.7.1 General

REQ_04_07_01	A web service shall include one or more resources.

REQ_04_07_02	A resource may be included in many web services.

REQ_04_07_03	Granting access to a resource, either directly or through a container, shall give access to that resource only in the web services exposing it.

NOTE 1 If all resources in a web service are granted, full access to the web service is given.

NOTE 2 If no resource in a web service is granted, no access to the web service is given.

NOTE 3 If a subset of the resources in a web service is granted, partial access to the web service is given; there might be cases where no access can be given, due to how the web service is structured.

REQ_04_07_04	The web service specification shall not be changed in any way, due to granting, denying, ignoring, or revoking access of included resources.

4.7.2 Examples

Table 29 shows examples of how resources are exposed in web services and how granting of these resources affects what is accessible.

Table 29 — Resources exposed in web services

Resource: web service	Example	Resource	Web service	Grant	Access
1:1	1	A	A	A	A
	2	B	B	—	—

Table 29 *(continued)*

Resource: web service	Example	Resource	Web service	Grant	Access
N:1	3	A	AB	A	AB
		B		B	
	4	C	CD	C	C (if possible)
		D		—	
	5	E	EF	—	—
		F		—	
1:N	6	G	G’	G	G’
			G’’		G’’
	7	H	H’	—	—
			H’’		—

Table 29 can be selectively explained by:

— example 2: resource B is included in web service B. If resource B is not granted, no access to web service B is allowed;

— example 4: resource C and D are included in the web service CD. If resource C is granted, but not resource D, access to C using the web service CD is allowed (web service CD shall be structured in a way that makes this possible);

— example 6: resource G is included in both web service G’ and G’’. If resource G is granted, full access to both web service G’ and G’’ is allowed.

4.8 Rate limits

To avoid that clients put excessive load on the server-side interface, rate limitation could be imposed on server requests. When rate limitations are applied, a number of HTTP headers are used in order to allow a client to be informed about the current rate limitations for the service.

Rate limits are applied to intervals, in each interval there is a limited amount of requests.

REQ_04_08_01	The web service may apply rate limits.

REQ_04_08_02	If rate limits are applied, then the HTTP header ‘X-Rate-Limit-Limit’ should be used to indicate the request limit within a given time frame.

REQ_04_08_03	If rate limits are applied, then the HTTP header ‘X-Rate-Limit-Remaining’ should be used to indicate remaining available number of requests within the given time frame.

REQ_04_08_04	If rate limits are applied, then the HTTP header ‘X-Rate-Limit-Reset’ should indicate the time in UTC, expressed as seconds since 01.01.1970 (unix timestamp, epoch [8]), when the ‘X-Rate-Limit-Remaining’ is reset to ‘X-Rate-Limit-Limit’.

The following holds as an example, the responses are sent back at 2016-04-06T20:00:00 (1459972800).

NOTE 1 See Table 30.

Table 30 — Examples for different X-Rate-Limit parameters

Resource	X-Rate-Limit-Remaining	X-Rate-Limit-Reset	X-Rate-Limit-Limit
{base_URI}/{resource1}	55	1459973400	60
{base_URI}/{resource2}	2	1459984500	4
{base_URI}/{resource3}	28	1459973460	30

REQ_04_08_05	If rate limits are applied, then the time frames applied for 'X-Rate-Limit-Reset' and the limits for 'X-Rate-Limit-Limit' should be defined by the offering party.

REQ_04_08_06	When the rate limit is exceeded, HTTP 429 shall be returned "Too Many Requests".

4.9 HTTP methods

REQ_04_09_01	The basic HTTP methods POST, GET, PUT and DELETE shall be supported to create, read, update, and delete the resources where applicable.

NOTE 1 The HTTP method PATCH can be used to enable and ease implementation of use cases.

REQ_04_09_02	The HTTP GET method shall be used to read information from an addressed resource on the server.

EXAMPLE 1 GET {base URI}/{resources} HTTP/1.1.

REQ_04_09_03	The HTTP POST method shall be used to create a new resource on the server.

EXAMPLE 2 POST {base URI}/{resources} HTTP/1.1.

REQ_04_09_04	The HTTP PUT method shall be used to change the information for an existing resource on the server.

NOTE 2 Any examples are provided by a selected use case, due to being highly dependent.

EXAMPLE 3 PUT {base URI}/{resources}/{resourceId} HTTP/1.1.

REQ_04_09_05	The HTTP DELETE method shall be used to delete a resource on the server.

EXAMPLE 4 DELETE {base URI}/{resources}/{resourceId} HTTP/1.1.

REQ_04_09_06	The authorization for the HTTP POST, GET, PUT, and DELETE methods on resources is managed by the offering party.

If use cases run into problems regarding concurrent access to resources, conditional requests[2] can be considered as a solution.

EXAMPLE 5 The ETag and If-Match headers can be used to detect conflicts.

For partial updates of resources, the HTTP PATCH method[3] can enable and ease some use cases.

EXAMPLE 6 The two PATCH methods JSON Merge Patch[4] and JavaScript Object Notation (JSON) Patch[5] can be options to implement PATCH for applicable use cases.

4.10 HTTP response status codes

If not otherwise defined, following HTTP response status codes are valid for the offering party and the accessing party.

REQ_04_10_01	Every response message from the web service shall include a status code indicating the result of the request operation. See RFC 7230[6], June 2014, chapter 3.

REQ_04_10_02	The status codes outlined in Tables 22, 23, 24 and 18 shall be supported as a minimum set by the web service.

REQ_04_10_03	Returning a 4xx or 5xx status code on a request shall imply that no further processing of the request will be done at the web service.

Table 31 provides success status codes.

Table 31 — Minimum server-side supported response status codes on success

Code	Reason	Description	Justifications
200	OK	The request succeeded.	A web service will always support an OK response.
201	Created	The request completed, and a new resource was created.	A web service will be able to return this code in the instance that a resource was created successfully.
202	Accepted	The request has been accepted for processing, but not completed.	A web service will be able to return this code to indicate that it has begun a processing task, e.g. a data read request.
204	No Content	The request successfully fulfilled and there is no additional content to send in the response payload body.	A web service will return this code to indicate successful operation without providing any content in the response payload.

Table 32 provides client related error status codes.

Table 32 — Minimum server-side supported response status codes on client error

Code	Reason	Description	Justifications
400	Bad Request	The request could not be completed due to malformed syntax.	A web service will use this response to indicate that the request was in some way malformed and should be corrected before a repeat request is made.
401	Unauthorized	This request requires user authentication to proceed.	A web service will use this response if the request of the client is not correctly authenticated.
403	Forbidden	The request was understood, but was refused.	A web service will use this response if the requesting party was not authorized to continue for whatever reason.
404	Not Found	The requested resource was not found.	A web service will use this response to indicate that the requested resource was not available.
406	Not Acceptable	The resource identified by the request is not capable of generating a response that matches the given accept headers.	A web service will use this response to indicate that a given request was not supported at the level required

Table 32 *(continued)*

Code	Reason	Description	Justifications
429	Too Many Requests	Indicates that the client has sent too many requests to a resource in a given amount of time ("rate limiting").	A web service will use this response to indicate that too many requests were sent to a specific resource for a given amount of time.

Table 33 provides server related error status codes.

Table 33 — Minimum server-side supported response status codes on server error

Code	Reason	Description	Justifications
500	Internal Server Error	The server was unable to complete the request due to an unexpected condition or error.	A web service will indicate that the server encountered an error, and the request was not processed.
501	Not Implemented	The server does not support the functionality required.	A web service will return this response if a function was requested that did not have implemented functionality.
503	Service Unavailable	The server is unable to service the request due to a temporary unavailability condition.	A web service will return this response if the service was unable to process the request for resource-related reasons.
505	Version Not Supported	The server does not or refuses to support the protocol version associated with the request.	A web service will return this response, if the major version of HTTP that was used in the request is not supported.

4.11 Error messaging

Synchronous and asynchronous REST interactions can fail due to various reasons. To enable the client application to narrow down the cause, the web service provides suitable error messages. An error consists of an id (exveErrorId) and a short human readable statement (exveErrorMsg).

To avoid confusion, it should be recognized that depending on the interaction pattern, error messaging can be included in responses to unsuccessful requests (e.g., HTTP 4xx) but can also be involved in some HTTP 200 transmissions.

EXAMPLE 1 If in an asynchronous interaction the request fails after some time.

For successful and unsuccessful interactions, the web service can provide additional information with a 'note' statement (exveNote).

REQ_04_11_01	For error messaging the web service shall include a unique identifier in the message body using the keyword “exveErrorId”.

EXAMPLE 2 “exveErrorId”: "7".

REQ_04_11_02	For error messaging the web service shall include a short human-readable statement in the message body using the keyword “exveErrorMsg”.

EXAMPLE 3 “exveErrorMsg”: "Your request timed out (limit: 120s)".

REQ_04_11_03	To unify and simplify error checking, “exveErrorId” and “exveErrorMsg” keywords shall not be present at all, if a request is fully successful.

NOTE 1 exveErrorId and exveErrorMsg are not set to null or left empty. If a client application finds exveErrorId or exveErrorMsg in a response message body an error occurred.

A partially successful request could return HTTP 200 and error information indicating the status of the request.

REQ_04_11_04	For error messaging the web service should include an error reference in the message body using the keyword “exveErrorRef”. This reference can be used in communication with the offering party to help in resolving problems.

NOTE 2 The keyword exveErrorRef is represented by a UUID; see ISO 20078-1.

EXAMPLE 4 “exveErrorRef”: "848d8c29-c2e6-4a88-8069-8e4e37454814".

REQ_04_11_05	The web service should include further information in the message body of any response (successful or unsuccessful) by using the keyword “exveNote”.

EXAMPLE 5 “exveNote”: "Please retry upon a random time period".

REQ_04_11_06	The elements: “exveErrorMsg”, and “exveNote” shall be plain string literals.

NOTE 3 These elements are not containing any further (child) elements and are not of type; e.g. array or object.

REQ_04_11_07	The web service may return multiple error messages in the same response.

NOTE 4 When sending multiple error messages an array of error message objects can be used.

REQ_04_11_08	The format of error message shall be JSON or XML depending on the content-type of the request header as defined by requirements REQ_04_05_02, REQ_04_06_04 and REQ_04_06_05. For content-type: application/json, the error message shall be in JSON format. For content-type text/xml, the error message shall be in XML format.

NOTE 5 Web service defines supported content-type(s). For example, if only content-type JSON is supported by the web service, the error message will be in JSON format accordingly.

REQ_04_11_09	Error messages shall be in English (en).

EXAMPLE 6 Failed asynchronous response using JSON:

```
{
 "dtcReadout": {
 "id": "abcde-12345-ghjke-67474",
 “asyncStatus": "Fail",
 “exveErrorId": “7”,
“exveErrorRef": “848d8c29-c2e6-4a88-8069-8e4e37454814”,
 “exveErrorMsg": " Your request timed out (limit: 120s)",
 “exveNote": "You did not provide a limit for the amount of results; this was automatically set to 150."  }
}
```

EXAMPLE 7 Failed asynchronous response containing multiple error objects using JSON:


```
{
"dtcReadout": {
"id": "abcde-12345-ghjke-67474",
“asyncStatus": "Fail",
“exveErrors”: [
                { “exveErrorId": “12345”,
                “exveErrorRef”: “848d8c29-c2e6-4a88-8069-8e4e37454814”,
                “exveErrorMsg": "The ECU is not in the correct state",
                “exveNote": "Please try again later"
                 },
                {
                “exveErrorId": “13456”,
                “exveErrorRef”: “848d8c29-c2e6-4a88-8069-8e4e37454815”,
                “exveErrorMsg": ”The ECU ..."
                }
    ]
  }
 }
```


EXAMPLE 8 Successful JSON request with additional note information:


```
{

        "dtcReadout": {
                "id": "abcde-12345-ghjke-67474",
                “asyncStatus": "Complete",
                “exveNote": "You are accessing dtcReadout without versioning information. The API will be upgraded in two months; please see https://{baseURI}/xyz for further information.",
                [... data...]
        }
}
```

EXAMPLE 9 Unsuccessful XML request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ecuReadout>
        <id>abcde-12345-ghjke-67474</id>
        <messageTimestamp>2016-02-24T09:23:46Z</messageTimestamp>
        <exveErrors>
                <exveErrorId>12345</exveErrorId>
                <exveErrorMsg>The ECU is not in the correct state</exveErrorMsg>
                <exveNote>Please try again later</exveNote>
                <exveErrorRef>848d8c29-c2e6-4a88-8069-8e4e37454814</exveErrorRef>
        </exveErrors>
        <exveErrors>
                <exveErrorId>13456</exveErrorId>
                <exveErrorMsg>The ECU ...</exveErrorMsg>
                <exveErrorRef>848d8c29-c2e6-4a88-8069-8e4e37454815</exveErrorRef>
        </exveErrors>
        <vehicleId>12345678909876543</vehicleId>
</ecuReadout>
```

4.12 Interaction pattern

4.12.1 Asynchronous

Some functions require an asynchronous interaction pattern (Figure 6). This is, for example, required when resources need to be read from the connected vehicle rather than being available off-board.

Reading the connected vehicle's current position and speed or the active diagnostics trouble codes could be examples of this. Depending on the architecture and implementation, such operations add a roundtrip delay that is hard to determine due to the connectivity of the vehicle, e.g. network latency, network coverage or vehicle in wrong mode.

To cope with unpredictable response times and avoid keeping the HTTP session open for a long time, the server will respond directly to the client request. Either with the result of the operation or with a location where the client can retrieve (request again) the status of the operation and finally retrieving the result.

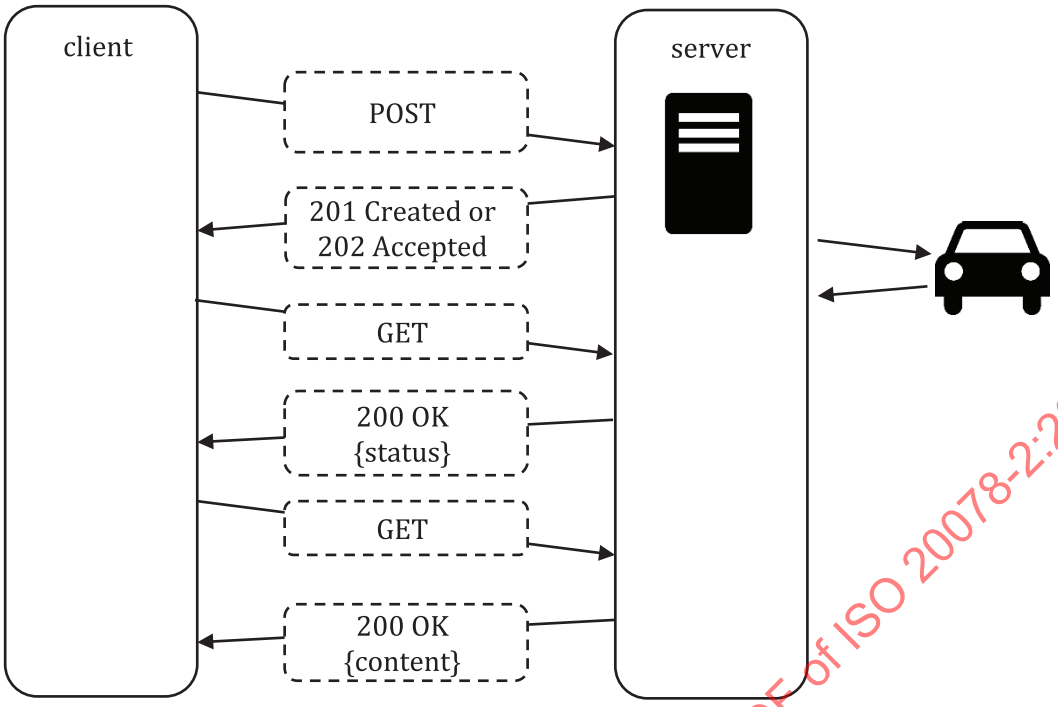

Figure 6 — Asynchronous interaction pattern

REQ_04_12_01	The asynchronous interaction pattern shall start with an HTTP POST request.

REQ_04_12_02	The POST request should include query parameters and/or message body parameters.

EXAMPLE 1 Request the readout of the active DTCs (Diagnostic Trouble Codes) from a connected vehicle: POST {base URI}/vehicles/{vehicleId}/dtcReadouts?dtcStatus=ACTIVE.

NOTE 1 The query parameters and/or message body parameters are web service specific. Available parameters, their format and expected behaviour are described in the web service documentation.

REQ_04_12_03	If the asynchronous request result is available immediately, the response to a POST request shall be “201 Created” and include the result in the message body.

REQ_04_12_04	If the asynchronous request result is not available immediately, the response to a POST request shall be “202 Accepted” and the status of the request shall be available at an URI provided in the HTTP header “Location”.

EXAMPLE 2 A location where the id is typically randomly generated and only valid for this request: {base URI}/vehicles/{vehicleId}/dtcReadouts/{id}.

NOTE 2 The client polls this URI until the result is available or tries to fetch the result at an estimated time of arrival. The intended/preferred access method (continuous frequent polling or finally try to catch results) and the allowed polling frequency is part of the web service documentation.

REQ_04_12_05	A GET request on the returned location shall return the status of the asynchronous request in the message body using the keyword “asyncStatus”.

REQ_04_12_06	The status of the asynchronous request shall be one of the following values: Pending, InProgress, Complete, Fail.

NOTE 3 See Table 34.

Table 34 — List of introduced codes for processing status

Status	Description
Pending	The requested operation has not yet started processing.
InProgress	The requested operation is processing but is not completed.
Complete	The requested operation is complete and has succeeded.
Fail	The requested operation is completed and has failed or timed out.

REQ_04_12_07	A GET request on the returned location should return a recommended waiting time in milliseconds before making the next request. If used, it shall be part of the message body using the keyword “asyncWait”.

REQ_04_12_08	A GET request on the returned location should return the date and time in the ISO 8601 series format when the asynchronous request is estimated to be completed. If used, it shall be part of the message body using the keyword “asyncEstimatedComplete”.

REQ_04_12_09	A GET request on the returned location should return the estimated progress in percentage of the asynchronous request. If used, it shall be part of the message body using the keyword “asyncProgress”.

REQ_04_12_10	A GET request on the returned location may return the date and time in the ISO 8601 series format when the asynchronous request is ended and the related data is made unavailable. If used, it shall be part of the message body using the keyword “asyncRequestEndTime”.

REQ_04_12_11	If the asynchronous request is completed, the message body shall contain the result.

REQ_04_12_12	If the asynchronous request has failed, the message body shall contain the reason.

REQ_04_12_13	If the asynchronous request end time has been exceeded, the response shall be a 404 Not found.

EXAMPLE 3 A response using JSON when the asynchronous request is in progress:

```
{
 "dtcReadout": {
  "id": "abcde-12345-ghjke-67474",
  “asyncStatus": "InProgress",
  “asyncWait”: 10000,
  “asyncEstimatedComple.te”: “2017-05-30T15:35:00Z”,
“asyncProgress”: 50,
  “asyncRequestEndTime”: “2017-06-30T00:00:00Z”
  }
}
```

EXAMPLE 4 A response using JSON when the asynchronous request is completed:{

```
 "dtcReadout": {
  "id": "abcde-12345-ghjke-67474",
  "asyncStatus": "Complete",
  “asyncRequestEndTime”: “2017-06-30T00:00:00Z”,
  "dtcs": [
   {
   "dtcId":"123456",
   "status": "ACTIVE"
    }]
  }
 }
```

EXAMPLE 5 A response using XML:

```
 <dtcReadout>
    <id>abcde-12345-ghjke-67474</id>
     <asyncStatus>InProgress</asyncStatus>
    <asyncWait>10000</asyncWait>
     “asyncRequestEndTime”: “2017-06-30T00:00:00Z”
 </dtcReadout>
```

EXAMPLE 6 A response using plain text:

```
  "id"="abcde-12345-ghjke-67474"
  "asyncStatus"="Complete"
  “asyncRequestEndTime”: “2017-06-30T00:00:00Z”
```

It is also possible to use this pattern to return intermediate or indicative values, before the final result is available. Therefore, it can be used, for example, in measurement protocols (e.g. show a continuous trend of sensor values on a tester).

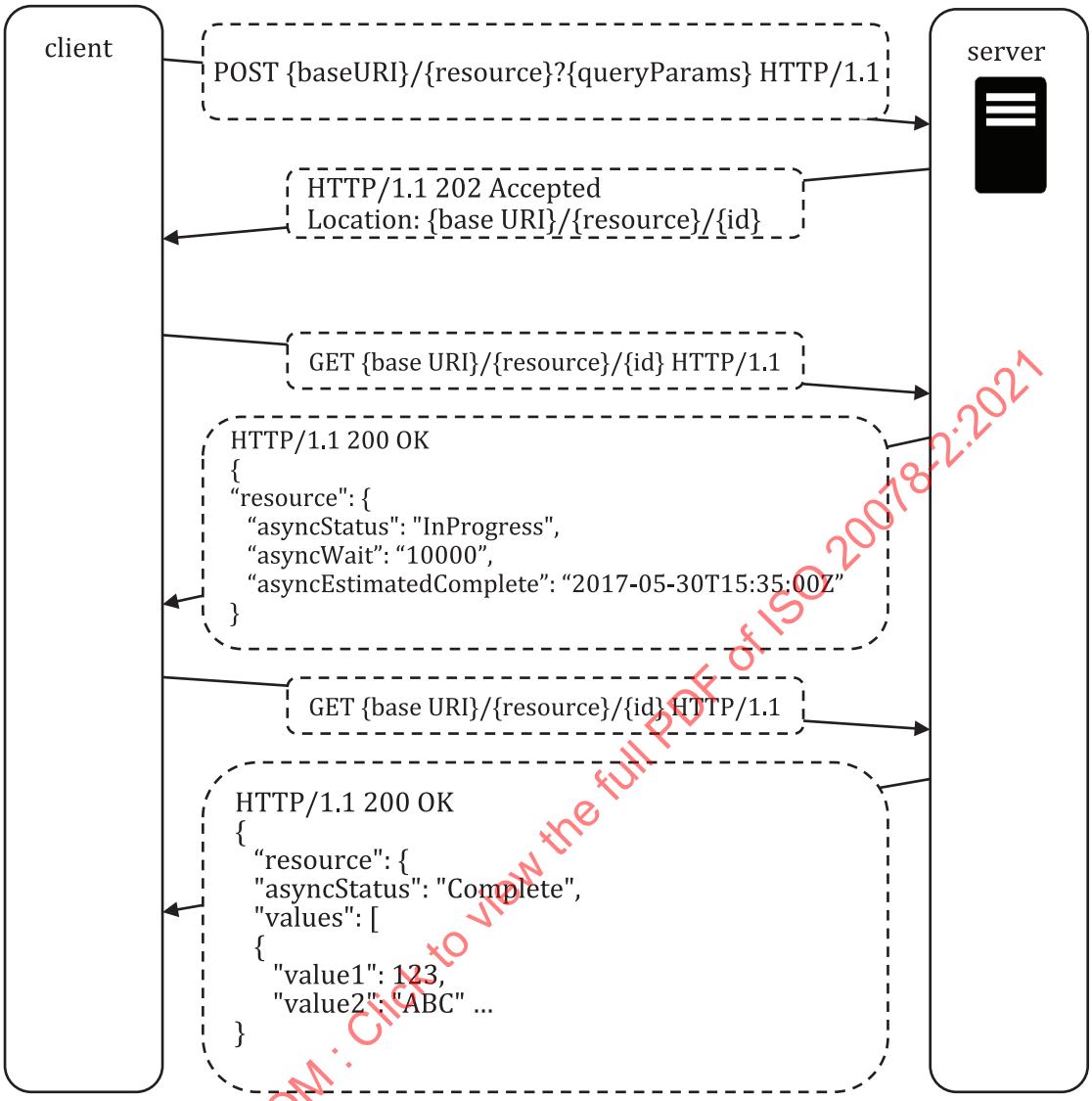

Figure 7 — Example of an asynchronous client server interaction

Figure 7 is showing the detailed asynchronous interaction pattern.

4.13 Resource discovery

Resource discovery is used to list the resources authorized to the accessing party for a specific connected vehicle.

REQ_04_13_01	The authorized resources for a connected vehicle shall be listed in the resource discovery response.

REQ_04_13_02	Each resource listed shall include: resource name, version and a URI to the resource. These should be named ‘name’, ‘version’ and ‘href’.

REQ_04_13_03	The offering party may add own keys to each entry.

EXAMPLE A resource discovery request and response is shown below for a HTTP Request:

GET {base URI}/vehicles/{ID}/resources/

HTTP Response:

Header: HTTP/1.1 200 OK

Body: {

```
"resources": [ {
        "name": "{resource1}",
        "version": "v1.0",
        "href": "{base URI}/vehicles/{ID}/{resource1}"
        }, {
        "name": "{resource2}",
        "version": "v1.0",
        "href": "{base URI}/vehicles/{ID}/{resource2}"
        } ]
}
```

4.14 Capability discovery

Capability discovery is used to list the capabilities to the accessing party for a specific connected vehicle authorized by the resource owner.

REQ_04_14_01	The capabilities for a connected vehicle should be listed in the capability discovery response.

REQ_04_14_02	Each capability listed should include: available resources, version and a URI to the resource. These should be named ‘name’, ‘version’ and ‘href’.

REQ_04_14_03	To access the capability discovery, it shall be authorized by the resource owner.

REQ_04_14_04	An offering party may add own keys to each entry.

EXAMPLE A resource discovery request and response is shown below for a HTTP Request:

```
 GET {base URI}/vehicles/{ID}/capabilities/
HTTP Response:
 Header: HTTP/1.1 200 OK
 Body: {
        "capabilities": [ {
                "name": "{resource1}",
                 "version": "1.0",
                 "href": "{base URI}/vehicles/{ID}/{resource1}"
                 }, {
                 "name": "{resource2}",
                 "version": "1.0",
                 "href": "{base URI}/vehicles/{ID}/{resource2}"
                } ]
        }
```

5 Container management API

5.1 General

Management of containers by the means of an API may facilitate collaboration between the offering party and accessing party at the resource configuration stage as well as during the container lifecycle. The details of a web services interface specified in earlier sections of this document are also applicable to container management API, unless explicitly stated otherwise.

If the offering party chose to provide a container management API, it shall be implemented according to the OpenAPI specification in Annex B.

If the accessing party creates subscriptions for container related push notifications, the accessing party shall implement the appropriate endpoint according to the OpenAPI specification in Annex C.

5.2 Container management operations

This subclause covers basic operations on containers, such as creation, deletion, listing and change of metadata attributes. See Annex A for additional information.

REQ_05_02_01	The offering party should expose an API to list containers available for an accessing party.

NOTE 1 See Table 35.

Table 35 — Examples of retrieving a list of containers regarding REQ_05_02_01

Normative generic format	**GET** {baseURI}/containers

Table 35 *(continued)*

Example	GET {baseURI}/containers Returns all available containers for an accessing party. Sample response: {“containers”: [{“containerId”: ”11f77c4a-652c-4646-a50f-ae6351dcf48f”, “name”: “RemoteDiagnostic”, “purpose”: “Reason for intended usage of included resources”, “status”: “ACTIVE”, “created”: “2020-02-22T02:02:02Z”, “updated”: “2020-05-10T010:00:00Z” }, {“containerId”: ”12f77c4a-652c-4646-a50f-ae6351dcf49a”, “name”: “PAYD”, “purpose”: “Usage-based insurance - Pay As You Drive”, “status”: “ACTIVE”, “created”: “2020-02-22T02:02:02Z”, “updated”: “2020-05-10T010:00:00Z” }] }

REQ_05_02_02	The offering party should expose an API to return container details.

NOTE 2 See Table 36.

Table 36 — Examples of retrieving container details regarding REQ_05_02_02

Normative generic format	**GET** {baseURI}/containers/{containerId}

Table 36 *(continued)*

Example	GET {baseURI}/containers/11f77c4a-652c-4646-a50f-ae6351dcf48f HTTP 200 response with details for containerId 11f77c4a-652c-4646-a50f-ae6351dcf48f: {“containerId”: ”11f77c4a-652c-4646-a50f-ae6351dcf48f”, “name”: “RemoteDiagnostic”, “purpose”: “Remote diagnostic support”, “status”: “ACTIVE”, “created”: “2020-02-22T02:02:02Z”, “updated”: “2020-05-10T010:00:00Z”, “resources”: [{“resourceId”: “23a8065f-7fab-4a54-be42-b6bf360e34cf”, “resourceName”: ”DTC Readout” }, {“resourceId”: “dcf5906c-096c-4169-81cc-8ce4ce4d388b”, “resourceName”: ”ECU Readout”}] }

REQ_05_02_03	The offering party should expose an API for container creation.

NOTE 3 See Table 37.

Table 37 — Examples of container creation regarding REQ_05_02_03

Normative generic format	**POST** {baseURI}/containers

Table 37 *(continued)*

Example	POST {baseURI}/containers
	
	Sample request payload:
	{“name”: “Vehicle Status”,
	“purpose”: “Remote services”,
	“resources”: [
	{“resourceId”: ”123”},
	{“resourceId”: ”124”}
	]
	}
	
	Sample response with HTTP status code 201:
	Payload:
	{
	"containerId": "2a53b196-e04f-4857-92cb-9dbcb5dcb956",
	“name”: “Vehicle status”,
	“purpose”: “Remote services”,
	“status”: “ACTIVE”,
	“created”: “2020-02-20T02:02:02Z”,
	“resources”: [
	{“resourceId”: ”123”},
	{“resourceId”: ”124”}
	]
	}
	Location header:
	{baseURI}/containers/2a53b196-e04f-4857-92cb-9dbcb5dcb956

REQ_05_02_04	The offering party should expose an API to inactivate and activate containers.

NOTE 4 Deactivation of a container neither deletes an assignment of VINs to this container nor revokes resource owners consent. It suspends the usage of container related resources including subscriptions without any changes of the container definition.

NOTE 5 The container could be reactivated at any time. From the activation time point, the offering party will start to provide again all related resources as was defined at the creation of the container. Resource data before activation time point will not be available.

NOTE 6 See Table 38.

Table 38 — Example of container status changes regarding REQ_05_02_04

Normative generic format	**PATCH** {baseURI}/containers/{containerId}

Table 38 *(continued)*

Example	**PATCH** {baseURI}/containers/2a53b196-e04f-4857-92cb-9dbcb5dcb956
	
	The requester shall provide status attributes together with its desired state in the payload.
	
	Sample request payload:
	{ “status”:”ACTIVE”}
	
	Sample response payload with HTTP status 200:
	{
	"containerId": "2a53b196-e04f-4857-92cb-9dbcb5dcb956",
	“name”: “Vehicle status”,
	“purpose”: “Remote services”,
	“status”: ”ACTIVE”,
	“created”: “2020-02-20T02:02:02Z”,
	“updated”: “2020-05-10T010:00:00Z”,
	“resources”: [{“resourceId”:”123”}, {“resourceId”:”124”}]
	}
	
	Response with HTTP status code 204 does not return any payload indicating that status of the container was already as requested.

REQ_05_02_05	The offering party should expose an API for deletion of containers.

NOTE 7 See Table 39.

Table 39 — Example of container deletion regarding REQ_05_02_05

Normative generic format	**DELETE** {baseURI}/containers/{containerId}
Example	DELETE {baseURI}/containers/2a53b196-e04f-4857-92cb-9dbcb5dcb956 The containerId is specified as a path parameter in the request. Response with HTTP status code 204 does not return any payload indicating the request was successfully processed.

A container is used to get access to resources of one or more vehicles. Vehicle association API allows to request a list of vehicles assigned to one container and to add or delete the association.

REQ_05_02_06	The offering party should expose an API for associating vehicles with a container.

NOTE 8 See Table 40.

Table 40 — Examples of vehicle association regarding REQ_05_02_06

Normative generic format	**POST** {baseURI}/containers/{containerId}/vehicles

Table 40 *(continued)*

Example	POST {baseURI}/containers/2a53b196-e04f-4857-92cb-9dbcb5dcb956/vehicles
	
	The request payload shall include vehicle parameter enumerating all the vehicles to be associated with the container:
	
	{
	“vehicles”: [
	{“vehicleId”: ”XXXXXX123456780”},
	{“vehicleId”: ”XXXXXX123456781”},
	{“vehicleId”: ”XXXXXX123456789”}
	]
	}

REQ_05_02_07	The offering party should expose an API for retrieving vehicles associated with a container.

NOTE 9 See Table 41.

Table 41 — Example of retrieving vehicles assigned to a container regarding REQ_05_02_07

Normative generic format	**GET** {baseURI}/containers/{containerId}/vehicles

Example	GET {baseURI}/containers/2a53b196-e04f-4857-92cb-9dbcb5dcb956/vehicles
	
	The response shall include list of vehicles associated with the container, with vehicle identifier and consent status for each.
	
	Sample response with HTTP status code 200:
	{
	“containerId”: “2a53b196-e04f-4857-92cb-9dbcb5dcb956”,
	“vehicles”: [
	{“vehicleId”:”XXXXXX123456780”,”consentStatus”: ”PENDING”},
	{“vehicleId”:”XXXXXX123456781”,”consentStatus”: ”GRANTED”},
	{“vehicleId”:”XXXXXX123456789”,”consentStatus”: ”REJECTED”},
	{“vehicleId”:”XXXXXX123456791”,”consentStatus”: ”REVOKED”}
	]
	}

REQ_05_02_08	The offering party may expose an API for removing a vehicle from a container.

NOTE 10 See Table 42.

Table 42 — Example of removing vehicle from a container regarding REQ_05_02_08

Normative generic format	**DELETE** {baseURI}/containers/{containerId}/vehicles/{vehicleId}
Example	DELETE {baseURI}/containers/2a53b196-e04f-4857-92cb-9dbcb5dcb956/vehicles/XXXXXX123456780 Remove assignment of vehicleId XXXXXX123456780 to containerId 2a53b196-e04f-4857-92cb-9dbcb5dcb956 Response with HTTP status code 204 does not return any payload indicating the request was successfully processed.

REQ_05_02_09	Offering party may expose an API for bulk removal vehicles from a container.

NOTE 11 See Table 43.

Table 43 — Example of removing multiple vehicles from a container regarding REQ_05_02_09

Normative generic format	**POST** {baseURI}/containers/{containerId}/vehiclesToRemove

Example	POST {baseURI}/containers/2a53b196-e04f-4857-92cb-9dbcb5dcb956/vehiclesToRemove
	
	The request shall include vehicle parameter enumerating all the vehicles to be removed from the container.
	
	Sample request payload
	{
	“vehicles”: [{“vehicleId”:”XXXXXX123456789”}, {“ vehicleId”:”XXXXXX123456780”}]
	}

NOTE 12 An alternative approach for bulk delete is to use DELETE verb and transfer the list of VINs in the payload. However, the use of DELETE verb and payload together is not encouraged by current engineering practices and it is possible that it is not supported by infrastructure components.

REQ_05_02_10	The offering party may expose an API for subscriptions to container related notifications. Notifications can be triggered by following events: — CONSENT_GRANTED — CONSENT_REVOKED — RESOURCE_DATA_UPDATED Table 44 contains description of defined events.

NOTE 13 Usage of subscriptions is described in this document, see 4.3.

NOTE 14 See Tables 44 to 47.

Table 44 — Explanation of notification events regarding REQ_05_02_10

CONSENT_GRANTED	Resource owner has provided consent for usage of resources as defined by a certain container.

Table 44 *(continued)*

CONSENT_REVOKED	Resource owner has revoked consent. Change of resource owner can also lead to revocation of the consent.
RESOURCE_DATA_UPDATED	New container related resource data is available at the ExVe backend. The accessing party can request the data using appropriate REST interface.

Table 45 — Example of subscription to container related notifications regarding REQ_05_02_10

Normative generic format	**POST** {baseURI}/containerEventSubscriptions

Example	POST {baseURI}/containerEventSubscriptions
	
	Sample request payload
	{
	“profileId”: “2233”,
	“containers”: [
	{“containerId”: “2a53b196-e04f-4857-92cb-9dbcb5dcb956”,
	“events”: [
	{”event”: ”CONSENT_GRANTED”},
	{”event”: ”CONSENT_REVOKED”},
	{”event”: ”RESOURCE_DATA_UPDATED”}
	]
	}
	]
	}
	
	Instead of profileId, the accessing party can provide authorization information directly in the request as defined in requirement REQ_04_03_01 above.
	
	HTTP 201 (created) response returns location header of the new subscription:
	Location: {baseURI}/containerEventSubscriptions/1234

Table 46 — Example of updating a subscription to container related notifications regarding REQ_05_02_10

Normative generic format	**PUT** {baseURI}/containerEventSubscriptions/{subscriptionId}

Table 46 *(continued)*

Example	PUT {baseURI}/containerEventSubscriptions/1234
	
	Sample request payload
	{
	“profileId”: “2244”,
	“containers”: [
	{“containerId”: “2a53b196-e04f-4857-92cb-9dbcb5dcb956”,
	“events”: [
	{”event”: ”CONSENT_GRANTED”},
	{”event”: ”CONSENT_REVOKED”}
	]
	}
	]
	}
	
	Updates existing container event subscription with id=1234:
	— Subscription profile id changed,
	— event RESOURCE_DATA_UPDATED unsubscribed.

NOTE 15 PUT will replace entire subscription or create new one (if not yet exists), so client will always know the complete state of the resource. Empty request payload will be ignored.

Table 47 — Example of a push notification regarding REQ_05_02_10

Normative generic format	**POST** {accessingPartyBaseURI}/containerEvents

Example payload of the push notification	POST https://ap.example.com/exVe/containerEvents
	
	Sample payload
	{
	“containerEvents”: [
	{“containerId”: “2a53b196-e04f-4857-92cb-9dbcb5dcb956”,
	“vehicleEvents”: [
	{“vehicleId”:”XXXXXX123456780”,”event”: ”CONSENT_GRANTED”},
	{“vehicleId”:”XXXXXX123456780”,”event”: ”RESOURCE_DATA_UPDATED”}
	]
	}
	]
	}

REQ_05_02_11	The offering party may expose an API to list container subscriptions.

NOTE 16 See Table 48.

Table 48 — Example of requesting a list of container related subscriptions regarding REQ_05_02_11

Normative generic format	**GET** {baseURI}/containerEventSubscriptions?containerId={containerId}

Example	GET {baseURI}/containerEventSubscriptions
	
	The response contains list of available containers and event subscriptions. Query parameter containerId is optional. If query parameter containerId is not provided, response payload contains a list of event subscriptions of all available containers.
	
	Sample response with HTTP status code 200:
	{
	containerEventSubscriptions: [
	{“containerId”: “2a53b196-e04f-4857-92cb-9dbcb5dcb956”,
	“subscriptions”: [
	{“subscriptionId”:”1234”,
	”profileId”: ”2233”,
	“events”:[
	{”event”: ”CONSENT_GRANTED”},
	{”event”: ”CONSENT_REVOKED”},
	{”event”: ”RESOURCE_DATA_UPDATED”}
	]
	}
	]
	}
	]
	}

REQ_05_02_12	The offering party may expose an API to delete container subscriptions.

NOTE 17 See Table 49.

Table 49 — Example of deleting a container related subscription regarding REQ_05_02_12

Normative generic format	**DELETE** {baseURI}/containerEventSubscriptions/{subscriptionId}
Example	DELETE {baseURI}/containerEventSubscriptions/1234 Deletes subscription with id 1234 Response with HTTP status code 204 does not return any payload indicating the request was successfully processed.

Annex A
(informative)

Container management API specification

A.1 Introduction

A.1.1 General

This annex contains detail specification of container management API described in this document. The web service specification is based on the ISO 20078:2021 series. All web services are defined as REST APIs, using JSON for the transfer of content.

The ISO 20078:2021 series is indispensable for implementing web services according to Annex A.

A.1.2 Security

All REST APIs are using OAuth2 compatible framework for access control and https for securing the transfer, see ISO 20078-3:2021 for details.

The exact details of how to obtain access is either described by offering party for ExVe resources or by accessing party for push interfaces.

A.1.3 Error codes

A.1.3.1 ISO 20078

The HTTP status codes (error codes) listed for each REST API are described in 4.10.

A.2 Container management REST APIs

A.2.1 API overview

Table A.1 gives an overview of container management REST APIs specified below in this document.

Table A.1 — Overview of container management REST APIs

Use case / 20078-2 requirement		Method and path	API reference
Request container data	Retrieve a list of containers REQ_05_02_01	GET {baseURI}/containers	Table A.2
	Retrieve container details REQ_05_02_02	GET {baseURI}/containers/{containerId}	Table A.3
	Retrieve vehicles associated with a container REQ_05_02_07	GET {baseURI}/containers/{containerId}/vehicles	Table A.4

Table A.1 *(continued)*

Use case / 20078-2 requirement		Method and path	API reference
Container creation and modification	Create a container REQ_05_02_03	POST {baseURI}/containers	Table A.5
	Remove a container REQ_05_02_05	DELETE {baseURI}/containers/{containerId}	Table A.6
	Change container status REQ_05_02_04	PATCH {baseURI}/containers/{containerId}	Table A.7
	Associate vehicles with a container REQ_05_02_06	POST {baseURI}/containers/{containerId}/vehicles	Table A.8
	Remove vehicle from a container REQ_05_02_08	DELETE {baseURI}/containers/{containerId}/vehicles/{vehicleId}	Table A.9
	Remove multiple vehicles from a container REQ_05_02_09	POST {baseURI}/containers/{containerId}/vehiclesToRemove	Table A.10
Container event subscriptions	Subscribe to container related notifications REQ_05_02_10	POST {baseURI}/containerEventSubscriptions	Table A.11
	List subscriptions for a container REQ_05_02_11	GET {baseURI}/containerEventSubscriptions?containerId={containerId}	Table A.12
	Update subscription for a container REQ_05_02_10	PUT {baseURI}/containerEventSubscriptions/{subscriptionId}	Table A.13
	Delete container subscription REQ_05_02_12	DELETE {baseURI}/containerEventSubscriptions/{subscriptionId}	Table A.14
Container push notification	Push notification REQ_05_02_10	POST {accessingPartyBaseURI}/containerEvents	Table A.15

A.2.2 Description of container management REST APIs

Table A.2 describes the operation to retrieve a list of containers.

Table A.2 — Retrieve a list of containers

Generic format	**GET** {baseURI}/containers

Table A.2 *(continued)*

Request headers	**Header**	**Usage**	**Value**	
	Authorization	required	Bearer {token}	
	Accept	required	application/json; exve-resourceversion=container.v1.0; charset=utf-8	
Response payload	**Attribute**	**Data type**	**Usage**	**Description**
	Containers	array	required	List of containers
	For each container:			
	containerId	string	required	Unique identifier of a container
	name	string	required	Name of the container
	purpose	string	required	Intended purpose of data usage
	status	string	required	Status of the container: ACTIVE, INACTIVE
	created	string	required	Timestamp in the ISO 8601 series date-time format
	updated	string	required	Timestamp in the ISO 8601 series date-time format
	The offering party can provide additional attributes in the response payload.			
Example	GET {baseURI}/containers Returns all available containers for an accessing party. Sample response: {“containers”: [{“containerId”: ”11f77c4a-652c-4646-a50f-ae6351dcf48f”, “name”: “RemoteDiagnostic”, “purpose”: “Reason for intended usage of included resources”, “status”: “ACTIVE”, “created”: “2020-02-22T02:02:02Z”, “updated”: “2020-05-10T01:00:00Z” }, {“containerId”: ”12f77c4a-652c-4646-a50f-ae6351dcf49a”, “name”: “PAYD”, “purpose”: “Usage-based insurance - Pay As You Drive”, “status”: “ACTIVE”, “created”: “2020-02-22T02:02:02Z”, “updated”: “2020-05-10T01:00:00Z” }] }			

Table A.2 *(continued)*

HTTP status codes	200 OK 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 406 Not Acceptable 500 Internal Server Error 501 Not Implemented 503 Service Unavailable 505 HTTP Version not supported

Table A.3 describes the operation to retrieve container details.

Table A.3 — Retrieve container details

Generic format	**GET** {baseURI}/containers/{containerId}			
Request URI parameters	**Parameter**	**Data type**	**Usage**	**Description**
	containerId	string	required	Unique identifier of a container
Request headers	**Header**	**Usage**	**Value**	
	Authorization	required	Bearer {token}	
	Accept	required	application/json; exve-resourceversion= container.v1.0; charset=utf-8	
Response payload	**Attribute**	**Data type**	**Usage**	**Description**
	containerId	string	required	Unique identifier of a container
	name	string	required	Name of the container
	purpose	string	required	Intended purpose of data usage
	status	string	required	Status of the container: ACTIVE, INACTIVE
	created	string	required	Timestamp in the ISO 8601 series date-time format
	updated	string	required	Timestamp in the ISO 8601 series date-time format
	resources	array	required	List of ExVe resources assigned to the Container
	For each resource:			
	resourceId	string	required	Unique identifier of a resource
	resource-Name	string	required	Name of the resource
	The offering party can provide additional attributes in the response payload.			

Table A.3 *(continued)*

Example	GET {baseURI}/containers/11f77c4a-652c-4646-a50f-ae6351dcf48f HTTP 200 response with details for containerId 11f77c4a-652c-4646-a50f-ae6351dcf48f: {“containerId”: ”11f77c4a-652c-4646-a50f-ae6351dcf48f”, “name”: “RemoteDiagnostic”, “purpose”: “Remote diagnostic support”, “status”: “ACTIVE”, “created”: “2020-02-22T02:02:02Z”, “updated”: “2020-05-10T010:00:00Z” “resources”: [{“resourceId”: “23a8065f-7fab-4a54-be42-b6bf360e34cf”, “resourceName”: ”DTC Readout” }, {“resourceId”: “dcf5906c-096c-4169-81cc-8ce4ce4d388b”, “resourceName”: ”ECU Readout”}] }
HTTP status codes	200 OK 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 406 Not Acceptable 500 Internal Server Error 501 Not Implemented 503 Service Unavailable 505 HTTP Version not supported

Table A.4 describes the operation to retrieve vehicles assigned to a container.

Table A.4 — Retrieve vehicles assigned to a container

Generic format	**GET** {baseURI}/containers/{containerId}/vehicles			
Request URI parameters	**Parameter**	**Data type**	**Usage**	**Description**
	containerId	string	required	Unique identifier of a container
Request headers	**Header**	**Usage**	**Value**	
	Authorization	required	Bearer {token}	
	Accept	required	application/json; exve-resourceversion= container.v1.0; charset=utf-8	

Table A.4 *(continued)*

Response payload	**Attribute**	**Data type**	**Usage**	**Description**
	containerId	string	required	Unique identifier of a container
	vehicles	array	required	List of vehicles assigned to the container
	For each vehicle:			
	vehicleId	string	required	Vehicle identification number
	consentStatus	string	required	Status of user's consent: PENDING, GRANTED, REJECTED, REVOKED
	The offering party can provide additional attributes in the response payload.			

Example	GET {baseURI}/containers/2a53b196-e04f-4857-92cb-9dbcb5dcb956/vehicles
	
	The response shall include list of vehicles associated with the container, with vehicle identifier and consent status for each.
	
	Sample response with HTTP status code 200:
	{
	"containerId": "2a53b196-e04f-4857-92cb-9dbcb5dcb956",
	"vehicles": [
	{"vehicleId":"XXXXXX123456780","consentStatus": "PENDING"},
	{"vehicleId":"XXXXXX123456781","consentStatus": "GRANTED"},
	{"vehicleId":"XXXXXX123456789","consentStatus": "REJECTED"},
	{"vehicleId":"XXXXXX123456791","consentStatus": "REVOKED"}
	]
	}			

HTTP status codes	200 OK 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 406 Not Acceptable 500 Internal Server Error 501 Not Implemented 503 Service Unavailable 505 HTTP Version not supported			

Table A.5 describes the operation to create a container.

Table A.5 — Create a container

Generic format	**POST** {baseURI}/containers

Table A.5 *(continued)*

Request headers	**Header**	**Usage**	**Value**	
	Authorization	required	Bearer {token}	
	Accept	required	application/json; exve-resourceversion= container.v1.0; charset=utf-8	
	Content-type	required	application/json; exve-resourceversion= container.v1.0; charset=utf-8	
Request payload	**Attribute**	**Data type**	**Usage**	**Description**
	name	string	required	Name of the container
	purpose	string	required	Intended purpose of data usage
	resources	array	required	List of ExVe resources
	For each resource:			
	resourceId	string	required	Unique identifier of a resource
Response header	**Header**	**Usage**	**Value**	
	Location	required	Absolute URI of the created resource	
Response payload	**Attribute**	**Data type**	**Usage**	**Description**
	containerId	string	required	Unique identifier of a container
	name	string	required	Name of the container
	purpose	string	required	Intended purpose of data usage
	status	string	required	Status of the container: ACTIVE, INACTIVE
	created	string	required	Timestamp in the ISO 8601 series date-time format
	resources	array	required	List of ExVe resources
	For each resource:			
	resourceId	string	required	Unique identifier of a resource
	The offering party can provide additional attributes in the response payload.			

Table A.5 *(continued)*

Example	POST {baseURI}/containers
	
	Sample request payload:
	{“name”: “Vehicle Status”,
	“purpose”: “Remote services”,
	“resources”: [
	{“resourceId”: ”123”},
	{“resourceId”: ”124”}
	]
	}
	
	Sample response with HTTP status code 201:
	{
	"containerId": "2a53b196-e04f-4857-92cb-9dbcb5dcb956",
	“name”: “Vehicle status”,
	“purpose”: “Remote services”,
	“status”: “ACTIVE”,
	“created”: “2020-02-20T02:02:02Z”,
	“resources”: [
	{“resourceId”: ”123”},
	{“resourceId”: ”124”}
	]
	}
	
	Location header:
	{baseURI}/containers/2a53b196-e04f-4857-92cb-9dbcb5dcb956

HTTP status codes	201 Created 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 406 Not Acceptable 500 Internal Server Error 501 Not Implemented 503 Service Unavailable 505 HTTP Version not supported

Table A.6 describes the operation to delete a container.

Table A.6 — Delete a container

Generic format	**DELETE** {baseURI}/containers/{containerId}

Table A.6 *(continued)*

Request parameters	**Parameter**	**Data type**	**Usage**	**Description**
	containerId	string	required	Unique identifier of a container
Request headers	***Header***	***Usage***	***Value***	
	Authorization	required	Bearer {token}	
Example	DELETE {baseURI}/containers/2a53b196-e04f-4857-92cb-9dbcb5dcb956 The containerId is specified as a path parameter in the request. Response with HTTP status code 204 does not return any payload indicating the request was successfully processed.			
HTTP status codes	204 No Content 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 406 Not Acceptable 500 Internal Server Error 501 Not Implemented 503 Service Unavailable 505 HTTP Version not supported			

Table A.7 describes the operation to change container status.

Table A.7 — Change container status

Generic format	**PATCH** {baseURI}/containers/{containerId}			
Request URI parameters	**Parameter**	**Data type**	**Usage**	**Description**
	containerId	string	required	Unique identifier of a container
Request headers	**Header**	**Usage**	**Value**	
	Authorization	required	Bearer {token}	
	Accept	required	application/json; exve-resourceversion= container.v1.0; charset=utf-8	
	Content-type	required	application/json; exve-resourceversion= container.v1.0; charset=utf-8	
Request payload	**Attribute**	**Data type**	**Usage**	**Description**
	status	string	required	Status of the container: ACTIVE, INACTIVE