EXLAP

Extensible Lightweight Asynchronous Protocol

Specification version 1.3

EXLAP - Extensible Lightweight Asynchronous Protocol Specification Version 1.3 von Volkswagen AG steht unter einer Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland Lizenz.
(http://creativecommons.org/licenses/by-sa/3.0/de/)

Table of contents

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

List of figures

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

List of Tables

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

List of sequence charts

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

1 Document

1.1 Purpose of document

This document describes EXLAP, the Extensible Lightweight Asynchronous Protocol, a communication protocol used for the communication between a client and a server on the wire. In addition, the document describes how this protocol is to be used on top of the internet protocol (IP), how data types are represented and services can be described.

1.2 Authors and reviewers

Authors and reviewers of this document.

Name	Initials	E-mail
Hans-Christian Fricke	HCF	hans-christian.fricke@volkswagen.de
Jens Krüger	JK	jens2.krueger@volkswagen.de

1.3 Related documents

[1] Extensible Markup Language (XML) 1.0 (Fourth Edition)
http://www.w3.org/TR/2006/REC-xml-20060816/

[2] XML Schema schema for XML Schemas: Part 2: Datatypes
http://www.w3.org/2001/XMLSchema-datatypes

[3] XML Schema Part 2: Datatypes Second Edition
http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[4] UTF-8, a transformation format of ISO 10646
http://tools.ietf.org/html/rfc3629

[5] The WebSocket Protocol (RFC 6455)
http://tools.ietf.org/html/rfc6455

[6] Hypertext Transport Protocol – HTTP/1.1 (RFC 2616)
http://tools.ietf.org/html/rfc2616

[7] Bluetooth Core 4.0 and SPP 1.2 Specification
http://www.bluetooth.org/Technical/Specifications/adopted.htm

[8] Basic and Digest Access Authorization, RFC 2617
http://tools.ietf.org/html/rfc2617

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

2 Overview

2.1 Extensible lightweight asynchronous Protocol (EXLAP)

2.1.1 The scope of EXLAP

EXLAP is a domain specific application protocol to transport service oriented data in a human readable format over the "wire".

The protocol follows the client-server communication paradigm and supports both, publish/subscribe and asynchronous remote procedure calls (ARPC), for communication. All data definitions are based upon XML which incorporates the protocol, the encoding on the wire, the data types and an interface definition language (IDL) for services that use ARPC style function calls and asynchronous messaging.

In addition EXLAP permits different underlying transport layers, as it only requires a serial link between the client (consumer) and the server (service provider). Therefore it is possible to implement EXLAP using TCP/IP sockets, on top of the Bluetooth serial profile or any other programmable serial interface (e.g. RS 232).

2.1.2 History and motivation of EXLAP

The EXLAP protocol was initially created as internal protocol within the Corporate Research of the Volkswagen AG to transport vehicle originated and event driven data in an abstract, vehicle and vendor independent, manner to data consuming clients in the IT domain (e.g. mobile devices). Leveraging the initial effort, the protocol was extended in version 1.1 with RPC functionality and consolidated for publication in version 1.2.

The current goal of the specification is to extend the applicability of EXLAP to all concerns of service communication within the infotainment part of a vehicle and equivalent fields. Version 1.3 introduced some minor extensions and further consolidation efforts.

2.1.3 The core characteristics of EXLAP

Based on the motivation and the requirements the development of the EXLAP protocol resulted in the following core characteristics that guided the protocol development:

- Human readable protocol “on the wire”.
  - More compact and more readable then WebServices, SOAP, REST, CORBA, etc.
- Client-server paradigm for communication.
  - Reflects the notion of service provider and service consumer.
  - Allows for the protocol to work over a single serial connection (TCP, BT SPP, SPI)
- Both, publish/subscribe and asynchronous RPC in the core protocol.
  - To allow for event driven and procedural use-cases without polling
- Service oriented.
  - Compact service specifications can be done with a XML based IDL.
  - Describing signatures of functions and data structures.
  - Allowing for subsequent code and documentation generation.
- Usage of an abstract and compact data type universe.
  - Seven elementary (basic) data types.
    - Absolute (any number, floating point or integer, within a min/max range).
    - Activity (a boolean value).
    - Binary (an array of byte).
    - Enumeration (a value out of a predefined set).
    - Relative (a value between 0.0 and 1.0).
    - Text (a sequence of characters, limited by a regular expression).
    - Time (a date with time and time zone).
  - Two composite (complex) types.
    - Alternative (in between an enumeration of objects and a C style union) .
    - Objects (a composition out of basic and complex types).
    - Lists (of objects).

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

To realize these goals EXLAP relies on the use of existing XML standards (i.e. XML [1], XML data types [2][3], UTF-8 encoding). The usage of XML and XML Schema allow for high formal deepness (i.e. using available XML editors and XML testing tools), and in addition the simple processing of the transported data by leveraging the native XML processing capabilities of today's platforms.

2.2 Protocol architecture

EXLAP is a simple message based asynchronous client/server request/response protocol that uses transaction numbers for synchronization of response envelopes (<*Rsp/*>) to request envelopes (<*Req/*>). In addition unidirectional datagram (<*Dat/*>) and status (<*Status/*>) message envelopes are used for the event driven communication and signalling from server to client.

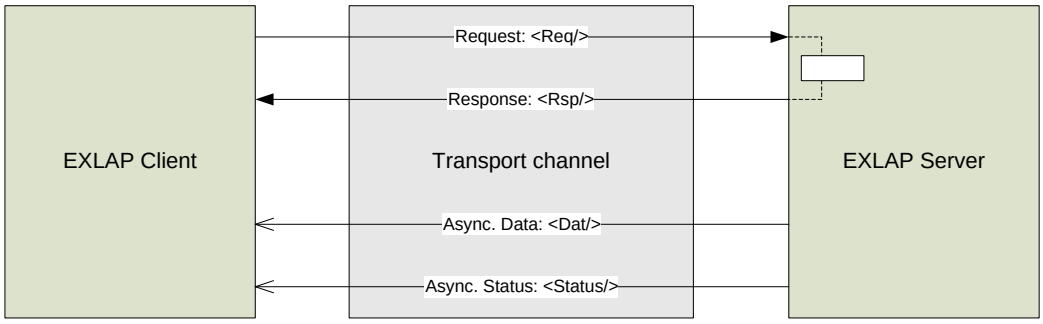


Figure 1: EXLAP architecture and communication overview

The messages, exchanged between client and server, are represented in XML by four element envelopes: <*Req/*>, <*Rsp/*>, <*Dat/*> and <*Status/*>. Each element features a subset of additional elements and attributes that are governed by an XML schema (see Section 6.3 for details).

2.2.1 The server role

The EXLAP server is the information providing service component (e.g. Tuner, Vehicle, Amplifier, etc.). The server implements a service specification, which is comprised by a set of functions and/or data objects that provide access to static, dynamic or event driven information.

2.2.2 The client role

The EXLAP client is the actor, an information consuming or controlling component (e.g. an external mobile device), that accesses the service by subscribing and/or requesting information.

2.2.3 Transport protocol layer binding and discovery

To transport data reliable from the server to the client and vice versa, EXLAP relies on an underlying transport layer. While EXLAP is in principal independent of the underlying transport layer still some definitions have to take place to build real-world systems, since connection setup, start and termination and discovery of communications partners have to be defined, as those often depend on the underlying bearer technology.

Currently the following binding recommendations are defined:

Protocol	Comments
TCP/IP	Using sockets, including a discovery protocol suitable even for Java 2 Micro Edition environments.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

WebSockets	Enables the consumption of EXLAP services by web applications and web browsers.
Bluetooth	Based on the Serial Port Profile with EXLAP specific service UUID for each service.

Table 1: Available transport protocol binding definitions

To dynamically discover services that are accessible via EXLAP, the protocol suite defines a recommendation for most transport protocols that are listed in Table 1. Since some transport layers, like Bluetooth, already have a definition of how services are announced, discovery and used, those mechanisms can be used. For more detailed information Section 5 provides the recommendations for the TCP/IP, the WebSocket and the Bluetooth binding.

2.2.4 Session state and scalability

The session state of client/server connections in regard to active subscriptions is bound to the underling physical connection - if the underling connection terminates, also the subscriptions for the session are invalidated. This must, of course, not affect the RPC style functions that may manipulate a global server state – based on the individual service definition.

2.2.5 The type system

The EXLAP protocol features its own type system, which is intentionally slightly different from that of common programming languages. The goal of the type system in which the interfaces, definitions and signatures of functions and data structures are defined, is to describe what those things represent on an abstract level and not how they should be implemented.

For example: If one must express the liquid level of a fuel tank it is not about a byte value (0-255) but instead about the *Relative* (`<Rel/>`) value 0.0 (empty) or 1.0 (full) and anything in between. This enforces thinking in representations, rather than in implementations and raw data types and is also much friendlier to users and architects outside of the IT domain.

Another example is that of an identifier: Often transactional identifiers are represented as integers, but EXLAP does not know any *pure* integer style types. Rather than thinking of an "`int`", think of an identifier only as "something uniquely identifiable", which should not have any additional meaning. Taking that idea further, one should use the *Text* data type (`<Txt>`) to represent an identifier. It is up to the designer of the interface if he only permits numbers and/or characters and a specific length for valid content using a regular expression that can limit what a Text field can contain. Also declaring an "`integer`" field does not contain any information about its range (e.g. 0-, 1-, 3-13, -65 to 68, unlimited) and often is limited to 16 or 32 bit values. The *Text* data type with regular expression takes care that an interface designer must specify these things – otherwise the tester and service consumer does not know what to expect, program or test against. Alternatively it is suggested to use an *Absolute* data type with a resolution of "1" to represent integer style numbers.

Below an overview of the currently available EXLAP data types:

Data type encoding	Description	Example
`<Act>`	An *Activity* (a Boolean value).	`<Act name="ClutchSwitchActuation" val="true"/>`
`<Abs>`	An *Absolute* represents and absolute value that is expressed as floating point number (optionally also as an integer).	`<Abs name="VehicleSpeed" val="128.3"/>`

`<Alt>`	An *Alternative* represents one out of a predefined choice of objects. It is like a enumeration of objects or a C style union.	`<Alt name="Dog">`
		`  <Txt name="BarkVolume" val="Louder"/>`
		`  <Txt name="Race" val="Golden Retriever"/>`
		`</Alt>`

`<Bin>`	A *Binary* represents an array of bytes.	`<Bin name="Icon"`

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

		`val="VGhpcyBpcyBhIHRlc3Qh"/>`
`<Enm>`	An *Enumeration* is represented by one constant of the predefined collection.	`<Enm name="GearSelection" val="Neutral"/>`
`<List>`	A *List*, which represents a collection out of 0..n elements that are based upon the same object definition.	`<List name="Tracks"> <Elem> <Txt name="Artist" val="U2"/> <Txt name="Title" val="Mofo"/> </Elem> <Elem> <Txt name="Artist" val="U2"/> <Txt name="Title" val="One"/> </Elem> </List>`
`<Rel>`	A *Relative* is something between 0.0 and 1.0. The minimum and maximum *can* be denoted by a descriptive label.	`<Rel name="TankLevel" val="0.35"/>`
`<Obj>`	An *Object*, which is a composite data structure consisting out of all types.	`<Obj name="Title> <Txt name="Artist" val="U2"/> <Txt name="Title" val="Mofo"/> </Obj>`
`<Tim>`	A *Time* consisting out of a date, a time and an optional time-zone.	`<Tim name="ServiceTime" val="2006-10-24T12:00:10.001231+01:00"/>`
`<Txt>`	A *Text* is a sequence of characters that can be limited by specifying a regular expression.	`<Txt name="StationName" val="Radio Gaga"/>`

Table 2: EXLAP data types

2.3 Communication principles

The following sections aim to give only a very compact introduction. The principles will be specified in detail in chapter 3.

2.3.1 General

After successful connection to an EXLAP server, the server acknowledges its readiness with an `<Init/>` status message. After receiving that, the client may send a request, using a `<Req/>` envelope, to the service (see Figure 2).

Any request contains a so called *command* envelope, starting with the command name and then, depending on the command, with additional options and sometime further data. Examples for commands are `<Subscribe/>`, `<Call>`, etc. (see Figure 2 and Figure 3). Commonly a client will first select the protocol version to speak and then retrieve the capabilities of a server, including the offered service, using a `<Protocol/>` command, as detailed in section 3.5.2.

2.3.2 Publish/subscribe

One of the core communication principles offered by the EXLAP protocol is the publish/subscribe pattern. Figure 2 shows a simplified, but syntactically correct real world example of an EXLAP communication by using the publish/subscribe pattern.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

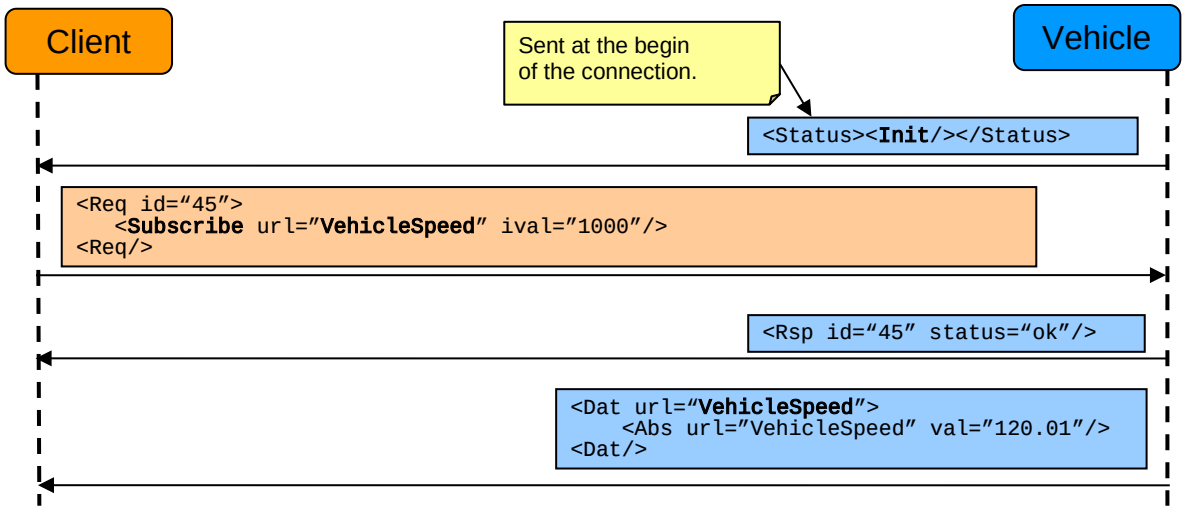


Figure 2: EXLAP <Subscribe/> communication flow

In this example the EXLAP client connects to an EXLAP server that implements a potential *Vehicle* service. The client subscribes to the data object "*VehicleSpeed*" at the server, that representsfor example the current speed of the vehicle monitored. As result from this operation the service will send (publish) the client data updates, whenever the monitored vehicle speed changes, but not more often that the optional interval specified (i.e. 1000 milliseconds).

This example also introduces an important concept of subscriptions realized within EXLAP: The throttling of dynamic data updates at the event source under control of the client.

2.3.3 Asynchronous remote procedure call (RPC)

In addition to subscriptions to data objects from a service (publish-subscribe pattern) EXLAP supports a RPC style call of functions, whereby the return of a result will be delivered in a non-blocking asynchronous way using transaction identifier.

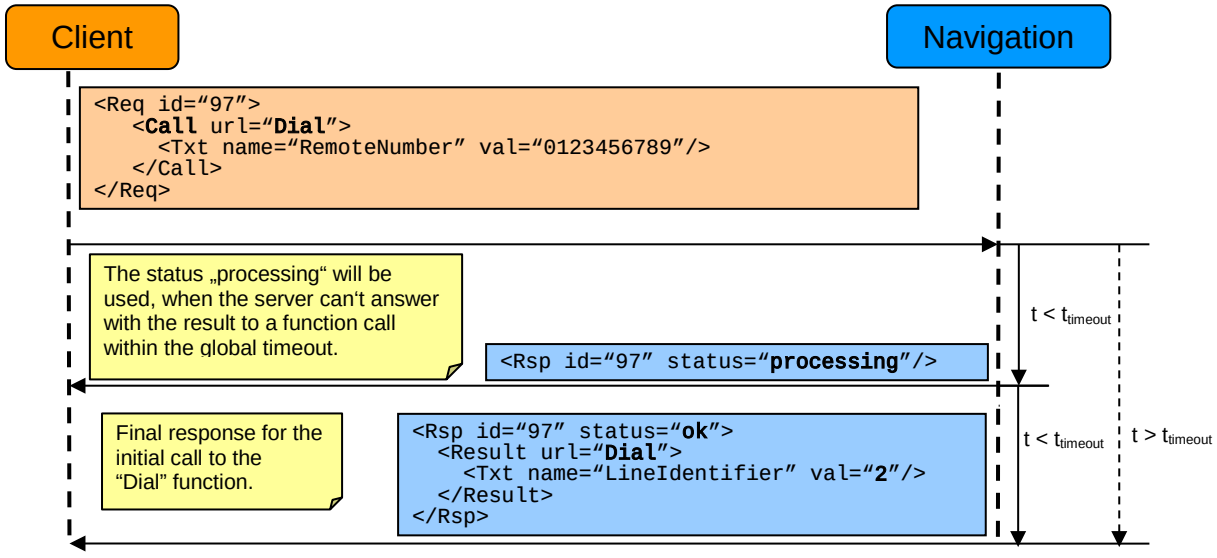


Figure 3: EXLAP <Call/> communication flow

Figure 3 shows the call of a function from the client to a potential *Telephone* service. Since the server, implementing the service, can't complete the function within the command timeout (default: 10 seconds) it must tell this to the client via an intermediate "*processing*" response, before returning the

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

final result. The "*processing*" intermediate response is another EXLAP feature that acts a alive signalling during the execution of long-running ARPC calls.

2.3.4 Additional commands

As application layer protocol EXLAP also supports for:

- Setting of the major EXLAP protocol version and retrieval of the server and service capabilities (*<Protocol/>* command).
- Retrieval of a list of available service specific data object and function names (*<Dir/>* command).
- Optional retrieval of IDL meta-data information regarding the data objects, type definitions and function signatures (*<Interface/>* command).

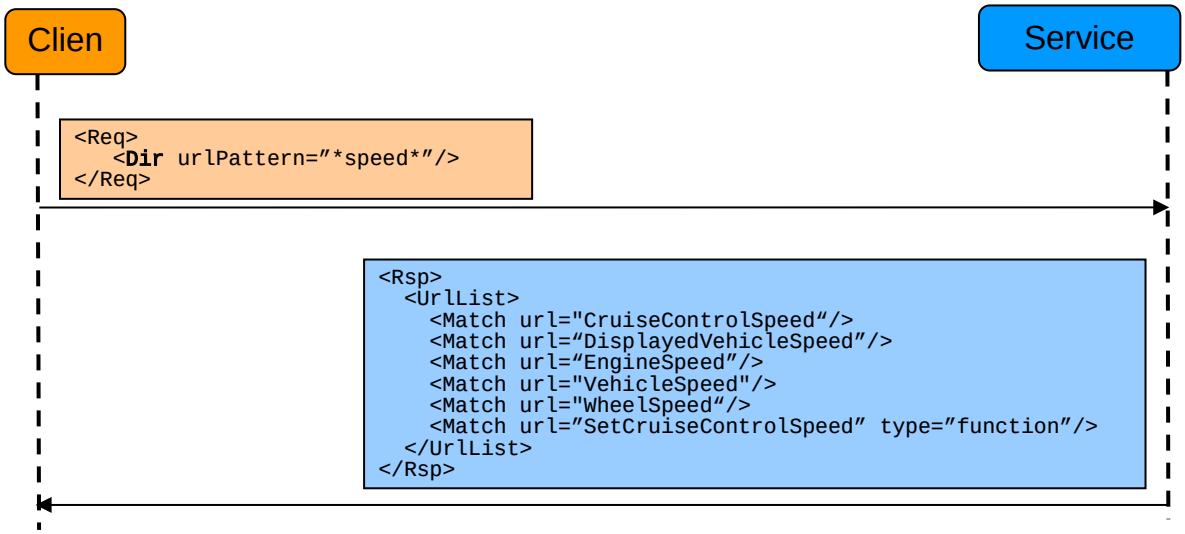


Figure 4: EXLAP <Dir/> communication flow

In Figure 4 a simple example is presented, that shows the interaction on the wire regarding a directory request on a potential *Vehicle* service. Using the directory it is possible to dynamically detect the capabilities of a service implementation.

2.4 Service profile specification and interface definition language (IDL)

Services in EXLAP are commonly defined by a set of functions and/or a set of data objects that are implemented by a server as described in section 2.2.1.

To specify services, the EXLAP protocol suite incorporates its own XML based interface definition language (IDL) which is comparable to classical IDL notations. Using this IDL, one can define a *service interface profile* that includes the data structures (data objects) and the in- and output parameters (signatures) of ARPC style functions. Another important aspect of EXLAP service interface profile is their extendibility: services have a versioning scheme and may grow without losing compatibility to prior versions within certain limits.

The main characteristics of the EXLAP service interface profiles are:

- Central place to define function signatures, data structures and documentation.
- Declaration of a versioning scheme.
- Enabler for:
  - Service specific code generation.
  - Formal validation of the serice interface profile (i.e. IDL) definition.
  - Automated testing.
  - Generation of documentation.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Example out of a example service specification (service interface profile definition):

```
<Profile name="MediaRenderer" version="1.1" xmlns="http://exlap.de/v1/protocol">
```

Example data object definition:

```
<!-- @description CurrentTrack A reference to the media file that is currently
selected. -->
<Object url="CurrentTrack" characteristic="event">
  <!-- @param CurrentTrack The current track that refers to a Track type -->
  <ObjectEntity name="CurrentTrack" typeRef="Track" />
</Object>

<!-- @description Track Represents a single media file. -->
<Type url="Track" required="false">
  <!-- @param TrackIdentifier The unique identifier of the track -->
  <Text name="TrackIdentifier" />
  <!-- @param Title The title of the track -->
  <Text name="Title" />
  <!-- @param Artist The artist of the track -->
  <Text name="Artist" />
  <!-- @param Length The Length of the track -->
  <Absolute name="Length" unit="s"/>
  <!-- @param Album The name of the album -->
  <Text name="Album" />
  <!-- @param TrackNumber The track number starting with 1 -->
  <Text name="TrackNumber" regExp="([1-9][0-9]*)?"/>
</Type>
```

Example function signature definition:

```
<!-- @description Seek Sets the 'Position' of the current track. -->
<Function url="Seek" required="false">
  <In>
    <!-- @param Position Determines the seek position of the track where playing
                         should be continued. -->
    <Absolute name="Position" unit="s"/>
  </In>
  <Out>
    <!-- @param Result Result code of the function -->
    <Enumeration name="Result">
      <!-- @enum ok Function executed with success -->
      <Member id="ok" />
      <!-- @enum mediaError Error while accessing the media (i.e. file, stream) -->
      <Member id="mediaError" />
      <!-- @enum hardwareError Error while accessing the media device -->
      <Member id="hardwareError" />
    </Enumeration>
  </Out>
</Function>
```

```
</Profile>
```

Please note that the EXLAP IDL uses the long names for the data types (i.e. *<Absolute/>*) instead of the abbreviated names (i.e. *<Abs/>*) that are used for transmission of the actual data on the wire. This is also done to denote that definition of a data type and representational encoding of a value are two different things. More information on this topic can be found in chapter 4.

2.5 Security

EXLAP does not define a transport encryption and authentication mechanism, as from view of the protocol they have to be taken care of at the underlying transport layer.

However EXLAP does define a simple authentication mechanism that – additionally to the transport channel authentication - allows an implementation an authorization mechanism to control the visibility

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

and access to data objects and functions offered by a service. The mechanism is described in detail in section 3.5.12.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3 The protocol in detail

This chapter describes the general operating principles of EXLAP in detail. First general definitions will be made, and then the terminology will be introduced. After that encodings and data representation will be presented and lastly the protocol specific commands are introduced.

3.1 General definitions and rules

Shall, *should*, *may* and *can* are used to describe mandatory, recommended, permitted and optional parts of the specification (see http://www.ietf.org/rfc/rfc2119.txt)

XML and encoding:

- EXLAP *shall* use XML V1.0 encoding/representation for the protocol [1].
- UTF-8 character encoding shall be used [4].
- There *shall* be no general limitation or restriction regarding the length of XML element strings or attribute values within the specification. Length or character limitations exist therefore only implicitly, based on the specific command or the data that is transmitted.

Up- and downward compatibility (see section 3.2 for specific EXLAP terms used):

- All protocol extensions within a major version *shall* be downward compatible.
- There *shall* be no protocol or service downward compatibility for two different major versions guaranteed.
- A client *should* inquire the current version of the server first and then continue the processing based on the supported features of the EXLAP specification (Note: see `<Protocol/>` command to set and inquire about the supported protocol versions).
- Regarding data objects: data objects *can* grow in members which *shall* be ignored by clients implementing a service of lesser minor version. Additional fields provided with a new minor version of a service *should* only be of informational character and *shall* not redefine the nature and usage of the data object or service in question.
- Regarding functions: The calling parameters (`<In/>` section) of functions shall not be changed with a minor service version. Only new functions with a unique signature *may* be added during a minor version upgrade. A function result in contrast *may* contain additional fields and parameters, which are to be ignored by clients of a lesser minor service version. Additional fields provided *should* only be of informational character and *shall* not redefine the nature and usage of the function result in question

Mandatory and optional parts to be implemented:

- A protocol version *shall* be supported completely or not at all. Only the commands and options explicitly stated as optional can be omitted. These optional commands are also reflected within the response of the `<Protocol/>` command.
- If an optional command is not implemented but invoked by the client, the server *shall* always response with a status of "`notImplemented`" in the command response.

3.2 Terminology and definitions

3.2.1 EXLAP terminology

This specification uses a number of terms to refer to the roles played by participants in, and objects of, the EXLAP communication.

Term	Comment
EXLAP	An XML based communication protocol.
Server	The entity that process requests and replies with a response, also known as EXLAP server.
Client	The entity that issues requests and waits for (asynchronous) replies, also known as EXLAP client.
Service	A set of objects and functions for a specific task/usage defined by a service

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

	interface profile.
Service interface profile	The superset of all data objects and function signature definitions that define a service with a specific version, as introduced in section 2.4.
Url	A unique resource locator, referring to a specific data object or a function name (this is an exclusive "or" – a url can belong to a function or a data object, but not both).
Data type	Denotes an EXLAP specific data type (e.g. Activity, Enumeration, …) as represented by the EXLAP XML encoding as introduced in section 2.2.5.
Data object	A a container object, identified by an url carrying a specific information (e.g. sensor data, state data, static/descriptive information) in its (1..n) member elements.
Member element	A member element is part of a data object and holds the actual value, represented by a data type. A data object can have 1..n member elements that are each identified by its own unique name. For example: The members of the data object `DoorPosition` could be "`FrontLeft`", "`FrontRight`", etc. Note: In case that a data object has only one member the member name *should* be identical with the name of the data object (i.e. its url).
List	A list that stores multiple data objects of the same type. The first element in a list *shall* be addressed by the index number "1". There are no empty indexes.
Simple data object	A data object that is composed from standard member elements (Abs, Rel, etc.) but does not embed lists or other (sub-) data objects. This is compatible to EXLAP V1.0 behaviour.
Complex data object	A data object that is composed out of the all available member data types including lists and (sub-) data objects. This is a fundamental extension that was introduced with EXLAP V1.1.
Function	A function, identified by a url, can be invoked by the EXLAP client using the `<Call/>` command in an asynchronous RPC style fashion. A function call and *can* include a data object as argument. A function *can* also return a result in form of a data object.
Envelope	The outermost XML element level (root element) used by EXLAP. There are four different outer envelopes which represent different parts of the protocol: 1. Request - `<Req>`, 2. Response - `<Rsp>`, 3. Asynchronous status - `<Status>` and 4. Asynchronous data delivery - `<Dat>`. as further described in the introduction of section 3.3.
Command	A command is part every `<Req/>` envelope. The command issued tells the server to interpret the following data. The response to the command issued from the client is then answered by a `<Rsp/>` envelope by the server. Valid commands for EXLAP V1.x are: `<Subscribe/>`, `<Unsubscribe/>`, `<Get/>`, `<Call/>`, `<Dir/>`, `<Authenticate/>`, `<Interface/>`, `<Alive/>` and `<Heartbeat/>`.

Table 3: EXLAP protocol terminology

3.2.2 Notation used to describe the protocol message format and syntax

The non-formal notation shown below is used for the description of the syntactical format of data structures, (optional) values and commands (i.e. `<Req {id=[nr]}/>`) of this specification. If the formal and complete specification is required, the XML schema specification as presented in section 6.3 *shall* be used.

In general following conditions apply when using the abbreviated format:

- Optional attributes or elements are placed in curly braces. The attribute value is used for content hints and can be empty if not stated otherwise.
  - e.g. `{attribute="Text hint"}`
- A set of options (enumeration) is placed in square brackets, whereby the underlined is the default one.
  - e.g. `[option1|option2|option3]`
- A multiplicity where the last element can be repeated unlimited times is expressed by "…".

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

- e.g. *<Match…>*

3.3 General protocol operating schema

EXLAP is based on four simple XML envelopes to package and represent the communication between client and server as shown in Table 4.

XML Envelope	Comment
<Req/>	A request from the client to the server.
<Rsp/>	A response from the server for a request from the client. Note: There *shall* always be a *<Rsp/>* for every *<Req/>*.
<Dat/>	A message envelope for asynchronous server to client data delivery (i.e. subscribed data objects). The *<Dat>* envelope is described in detail in section 3.4.2 and 3.6.
<Status/>	Asynchronous status information from the server to the client. The status envelope is described in detail in section 3.7.

Table 4: The four EXLAP XML envelopes

3.3.1 Request and response

EXLAP uses an asynchronous request/response schema, as shown in Figure 1, with no guaranteed order of sequence. If one or more requests (*<Req/>*) are issued the server *shall* only answer within the timeout to each single request with no regard for order. For every request (*<Req/>*) the server *shall* answer with the corresponding response (*<Rsp/>*).

The client *may* send multiple requests (commands) in quick succession while the client is not waiting for a response of the first request. The server that receives the requests *may* also process the requests received from the client *in order* (one request at a time - first come, first serve) or process received requests in an individual fashion (*random-order*, i.e. for multithreaded processing of incoming requests) and reply the result for each processed request as fast as possible to the client.

3.3.2 Sequence numbers

Request and response pairs that relate to each other are identified by a specific and unique *id.* The usage of the *id* is optional but strongly suggested to use if a client wants to issue multiple parallel requests.

General format of a request:

```
<Req {id="xsd:nonNegativeInteger-canonical"}/>
```

General format of a response for a request (note: The *status* and *msg* attribute are explained in section 3.3.3):

```
<Rsp {id="xsd:nonNegativeInteger-canonical]"}
     {status="[ok|...]"}
     {msg="text"}/>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Fields and attributes:

Attribute	Description
id	Optional attribute in a *<Req/>* that specifies a numerical identifier, that *shall* be replied in a *<Rsp/>* if present in a *<Req/>*. The syntax of the content of the *id* follows the canonical representation of *xsd:nonNegativeInteger* which prohibits the optional "+" sign and leading zeroes. • Note: In addition, *id* with an integer value of larger than 999999999 *shall* not be used.
	<Req id="56"/> *<Rsp id="56"/>*

Application notes:

- The response generator/handler of a server implementation *shall* only copy the *id* from the request to the response, but never assign them any meaning. Using a simple (*id++*) numbering scheme on the client for the generation of subsequent and unique *id* is suggested.
- While the XML default attribute value is zero (0) there shall be no *id="0"* part in the response when no *id* attribute was present in the request.

3.3.3 Error handling and signalling

To have a general error processing scheme, every response (*<Rsp/>*) *shall* return the sucession *status* of the command issued and *may* support the optional attribute *msg* which *may* be used to return a comprehensive error message for debugging purposes.

The minimal format of responses *shall* be:

```
<Rsp {id="[xsd:nonNegativeInteger-canonical]"}
     {status="[ok|error|internalError|syntaxError|...]"}
     {msg="text"}/>
```

Fields and attributes:

Attribute	Description
status	Optional attribute that contains the processing status of the request. Four status values *shall* be expected and supported in addition to the command specific ones: *ok*, *error*, *syntaxError* and *internalError*. • *ok*: The command action of the request was processed successfully. Additionally there *can* be a message in the *msg* attribute. • *error*: Signals a unspecified and recoverable error. The attribute *msg* attribute *may* be used from side of the server to provide specific details. • *internalError*: The server has encountered an internal software or system error while processing the command (e.g. out of memory) and a recovery is not likely. The *msg* attribute *may* be used at the side of the server to provide specific details. • *syntaxError*: The server has encountered an error in the XML it received from the client. The received XML does thereby not conform to the EXLAP schema specification. The *msg* attribute *may* be used at the side of the server to provide specific details. Note 1: These status attribute values *shall* apply for all commands. Note 2: The default value for this attribute, if not present, *shall* be "*ok*".

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

	<Rsp id="10" status="ok"/> is equivalent to: <Rsp id="10"/> <Rsp status="error" id="10"/> <Rsp status="internalError" id="10"/> <Rsp status="syntaxError" id="10"/>
msg	Optional attribute that may contain a text (*xsd:string*, encoded in UTF-8) that *should* include a detailed textual description regarding the status code. The message *should* be in English. Note: The content of this attribute is intended for debugging purposes only.
	<Rsp id="10" msg="Everything o.k."/> <Rsp id="11" status="error" msg="Something went wrong"/>

3.3.4 Timeouts

A response to an issued request *shall* be sent within 10 seconds. If, in case of the <*Call/*> command, a final response cannot be sent within 10 seconds, an intermediate response with the status of "*processing*" *shall* be sent, as described in section 3.5.6.

Note: If a response is not received within the timeout limit the server *may* assume a transmission error or a general failure of the server or connection. The only method to detect a server-failure for sure *should* be the <Alive/> mechanism as described in section 3.5.10.

3.3.5 Syntactical validation policies regarding EXLAP

If an EXLAP envelope is received by the server that does not begin with <*Req*> the request *shall* be replied with a "*syntaxError*" status response. If the client receives an envelope that does not begin with <*Rsp*>, <*Dat*> or <*Status*> it *shall* ignore the whole message envelope and the following character stream until a valid envelope begin is detected.

Example from view of the server side:

```
<UnknownEnvelope>
  <Test/>
</UnknownEnvelope>

<Rsp status="syntaxError"/>
```

Application note: The client software implementation *may* log or signal internally that an invalid response has been received from the server, so that a tester/developer has the chance to inspect.

If an unknown inner element in a valid envelope (<*Req/*> for the server, <*Rsp/*>, <*Dat/*> or <*Status/*> for the client) is received, the server *shall* answer with a "*syntaxError*" status in the response. If the client receives such an unknown element in a valid envelope it *shall* ignore the whole envelope.

Example from side of the server:

```
<Req id="10">
  <UnknownElement param="1"/>
</Req>

<Rsp id="10" status="syntaxError"/>

<Req id="11">
  <Subscribe url="VehicleSpeed">
    <UnknownElement/>
  <Subscribe/>
</Req>

<Rsp id="11" status="syntaxError"/>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Application note: The client software implementation *may* log or signal internally that an invalid response has been received from the server, so that a tester/developer has the chance to inspect.

If a request with unknown attributes is received by the server the unknown attributes or elements *shall* be ignored. The same is valid for the client, in case responses, data or status updates (<*Rsp/*>, <*Status/*>, <*Dat/*>) are received with unknown attributes.

Example:

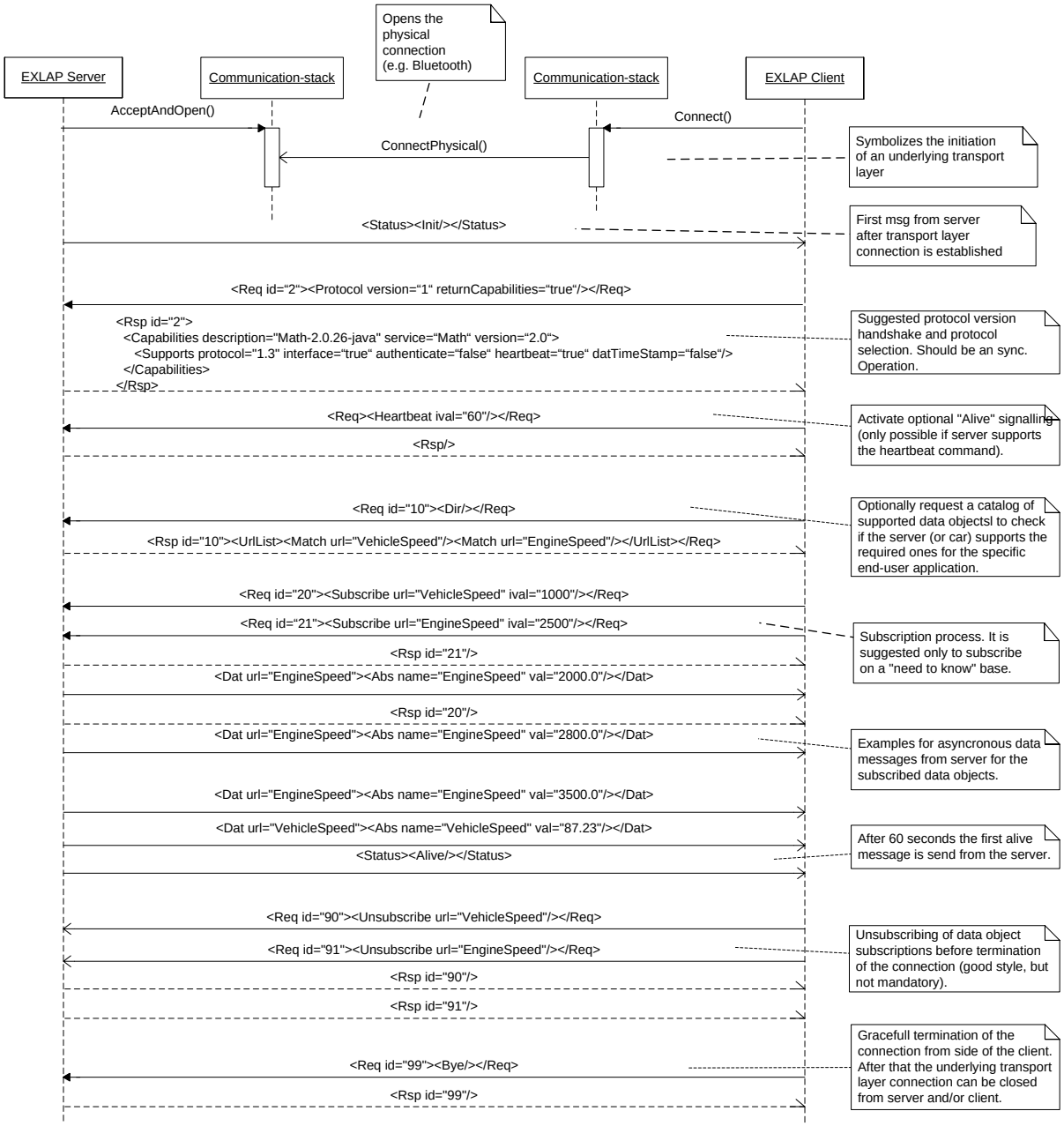
```
<Req id="10" unknownAttribute="abc">
  <Subscribe url="VehicleSpeed" unknownAttribute="test"/>
</Req>

<Rsp id="10" status="ok">
```

Application note: This behaviour is important to secure the downward compatibility of the EXLAP protocol for minor version changes.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3.3.6 Basic EXLAP communication scenario

To provide an example on the general operating concept of the EXLAP protocol, before the specific commands are explained in detail, the diagram shown in Sequence 1 is intended to give an overview of a complete EXLAP protocol communication session.



Sequence 1: Overview of a complete EXLAP communication session

3.4 Data representation and encoding

EXLAP uses its own data type universe as introduced in section 2.2.5. The intention of this data type universe is to abstract type and value representation from specific platform or program language types like `long`, `int16`, `float`, `double`, `array of char`, but also arbitrary bit field representations of enumerations in `int` or `long`**.**

How data types are encoded and represented on the wire is explained in this section.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3.4.1 Representation of EXLAP specific data type values

All the EXLAP data types are based upon the basic XML schema types for value representation. All used XML schema types are defined by the W3C (see [2] and [3]) with the namespace prefix of "`xsd:`" and are used to represent and encode the EXLAP data types on the wire, as shown in Table 5. Please note that the table does not contain the data types *<Alt/>*, *<Obj/>* and *<List/>*, since those are composite types.

EXLAP data type	Description	Example
<Act/>	An *Activity* is a boolean value that can be "*true*" or "*false*". An *Activity* follows the *xsd:boolean* specification.	*val="true"* *val="false"*
<Abs/> and *<Rel/>*	An *Absolute* or *Relative* is always represented as floating point number. It follows the `xsd:double` specification. In general a floating point number has the format `{-}[integerPart].[fractionalPart] {[e|E]{-}exponent}` with the special values: `val="NaN"` (not-a-number) `val="INF"` (positive infinity) `val="-INF"` (negative infinity)	*val="-1e3"* *val="456.79115E11"* *val="12.34e-2"* *val="120"* *val="-0"* *val="0"* *val="-.45E7"* *val="+123"* *val="0005"*
<Bin/>	An *Binary* is an array of bytes represented by a base 64 encoding on the wire (`xsd:base64Binary`).	*val="VGhpcyBpcyBhIHRlc3Qh"*
<Enm/>	An *Enumeration* is a predefined group of elements of which one can be selected and have a textual representation (`xsd:string`).	*val="opened"* *val="closed"* *val="ajar"*
<Tim/>	The *Time* format is a string that follows the ISO8601 extended format including date and timezone: `YYYY-MM-DD[T]HH:MM:SS.fractions[-|+]HH:MM` or `YYYY-MM-DD[T]HH:MM:SS.fractionsZ` if the time zone is GMT/UTC whereby "fractions" are fractional seconds (e.g. ".12", ".6243875" etc.). Valid values are complete date/time values + time zone oder time values (e.g. `HH:MM:SS.fractions[-|+]HH:MM`) with time zone. Instead of the `[+|-]HH:MM` GMT/UTC offset also "Z" can be appended at the end of the string (See ISO 8601 or `xsd:dateTime` definition). Other variation and abbreviation of ISO8601 dates/times are not supported by the EXLAP encoding.	*val="2006-10-24T12:00:10.001231+01:00"* represents 2006-10-24, 13:00 + 10 s + 001 ms + 231 us) *val="10:24:11.31Z"* represents 10:24 + 11 s and 310 ms in GMT) *val="2008-10-23"* (only the date)
<Txt/>	A *Text* is a sequence of characters in between the quotation character ["] (`xsd:string`). A *<Text/>* follows the `xsd:string` specification with UTF-8 encoding and the standard for XML breakout codes.	*val="This is a test"* *val="8>7"*

Table 5: Data type encoding

For a full and detailed reference the original W3C specification *shall* be used. Regarding restrictions and limitations please see section 6.3 for the EXLAP-ML validation schema.

3.4.2 Encoding of EXLAP data structures (<Dat/>)

The *data-encoding* used in EXLAP for a data objects (e.g. *<Dat/>*), function call parameters or results (e.g. *<Call/>*) and *<Get/>* requests is therefore structured as following:

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
data-encoding == 0..n * [0..n * {data-encoding-member}|
                         0..n * {data-encoding-object}|
                         0..n * {data-encoding-alternative}|
                         0..n * {data-encoding-list}]
```

The following sub-sections will provide concrete overview concerning ***data-encoding-member***, ***data-encoding-object*** and ***data-encoding-list*** which together make up the ***data-encoding***.

3.4.3 Encoding of EXLAP simple types (<Abs>, <Act>, <Bin>, <Enm>, <Rel>, <Tim>, <Txt>)

The simple EXLAP member types are encoded using the schemes described in this section.

```
data-encoding-member ==  <[Abs|Act|Bin|Enm|Rel|Tim|Txt]
                            name="member"
                            {val="value"}
                            {state="[nodata|error|ok]}
                            {msg="text"}/>
```

Attributes and elements:

Attributes and Elements	Description
<Abs> *<Act>* *<Bin>* *<Enm>* *<Rel>* *<Tim>* *<Txt>*	Specifies the data type of a member element. The data type is presented in addition to the *name* of the member to allow for dynamic data processing and access - without prior knowledge of the data structure of a data object. Application note: A language or platform specific library may store the value of the member natively based on the data type (e.g. an Absolute member type corresponds to a double in Java or C++).
val	Attribute that hold the value of the member element, identified by the attribute name. The content of the attribute val is represented in the specified data type encoding, as documented in the section 3.4. Application note: When the state of the object is "*error*" or "*nodata*" the *val* attribute *shall* be omitted.
name	Mandatory attribute that specifies the name of the member element.
state	Optional attribute that provides information concerning the state and/or validity of the data provided in the *val* attribute. If the state attribute is not present, the default of "*ok*" *shall* be assumed. • *ok*: The value specified by the "*val*" attribute is valid. • *nodata*: In this case no *val* attribute *shall* be present or the *val* attribute *shall* be ignored. The state "*nodata*" is *not an error state!* and defines that: o The value is currently not available – which might be temporarily until the value becomes available (i.e. when the system or sub-system is ready). o The member is not implemented (e.g. service server) or not provided (e.g. a service client that calls a method). o In any case, the definition or description for the member, the call or the data object *shall* describe that such a condition can happen and/or is allowed. • *error*: Signals that this member is of erroneous state. No *val* attribute *shall* be present or the *val* attribute shall be ignored. Detailed information should be provided in the *msg* attribute.

	`<Abs name="Odometer" state="error"/>`
	`<Abs name="Odometer" val="56868.76"/>`
	`<Abs name="Odometer" state="nodata"/>`

msg	Optional attribute that *can* hold plain text (UTF-8) descriptive or debugging information related to the member (for example: The root cause of

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

	state="error" or "*nodata*" in an actual member element).
	`<Abs name="Odometer" state="error" msg="dev:can0 is not up"/>`

3.4.4 Encoding of EXLAP objects (<Obj/>) in XML

The complex data type *<ObjectEntity>* is represented by *<Obj/>* at the wire-level. An *<Obj/>* is a structure that encapsulates other simple and complex data type.

```
data-encoding-object == <Obj name="[member]"
                            {state="[nodata|error|ok]"}
                            {msg="text"}>
                            {data-encoding}
                        </Obj>
```

Attributes and elements:

Attributes and Elements	Description
<Obj>	Denotes an object element that must contain a *data-encoding* conforming content as defined in section 3.4.2.
name	Mandatory attribute that specifies the name of this member.
msg	Optional attribute that *can* hold plain text (UTF-8) descriptive or debugging information.
state	Optional attribute that provides information concerning the state and/or validity of the data provided in the object. If the state attribute is not present, the default of "*ok*" *shall* be assumed. • *ok*: The member contained in the object are valid. • *nodata*: All the members in the object are of the state *nodata* regardless if they are present or not. Please refer to the state definition in section 3.4.3 for additional details. • *error*: Signals that this object is in an erroneous state and that all the members are of the state *error* regardless if they are present or not Detailed information should be provided in the *msg* attribute.

	`<Obj name="Track">`
	`  <Txt name="Title" val="Happy People"/>`
	`  <Txt name="Artist" val="REM"/>`
	`<Obj/>`
	`<Obj name="Track" state="nodata"/>`
	`<Obj name="Track" state="error"/>`

Simple example:

```
<Obj name="Position">
  <Abs name="Latitude" val="52.254669"/>
  <Abs name="Longitude" val="10.533764"/>
  <Abs name="Height" val="79.5"/>
</Obj>
```

Practical example (response of a <Call/> command):

```
<Rsp>
  <Result url="GetLocationByIdentifier">
    <Obj name="Location">
      <Txt name="City" val="Braunschweig"/>
      <Txt name="Street" val="Georgstrasse"/>
      <Obj name="Position">
        <Abs name="Latitude" val="52.254669"/>
        <Abs name="Longitude" val="10.533764"/>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
        <Abs name="Height" val="79.5"/>
      </Obj>
    </Obj>
  </Result>
</Rsp>
```

3.4.5 Encoding of EXLAP lists (<List/>) in XML

Lists are a powerful way to structure and represent dynamic data, whereby each element of the list refers to the same object type.

```
data-encoding-list == <List name="[member]"
                           {state="[nodata|error|ok]"}
                           {msg="text"}>
                        0..n * {<Elem>
                                   {data-encoding}
                                </Elem>}
                      </List>
```

Attributes and elements:

Attributes and Elements	Description
<List>	Denotes an object element that can contain 0..n elements.
name	Mandatory attribute that specifies the name of this *<List>* member.
<Elem>	Encapsulates a single element of a list, which *shall* contain a data-encoding conforming content as defined in section 3.4.2.
msg	Optional attribute that *can* hold plain text (UTF-8) descriptive or debugging information.
state	Optional attribute that provides information concerning the state and/or validity of the data provided in the list. If the state attribute is not present, the default of "*ok*" *shall* be assumed. • *ok*: The elements contained in the list are valid. • *nodata*: All the elements in the list are of the state *nodata* regardless if they are present or not. Please refer to the state definition in section 3.4.3 for additional details. • *error*: Signals that this list is in an erroneous state and that all the members are of the state *error* regardless if they are present or not Detailed information should be provided in the *msg* attribute.

	`<List name="Tracks">`
	`  <Elem>`
	`  <Txt name="Title" val="Happy People"/>`
	`  <Txt name="Title" val="REM"/>`
	`  </Elem>`
	`</List>`
	`<List name="Tracks" state="nodata"/>`
	`<List name="Tracks" state="error"/>`

Example (Result of a function call):

```
<Result url="Playlist"/>
  <List name="Tracks">
    <Elem>
      <Txt name="Artist" val="Arch Enemy"/>
      <Txt name="Title" val="We will rise"/>
      <Txt name="Album" val="Live in Japan"/>
    </Elem>
    <Elem>
      <Txt name="Artist" val="Arch Enemy"/>
      <Txt name="Title" val="Nemesis"/>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
      <Txt name="Album" val="Live in Japan"/>
    </Elem>
  </List>
</Result/>
```

Empty list:

```
<Result url="Playlist"/>
  <List name="Tracks"/>
</Result>
```

Practical example (response to a <Get/> request)

```
<Rsp id="678">
  <ObjectData url="CurrentPlaylist">
    <Txt name="Description" val="H.C.’s playlist"/>
    <List name="Tracks">
      <Elem>
        <Txt name="Artist" val="Arch Enemy"/>
        <Txt name="Title" val="We will rise"/>
        <Txt name="Album" val="Live in Japan"/>
      </Elem>
      <Elem>
        <Txt name="Artist" val="Supertramp"/>
        <Txt name="Title" val="Logical song"/>
        <Txt name="Album" val="Breakfast in America"/>
      </Elem>
    </List>
  </ObjectData>
</Rsp>
```

3.4.6 Encoding of EXLAP alternative types (<Alt/>) in XML

The complex data type *<Alternative>* is represented by *<Alt/>* at the wire-level. An *<Alt/>* is a structure that holds the content of one object type out of the choice of many (i.e. struct or union like). The definition is as follows:

```
data-encoding-alternative == <Alt name="[member]" type="[typeRef-choice]"
                                {state="[nodata|error|ok]"}
                                {msg="text"}>
                                {data-encoding}
                             </Alt>
```

Attributes and elements:

Attributes and Elements	Description
<Alt>	Denotes an alternative element that must contain a *data-encoding* conforming content as defined in section 3.4.2.
name	Mandatory attribute that specifies the name of this member.
msg	Optional attribute that *can* hold plain text (UTF-8) descriptive or debugging information.
type	Mandatory attribute that specifies the type (i.e. structure) of the transported data-content which is based on a type definition selection specified in the corresponding *<Alternative>* definition (i.e. *<Choice typeRef=…/>*).

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

state	Optional attribute that provides information concerning the state and/or validity of the data provided in the object. If the state attribute is not present, the default of "*ok*" *shall* be assumed. • *ok*: The member contained in the alternative are valid. • *nodata*: All the members in the alternative are of the state *nodata* regardless if they are present or not. Please refer to the state definition in section 3.4.3 for additional details. • *error*: Signals that this alternative is in an erroneous state and that all the members are of the state *error* regardless if they are present or not Detailed information should be provided in the *msg* attribute.

Simple example:

```
<Alt name="Animal" type="Dog">
  <Txt name="BarkVolume" val="Louder"/>
  <Txt name="Race" val="Golden Retriever"/>
</Alt>
```

3.5 Command messages

This section describes the different commands that can be embedded in a <*Req/*> envelope of EXLAP and be sent from the client to the server to request and initiate actions.

3.5.1 Overview of the available EXLAP commands

The following table provides an overview of the command messages defined in the EXLAP protocol:

Command	Description	Mandatory
<*Protocol/*>	Command that requests the server to use a specific major version of the EXLAP protocol.	Yes
<*Dir/*>	Command to request an object catalog of data object urls that match a given pattern.	Yes
<*Subscribe/*>	This command requests asynchronous data updates based on content changes of a data object specified by an url (e.g. "*EngineSpeed*") from the server.	Yes
<*Unsubscribe/*>	Command to remove an active subscription for a given url from the server.	Yes
<*Call/*>	Command to (asynchronously) call a function on side of the server.	Only if there are functions listed in the directory.
<*Get/*>	Command to request a single update of a data object specified by a url.	Yes
<*Bye/*>	Command that allows the client to terminate an EXLAP connection to the server gracefully.	Yes
<*Alive/*>	Command that allows the client to test if the server is still responding without issuing a command that changes the server's state.	Yes
<*Heartbeat/*>	Command to request the server that it *shall* send an <*Alive*> status message in a specified interval to the client.	No
<*Interface/*>	Command that requests detailed type/structure information concerning a single data object or function.	No
<*Authenticate/*>	Optional command that requests a challenge/response digest authentication to gain access to protected data objects.	No

Table 6: Overview of the EXLAP commands

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3.5.2 Protocol version selection - <Protocol/>

With the *<Protocol/>* command the client requests the server to use a specific major version of EXLAP. This command can be issued at all times from a client to a server and should not change the application or service state. The server should – if required and optionally supported by the implementation – seamlessly change the EXLAP version it speaks. In case a future service might use protocol extensions that cannot be represented using older EXLAP major versions – the future service *shall* not be made available to those older EXLAP versions.

Application and implementation notes:

- Note 1: If the server receives no *<Protocol/>* command, it may default to an EXLAP version of its own choice.
- Note 2: It *shall* be guaranteed that the *<Protocol/>* command as defined in EXLAP version 1 will be downward-compatible in all future versions and only extended by optional attributes.
- Note 3: Every future major version can redefine the contents of the *<Capabilities/>* element in the response. If a client asks for a specific major version, it *shall* therefore be sure that it can handle the response.

Format:

```
<Req>
  <Protocol version="major-version:xsd:positiveInteger-canonical"
            {returnCapabilities="[true|false]"}/>
</Req>
```

Request attributes:

Attribute	Description
version	The requested major protocol version (*xsd:positiveInteger-canonical* format).
returnCapabilities	If this optional attribute is "*true*" then the response will contain the *<Capabilities/>* element that provides additional information regarding the EXLAP versions and features that are supported and the service that is implemented by the server.

Example:

```
<Req id="555">
  <Protocol version="1" returnCapabilities="true"/>
</Req>
```

Application note: The response to the *<Protocol/>* command request, *shall* be replied with *status="ok"* when the server supports the requested protocol version and with "*protocolNotSupported*" otherwise.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Format for a response to a <*Protocol/*> command:

```
<Rsp {id="nr"}
     {status="[ok|error|syntaxError|internalError|
               protocolNotSupported]"}
     {msg="[text]"}>
  <Capabilities {description="text"} service="service-name"
       {version="[major-version:xsd:positiveInteger-canonical(1)].
                 [minor-version:xsd:positiveInteger-canonical(0)]"}>
     1..n * <Supports protocol="[major.minor]>
               {interface="[true|false]"}
               {authenticate="[true|false]"}
               {heartbeat="[true|false]"}
               {datTimeStamp="[true|false]"}/>
  <Capabilities>
</Rsp>
```

Response attributes and elements:

Attribute	Description
status	Optional attribute that returns the processing status of the request. • *ok*: The server *shall* respond with this status if the command was executed successfully and the specified protocol is supported. The error status values apply as described in section 3.3.3. • *protocolNotSupported*: The server *shall* respond with this status code if the protocol version that was requested is not supported.
<*Capabilities/*>	Element that contains the capability details of the service implemented by the server.
description	Optional attribute that may contain a free text that describes the server in a short form. It relates closely to the "*description*" attribute of the <*ServiceBeacon/*> definition as described in section 5.2.3
service	Mandatory attribute that relates to the "*service*" attribute of the <*ServiceBeacon/*> definition as described in section 5.2.3.
version	Optional attribute which relates to the service interface profile "*version*" attribute of the <*ServiceBeacon/*> definition as described in section 5.2.3 with a default of "*1.0*".
<*Supports/*>	*1..n* elements that describe the EXLAP protocol major versions that the EXLAP server supports.
Protocol	Mandatory attribute that specifies the supported protocol version in the format *[major].[minor]* (each in *xsd:positiveInteger-canonical* form). The minor part reflects the highest supported minor version within a major protocol version.
interface	Optional attribute that *shall* be set to "true" when the server supports the <*Interface/*> command. If the attribute is not present, "false" *shall* be assumed.
authenticate	Optional attribute that *shall* set to "true" when the server supports the <*Authenticate/*> command. If the attribute is not present, "false" shall be assumed.
heartbeat	Optional attribute that *shall* be set to "true" when the server supports the <*Heartbeat*> command. If the attribute is not present, "false" *shall* be assumed.
datTimeStamp	Optional attribute that *shall* be set to "true" when the server supports the the *timeStamp* attribute of the <*Subscribe*> command and the <*Dat*> message. If the attribute is not present, "false" *shall* be assumed.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Example:

```
<Rsp status="ok" id="556">
  <Capabilities description="Math-2.0.26-java"
                service="Math" version="2.0">
    <Supports protocol="1.3"
              interface="true"
              authenticate="false"
              heartbeat="true"
              datTimeStamp="false"/>
  </Capabilities>
</Rsp>
```

3.5.3 Synchronous request of data - <Get/>

The *<Get>* command requests (i.e. polls) the current value of a data object. The server delivers the result within the corresponding response message in an immediate fashion.

Implementation and application notes:

- Note 1: A *<Get/>* response delivers always the latest available value of the requested data object.
- Note 2: The request is non-blocking. If no data is available, then a data object with members of the *status="nodata"* is returned.
- Note 3: EXLAP versions before version 1.3 required a subscription of the data object to allow a *<Get/>* request. In case no subscription was issued beforehand, the server would have send a *status="noSubscription"* response. To be downward compatible to EXLAP 1.2 (if required at all, as the V1.3 is the first public EXLAP specification) it is suggested a) that unknown error states are interpreted as an *"error"* and b) to at least subscribe data object urls with an update interval of 60000 before performing a *<Get/>* request.

Format for a *<Get/>* request:

```
<Req>
  <Get url="url"/>
</Req>
```

Request attributes:

Attribute	Description
url	The *url* of the data object to be retrieved.

Example:

```
<Req id="62790">
  <Get url="VehicleIdentificationNumber"/>
</Req>

<Req id="5855">
  <Get url="PositionInfo"/>
</Req>
```

The response for a *<Get/>* command acknowledges the *<Get>* command and delivers the requested data. The *url* attribute states to which data object the data belongs, so that a response to a *<Get/>* command can be processed without regarding for the *id*.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Format for a response of a <*Get/*> request:

```
<Rsp {id="nr"}
     {status="[ok|error|internalError|syntaxError|
               noMatchingUrl|accessViolation]"}
     {msg="text"}>
  <ObjectData url="url">
    {data-encoding}
  </ObjectData>
</Rsp>
```

Response attributes and elements:

Attribute	Description
url	The *url* of the data object that is returned.

status	Optional attribute that returns the processing status of the request. The ok and error status values apply as described in section 3.3.3.
	• *noMatchingUrl*: The server *shall* return this status code to signal the client, that no data object matches the name stated in the *url* attribute. Note: Data objects that are not accessible, due to a missing authorization (see <*Authenticate/*> command) are treated as not present.
	• *accessViolation*: The server *shall* respond with this status when the client tries to access an url that belongs to a function. Note: The <*Get/*> command must only be used on data objects.
	
	`<Rsp status="noMatchingUrl" id="62785"/>`
	`<Rsp status="accessViolation" id="62785"/>`

<*ObjectData/*>	An element container holding the value(s) of the requested data object, list or element of a list in the form of a *data-encoding*. Note: The container *can* be empty if the server has no data to respond, because a <*Get/*> command *shall* always return immediately.
	
	`<ObjectData url="VehicleSpeed">`
	`  <Abs name="VehicleSpeed" val="87.7"/>`
	`</ObjectData/>`

Example for a complete response:

```
<Rsp id="62790">
  <ObjectData url="VehicleIdentificationNumber">
    <Txt name="VehicleIdentificationNumber" val="WVWZZZ1JZ2W123456"
state="ok"/>
  </ObjectData>
</Rsp>
```

Example for a response when no data was available:

```
<Rsp id="62790">
  <ObjectData url="VehicleIdentificationNumber">
    <Txt name="VehicleIdentificationNumber" state="nodata"/>
  </ObjectData>
</Rsp>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3.5.4 Data object subscription - <Subscribe/>

This command requests asynchronous data updates based on content changes of a data object or data member elements from the server. Requests are specified by a simple *url* (e.g. *EngineSpeed*). The data updates will be sent using <*Dat/*> envelopes.

The <*Dat/*> envelopes *shall* be sent, when the content of the data object specified by *url* is updated at the event source or by the server application implementation. To initialize the client applications state, a <*Dat/*> message with the current object state *shall* be sent immediately after the subscribe command was accepted, even if the state of the newly subscribed data object is "*nodata*" or "*error*".

- Note 1: A subsequent <*Subscribe/*> request (i.e. with the same *url*) *shall* overwrite an existing request. If the optional attributes are not specified, the default values *shall* be assumed by the server.
- Note 2: EXLAP versions before 1.3 had the optional attribute *notification* with the values *onUpdate*, *never* (default) and *onceAvailable*. EXLAP 1.3 services will always deliver updates of subscribed data objects, which follow the *onUpdate* behaviour. New service server implementations may support the old *notification* attribute and set the interval on their behalf to 60000 if they do receive a *notification="never"* or *notification="onceAvailable"* in subscribe requests to mimic the old behavior to a certain degree.

Format for a <*Subscribe/*> request:

```
<Req>
  <Subscribe url="url"
     {ival="[milliseconds:xsd:nonNegativeInteger(0)]"}
     {content ="[true|false]"}
     {timeStamp ="[true|false]"}/>
</Req>
```

Request attributes:

Attribute	Description
url	The *url* of the data object to be subscribed.
ival	Optional attribute to specify the minimal interval in milliseconds (ms) between an update of a data object sent by the server (i.e. sending of two subsequent <*Dat/*> messages for the same *url*) in a xsd:nonNegativeInteger format. • Note 1: An *ival* with the value of *"0"* disables any throttling of updates, which *shall* also be the default value, if this optional parameter is not supplied. • Note 2: An *ival* *shall* only apply for data objects with a "*dynamic*" *characteristic* (see section 3.5.11 and 4.3). If applied to data objects with a *characteristic* of "*event*" or "*static*" the ival attribute shall be ignored. • Application note: The motivation behind this option is to slow down the rate in which updates of data objects are sent to the client, since not all clients might be able or might be interested to process every change of the content. <Req id="56567"> <Subscribe url="EngineSpeed" ival="200"/> </Req>
content	Optional attribute to specify if data *shall* be sent in the body of an <Dat/> message in case of a data object update, with a default value of "true". In case no data *shall* be sent in the body of an <Dat/> message, the attribute value must be "*false*". The empty <Dat/> message will signal the client that value of the data object specified by "url" has changed and can be retrieved with a <Get/> command from the server. • Note: This mechanism is intended for complex objects with many

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

	members, so that the application logic at the client can decide when the data is really required and *should* be transferred (i.e. via *<Get/>* command).

	`<Req id="56567">`
	`  <Subscribe url="EngineSpeed" content="false"/>`
	`</Req>`
	
	Results in the following <Dat/> for every change in EngineSpeed:
	
	`<Dat url="EngineSpeed"/>`

timeStamp	Optional and not mandatory to implement attribute to denote the date/time the *<Dat/>* message was produced at the server, with default value of "*false*". To enable the time stamping the attribute value must be "*true*". • Note 1: If the generation of time stamps is not supported by the server, this field *shall* be ignored by the server if present in a client request. To check if time stamping is supported, the client *should* use the information replied by the *<Protocol/>* command, as described in section 3.3.4. • Note 2: The intention behind this option is that the client can timely relate the received data object updates in case of congestions on the transport layer (i.e. for logging applications, etc.).

	`<Req id="56567">`
	`  <Subscribe url="EngineSpeed" timeStamp="true"/>`
	`</Req>`
	
	Results in the following <Dat/>:
	
	`<Dat url="EngineSpeed"`
	`  timestamp="2012-09-13T10:46:23.155+00:00">`
	`  <Abs "EngineSpeed" val="3223.2"/>`
	`</Dat>`

Example for an object subscription:

```
<Req id="56567">
 <Subscribe url="EngineSpeed"/>
</Req>
```

Reply from the server to acknowledge the *<Subscribe/>* command:

```
<Rsp {id="nr"}
     {status="[ok|error|internalError|syntaxError|
               noMatchingUrl|
               subscriptionLimitReached|
               accessViolation]"}
     {msg="text"}/>
```

Response attributes and elements:

Attribute	Description
status	Optional attribute that returns the processing status of the request. The ok and error status values apply as described in section 3.3.3. • *noMatchingUrl*: The server shall return this status code if no data object in the active configuration of the server matches the given *url*. Data objects that are not accessible, due to a missing authorization (see *<Authenticate/>* command) are treated as not present. • *subscriptionLimitReached*: The service server *shall* return this status code to signal that it cannot process any more simultaneous subscriptions

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

	(e.g. CPU/memory limitation in the server or limit in the servers implementation). • *accessViolation*: The server *shall* return this status code to signal the client, that the given *url* is valid, but the *url* can't be subscribed (i.e. read). This is the case when the url belongs to a function.
	<Rsp status="ok" id="56567"/> <Rsp status="noMatchingUrl" id="56568"/>

3.5.5 Cancel an unsubscription - <Unsubscribe/>

Command to remove an active subscription for a given *url* from the server.

Note: Subsequent unsubscriptions to the same (already unsubscribed) *url* *shall* result all in the status "*ok*".

Format for a <*Unsubscribe/*> request:

```
<Req>
  <Unsubscribe url="url"/>
</Req>
```

Request attributes:

Attribute	Description
url	Attribute that specifies the data object to be unsubscribed.

Example for a data object unsubscription:

```
<Req id="623">
  <Unsubscribe url="VehicleSpeed"/>
</Req>
```

Format for a response of an <*Unsubscribe/*> request:

```
<Rsp {id="nr"}
     {status="[ok|error|internalError|syntaxError|
               noMatchingUrl|accessViolation]"}
     {msg="text"}/>
```

Response attributes and elements:

Attribute	Description
status	Optional attribute that returns the processing status of the request. The ok and error status values apply as described in section 3.3.3. • *noMatchingUrl*: The server *shall* respond with the status "*noMatchingUrl*" when the url pattern provided by the *urlPattern* attribute matches no data object url in the active configuration of the server • *accessViolation*: The server *shall* return this status code to signal the client, that the given *url* is valid, but the *url* can't be unsubscribed/subscribed (i.e. read). This is the case when the url belongs to a function.
	<Rsp id="623"/> <Rsp id="623" status="noMatchingUrl"/>

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3.5.6 Calling of functions - <Call/>

The <*Call/*> command invokes an asynchronous function call specified by a *url* at the server. The result of the call will be delivered in a subsequent <*Rsp/*>. If the server can't answer to the function within the global timout of 10 seconds it *must* extend the global response timeout by sending a <*Rsp id="nr" status="processing"/*> response before expiration of the timeout.

- Note 1: It is strongly encouraged to use unique *id* with the requests.
- Note 2: If the <*Call/*> command is not implemented, because the service to implement offers no function/method calls, <*Call/*> always *shall* reply with the status of "*noMatchingUrl*".

Format for a <*Call/*> request:

```
<Req>
  <Call url="url">
    {data-encoding}
  </Call>
</Req>
```

Request attributes:

Attribute	Description
url	Mandatory attribute to specify the url of the function to be called.
data-encoding	Arguments passed to the called function. All in the interface defined arguments have to be present. If an argument is optional (i.e. not *required*) in the service profile interface it *may* be omitted or be present without the *val* attribute but a *state="nodata"*.

	`<Req id="97">`
	`  <Call url="Dial">`
	`  <Txt name="RemoteNumber" val="0123456789"/>`
	`  </Call>`
	`</Req>`

Example for a <*Call/*> command:

```
<Req id="623">
  <Call url="EjectMedia"/>
</Req>
```

A response to a <*Call/*> command request can come in three variations:

a) As s <*Rsp/*> with status "*ok*" and a <*Result/*> element (even if there are no return values),
b) a <*Rsp/*> with the status of "*processing*" to signal the client to extend the response timeout,
c) or a <*Rsp/*> signalling any other (i.e. error) status.

Format for a response of an <*Call/*> request:

```
<Rsp {id="nr"} {status="[ok|error|internalError|syntaxError|
                         processing|accessViolation
                         noMatchingUrl|
                         invalidParameter]"}
     {msg="text"}>
  {<Result url="url">
    {data-encoding}
  </Result>}
</Rsp>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Response attributes and elements:

Attribute or element	Description
<Result/>	Element that contains the results (i.e. return values) of a successfully executed function call. It *shall* be present when the status is "*ok*", but *shall* not be present when the status is not "*ok*".
url	Url of the function initiating this response data.
data-encoding	Response content, which carries the encoded data (see also section 3.4.2).
status	Optional processing status of the request. The ok and error status values apply as described in section 3.3.3. • *ok*: Additional note: A status of "*ok*" only implies that the function exists and all the required parameters where provided correctly. If no result values are returned, then the function call is equal to a "*void function()*" in C/Java. If a simple return value like "*boolean function()*" is defined, it should be passed within the *<Result/>* element of the response (see example). • *processing*: The server *shall* return this status code to signal the client, that he resetted the timeout for the final answer to this request. This response *shall* be used when the server can't answer with a result within the global timeout. Since this response must be received by the client within the global timeout the server *should* send this status after `t=(GlobalTimeout/2)` has passed. • *invalidParameter*: The server *shall* return this status code to signal the client, that one of the submitted parameter contains an invalid or erroneous (i.e. out of range) value or a parameter is missing. • *noMatchingUrl*: The server *shall* return this status code to signal the client, that no function is matching the *url* attribute. • *accessViolation*: The server *shall* respond with this status when the client tries to execute (Call) an *url* that belongs to a data object.

```
<Rsp id="623"/>

<Rsp id="623" status="noMatchingUrl"/>

<Rsp id="623" status="processing"/>

<Rsp status="ok" id="5698">
  <Result url="EjectMedia">
    <Enm name="Result" val="ok"/>
  </Result>
</Rsp>

<Rsp id="97" status="ok">
  <Result url="Dial">
    <Txt name="LineIdentifier" val="238b3a2"/>
    <Enm name="Result" val="ok"/>
  </Result>
</Rsp>
```

3.5.7 Directory functionality - <Dir/>

The *<Dir/>* command shall request a catalog of available *urls* of data objects and functions that match the given pattern.

Application and implementation notes:

- Note 1: If *<Dir/>* is stated without the *urlPattern* attribute, the *urlPattern="*" shall* be the default value, that matches all the available urls.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

- Note 2: The data objects and functions visible via *<Dir/>* *should* not change during runtime. An exception to this rule is when the *<Authenticate/>* command is used to get access to formerly hidden/protected data object or function urls.

Format for a *<Dir/>* request:

```
<Req>
  <Dir {urlPattern="{*}url{*}"}
       {fromEntry="xsd:positiveInteger(1)"}
       {numOfEntries="xsd:nonNegativeInteger(999999999)"}
</Req>
```

Request attributes:

Attribute	Description
urlPattern	Optional attribute, to specify a name pattern, which is applied to the combined url catalog (data objects and functions) of the server. Only urls that match the wildcard *shall* be returned in the response. The following wildcard options are supported: • `"*"` - matches all. • `"text"` – matches the exact url "text". • `"text*"` – matches urls that start with "text". • `"*text*"` – matches urls that contain "text". • `"*text"` – matches urls that end with "text". Note: The url pattern *shall* be case-insensitively processed.
fromEntry	Optional attribute that specifies the first entry of the result (url in the catalog that match the *urlPattern* attribute) that *shall* be included in the response. • Note 1: Since object catalog can contain a large number of data object urls, the response to a directory-request could be quite large for memory constrained embedded client. To allow the client to split the response, the *<Dir/>* command supports fragmented responses which return only parts of the resulting directory matches. To do that, the optional attribute *fromEntry* states the starting number (first element is "1") from which the resulting set of matching data object url *shall* be returned. If *fromEntry* is not specified, the *fromEntry* value *shall* be assumed as "1". • Note 2: If "*fromEntry*" is greater than the highest number of found matching data object url a `status="noMatchingUrl"` *shall* be returned.

	`<Req id="5456">`
	`  <Dir urlPattern="*Speed" fromEntry="10"/>`
	`</Req>`

numOfEntries	Optional attribute that specifies the maximum number (`xsd:positiveInteger-canonical`) of results (items that match the pattern attribute) that *shall* be returned in the response. The maximum value of *numOfEntries* *shall* be not limited. If *numOfEntries* is not specified, all entries starting from *fromEntry* *shall* be returned.

	`<Req id="5454">`
	`  <Dir urlPattern="*Speed" fromEntry="1" numOfEntries="10"/>`
	`</Req>`

Example for a *<Dir/>* request:

```
<Req id="5454">
  <Dir urlPattern="*"/>
</Req>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

The response to a *<Dir/>* request includes a list of matching urls.

Response for a *<Dir/>* command:

```
<Rsp {id="nr"}
     {status="[ok|error|syntaxError|internalError|noMatchingUrl]"}
     {msg="text"}>
  <UrlList>
    0..n <Match url="url"
                {type="[object|function]"}
                {isSubscribed="[true|false]"}/>
         <Match .../>
    ...
  </UrlList>
</Rsp>
```

Response attributes and elements:

Attribute or element	Description

<UrlList>	An element envelope, that contains the matching data object or function urls. The elements returned are accessible with the current permissions *(see <Authenticate/>* command).
	• Note 1: If credentials or a missing authorization (see *<Authenticate/>* command) do not permit any read request of a specific data object url, the url *shall* not be returned in the response.
	• Note 2: If less matching elements are returned than requested (i.e. fragmented requests), it *should* be assumed, that there are no more matches.
	• Note 3: If there are matching urls, but out of the scope of *fromEntry*, an empty *UrlList shall* be returned with the status of "ok". If no url matches the *urlPattern*, the status of "*noMatchingUrl*" *shall* be returned.
	• Note 4: There *shall* be no guaranteed sorting order of the urls (i.e. alphabetical) assumed, although an alphabetical sorting order of the directory could be beneficial for human reading.
	• Note 5: If no urls are returned or match the filter criteria an empty *<UrlList/>* element shall be returned.
	`<Rsp status="ok" id="5456">`
	`  <UrlList/>`
	`</Rsp>`

<Match>	Contains the data from a single, matching url. • Note: The attributes *subscription* and *isSubscribed shall* not applied when the url belongs to a function (see also *type* attribute).

url	Attribute that specifies a matching data object url of the object catalog (i.e. to the *urlPattern* attribute from the request.
	`<Rsp status="ok" id="5454">`
	`  <UrlList>`
	`  <Match url="VehicleSpeed"/>`
	`  </UrlList>`
	`</Rsp>`

isSubscribed	Optional attribute that indicates if a data object is currently subscribed by the client. • Note: If the attribute *isSubscribe* is not present in the response, the default of "*false*" *shall* be assumed, which *shall* always apply when the url belongs to a function. `<Rsp status="ok" id="5477">`

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

	<UrlList> <Match url="EngineSpeed" isSubscribed="true"/> <Match url="VehicleSpeed"/> </UrlList> </Rsp>
type	Optional attribute that specifies the nature of the *url*: • ***object***: data object (accessible via <*Get/*> and <*Subscribe/*>), also the default value. • ***function***: executable function (accessible via <*Call/*>).
status	Optional attribute that returns the processing status of the request. The ok and error status values apply as described in section 3.3.3. • *error*: Additional note: The *server shall* respond with this status code when the attributes *fromEntry* or *numOfEntrys* are out of range or invalid. • *noMatchingUrl*: No *url* is matching the *urlPattern* search criteria.
	<Rsp id="6233" status="error" msg="Invalid fromEntry"/>

Example:

```
<Rsp id="67">
  <UrlList>
    <Match url="Dial" type="function"/>
    <Match url="LineStatus" isSubscribed="true"/>
  </UrlList>
</Rsp>
```

3.5.8 Termination of connection - <Bye/>

Command to allow the client to terminate an EXLAP connection to the server gracefully. Note: A client *should* always try to send a <*Bye*> before terminating the connection.

Format of the <*Bye/*> command:

```
<Req>
   <Bye/>
</Req>
```

Example:

```
<Req id="666">
  <Bye/>
</Req>
```

The server response to an <*Bye/*> request *should* be acknowledged with the status "*ok*".

- Application note 1: After the response is sent by the server it *should* terminate the connection. The client *shall* wait for the response or timeout before closing the connection to the server.
- Application note 2: The server *shall not* sent a <Status><Bye/></Status> message to the client, if the client initiated the termination of the connection itself.

Format of a response to a <*Bye*> request:

```
<Rsp {id="nr"}
        {status="[ok|error|syntaxError|internalError]"}
        {msg="text"}/>
```

Example:

```
<Rsp id="666"/>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3.5.9 Heartbeat - <Heartbeat/>

Optional command to tell the server that it *shall* send an <Alive/> status message in a specified interval (in seconds) to the client. Application note: The heartbeat can be used to detect problems within the communication in automated environments.

Format of a request for a <Heartbeat/> command:

```
<Req>
   <Heartbeat ival="[xsd:nonNegativeInteger-canonical(0-60)]"/>
</Req>
```

Request attributes:

Attribute	Description
ival	Attribute to specify the interval in seconds (`xsd:positiveInteger-canonical`, values from 1 to 60) after which the server *shall* sent an <Alive/> status message to the client. If the value of the attribute is 0, the function *shall* be switched off. • Note 1: The server *should* sent an *<Alive/>* response for a *<Heartbeat/>* command within the timely precision of +/- 1 second. • Note 2: The time base for the next *<Alive/>* status message to be received by the client *should* be the last received *<Alive/>* status message. • Note 3: After 3 missing *<Alive/>* status messages the client *should* assume the connection as erroneous.

Example:

```
<Req id="8888">
  <Heartbeat ival="10"/>
</Req>
```

Format of a <Heartbeat> response:

```
<Rsp {id="nr"}
        {status="[ok|error|syntaxError|internalError|
                        notImplemented]"}
        {msg="Text"}/>
```

Response attributes and elements:

Attribute	Description
status	Optional attribute that returns the processing status of the request. The ok and error status values apply as described in section 3.3.3. • *error*: Additional note: The server *shall* respond with this status code when the interval specified by the *ival* attribute was out of range or invalid. • *notImplemented*: The server *shall* respond with this status code when the optional *<Heartbeat/>* command is not supported by the server (see also *<Protocol/>* command in section 3.5.2).

	`<Rsp id="8888"/>`
	`<Rsp status="error" id="8822" msg="Invalid ival"/>`
	`<Rsp status="notImplemented" id="8889"/>`

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3.5.10 Connection test - <Alive/>

The *<Alive/>* command allows the client to test if the server is still responding.
Format of a *<Alive/>* request:

```
<Req>
   <Alive/>
</Req>
```

Example:

```
<Req id="8888">
  <Alive/>
</Req>
```

The response from the server to an *<Alive/>* request *should* be a simple `status="ok"`. If a *<Rsp>* answer for an *<Alive/>* request is not received within the general timeout for requests (see section 3.3.4) the client *should* assume a problem on side of the server or within the connection (e.g. connection loss, queue overflow, processing problems).

- Note: It is up to the client to gracefully handle a response status other than "*ok*" or a timeout. It is suggested that a client *should* at least retry once to send an *<Alive/>* to the server.

Response for an *<Alive/>* command:

```
<Rsp {id="nr"}
     {status="[ok|error|syntaxError|internalError]"}/>
```

Example:

```
<Rsp status="ok" id="8888"/>
```

3.5.11 Data object structure and function interface information inquiry - <Interface/>

The optional *<Interface/>* command requests a detailed structure information concerning a data object, a data type definition or a function, specified by the *url* attribute. In contrast to the *<Dir/>* command, Interface also returns the signatures of type definitions that are referenced via the "*typeRef*" attribute in object- and list entities.

Format of the *<Interface/>* command:

```
<Req>
   <Interface url="url"/>
</Req>
```

Request attributes:

Attribute	Description
url	The url of the data object or function interface to be retrieved.

Example:

```
<Req id="557">
  <Interface url="Odometer"/>
</Req>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

The detailed response format of the <*Interface/*> command is described in the XML schema laid out in section 6.3.and the description of the service interface profile definitions in chapter 4. A compact and abbreviated form of the <*Interface/*> command response can be found below:

```
<Rsp {id="nr"} status="[ok|error|syntaxError|internalError|
                        notImplemented|noMatchingUrl]"
                  {msg="text"}>
 {<Object url="url"
        {required="[true|false]"}
        {context="[global|session]"}
        {interval="xsd:nonNegativeInteger(0)"}
        characteristic="[static|dynamic|event]">
   0..n * <[Absolute|Activity|Alternative|Binary|Enumeration|ListEntity|
           ObjectEntity|Relative|Text|Time|]/>
 </Object>}
 {<Type url="url">
   0..n * <[Absolute|Activity|Alternative|Binary|Enumeration|ListEntity|
           ObjectEntity|Relative|Text|Time|]/>
 </Type>}
 {<Function url="url" {required="[true|false]"}>
   <In>
     0..n * <[Absolute|Activity|Alternative|Binary|Enumeration|ListEntity|
             ObjectEntity|Relative|Text|Time|]/>
   </In>
   <Out>
     0..n * <[Absolute|Activity|Alternative|Binary|Enumeration|ListEntity|
             ObjectEntity|Relative|Text|Time|]/>
   </Out>
 </Function>}
</Rsp>
```

The response to the <*Interface/*> command shall contain the type information for the requested *url* whereby it is comprised of one <*Function/*>, <*Object/*> or <*Type/*> element.

Response attributes and elements:

Attribute or element	Description
status	Optional attribute that returns the processing status of the request. The ok and error status values apply as described in section 3.3.3. • ***noMatchingUrl***: The server *shall* respond with this status code when no data object or function url is matching the *url* attribute. • ***notImplemented***: The server *shall* respond with this status code when the optional <*Interface/*> command is not supported by the server (see also <*Protocol/*> command).
url	The url of the object or function for which the interface information is provided.
required	Optional attribute that denotes if a function or object must be implemented by a service provider or not. • ***true***: Data object or function that is mandatory to implement by a service provider. • ***false***: Data object or function that is not mandatory to implement by a service provider. A service consumer *shall* check via the <*Dir/*> command if that function or object is available and can be used.
<*Type/*>	Element that contains a type definition that can be referenced via the "*typeRef*" attribute of lists and objects.
<*Object/*>	Element that contains a data object definition.
context	Optional attribute that defines the operating context of this object. • ***global***: Data object that exists once per service. An update to this object will be send to all sessions. • ***session***: Data object that exists once per session (i.e. to a

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

	client-server connection). An update to this object will be send to the corresponding session only.
characteristic	Mandatory attribute that describes the character of the object's value. • ***static***: The values of this object remain constant during the connection. • ***event***: A signal where the appearance / update of the object is the information itself, other to dynamic where the information is within the content. A data object with this attribute set cannot be throttled (see *ival* attribute in <*Subscribe/*> command in section 3.5.4). • ***dynamic***: Data from sources that producinge a continuous stream of sensor information (e.g. a sensor that measures a vehicles speed all 10ms). Updates will be received when for example new sensor information is measured or generated, even if the value content did not change.
<*Function/*>	Element that contains a function interface definition for the requested url.
<*In/*>	Mandatory element, that contains 0..n argument descriptions of required input arguments values for calling the function.
<*Out/*>	Mandatory element, that contains 0..n return values descriptions of values that are returned when the function is executed with a *status* of *ok*.
<*Absolute* *Relative* … *ListEntity* *ObjectEntiy/*>:	Member element definition, as defined in section 4.6. • ***name*:** Name of the member element. • ***typeRef***: Url that links to the <Type url="name"/> definition that *shall* be used as a template for this <*ListEntity/*>, <*ObjectEntity/*> or <*Alternative/*> member to construct a complex data object.
<*!-- @param ->*	Optional descriptive comment for member type definitions (i.e. <*Absolute/*>, <*Relative/*>, etc.)
<*!-- @enum ->*	Optional descriptive comment for each <*Member/*> of a <*Enumeration/*>.
<*!-- @description ->*	Optional descriptive comment for every <*Object/*>, <*Type/*> and <*Function/*>
<*!-- @choice ->*	Optional descriptive comment for each <*Choice/*> of an <*Alternative/*>.

Example:

```
<Rsp id="557">
  <!-- @description Odometer The vehicles overall driven distance -->
  <Object url="Odometer" characteristic="dynamic">
    <!-- @param Odometer The total mileage of the vehicle -->
    <Absolute name="Odometer" unit="km"/>
  </Object>
</Rsp>
```

If the descriptive comments (e.g. @description, @param etc.) are supplied within an interface response these informations *may* be used in client applications or consumed by application users.

Note: More information regarding the descriptive comments can be found in section 4.2.

3.5.12 Authentication for extended access - <Authenticate/>

The optional authentication functionality *may* be used to restrict the access to specific data objects. The access can be restricted for the write and/or read access. The technical authentication scheme is based on RFC 2617 (HTTP Authentication: Basic and Digest Access Authentication) which is commonly used to securely authenticate the access to web resources. A specific "*username*" and "*password*" combination *shall* be used to gain access to a related set of read and/or write rights to

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

specific data objects. A possible way to manage the access rights is *suggested* in this specification document, but in general out of the scope of EXLAP.

3.5.12.1 General access and authentication scheme

If a client wants to access special or restricted data objects and functions it may first authenticate itself via the `<Authenticate/>` command and a `user + password` combination. After a successful authentication the data objects and their urls *shall* become visible and accessible for the `<Dir/>`, `<Interface/>`, `<Get/>`, `<Subscribe/>` and `<Call/>` commands. If an authentication unique to a physical product is required, it is encouraged to use a specific username and password combination for each copy or instance of the product sold. How this can be realized or implemented is out of the scope of this document.

3.5.12.2 User based access management

A user based permission management is suggested, so that `user` with a `password` is allowed to access specific groups. By default there is an implicitly defined user with the name "`default`" and password "`default`" that belongs to the group "`default`" which includes all urls that are not otherwise assigned using the group management.

Proposed XML example configuration:

```
<Users>
  <!-- "default" user - also implicitly defined/used -->
  <User name="default" password="default">
    <Permit group="default"/>
  </User>
  <!-- administrative user with default special access -->
  <User name="admin" password="secret">
    <Permit group="default"/>
    <Permit group="admin"/>
  </User>
</Users>
```

3.5.12.3 Group management

Using the group management every url can be assigned to groups, whereby a user can be assigned to multiple groups (modelled similar to the Unix file access system). By default every url is assigned to the "`default`" group and is accessible without any authentication.

Proposed XML example configuration:

```
<!-- Urls that will be protected  -->
<Access>
  <Assign url="LogEvent" group="admin"/>
  <Assign url="GetProperties" group="admin"/>
  <Assign url="SetProperty" group="admin"/>
  <Assign url="ServiceControl" group="admin"/>
</Access>
```

Note: The proposed XML configurations are to understand as an example format for a XML based configuration of a potential EXLAP service implementation.

3.5.12.4 Authentication "challenge" phase

Authentication is an optional part of the EXLAP specification and done in a two way fashion based on the HTTP basic access authorization scenario [8]. The first phase requests a challenge/response digest authentication to gain access to protected data objects.

- Note 1: There *shall* only be one user credential active at one time.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

- Note 2: A new <*Authenticate/*> sequence overwrites the old one.
- Note 3: Access rights are not additive but exclusive to the current `user` and `password` combination.
- Note 4: Data objects that are not accessible with the current credentials *shall* not be visible or accessible via the <*Get/*>, <*Subscribe/*>, <*Interface/*>, <*Dir/*> and <*Call/*> commands. A "*noMatchingUrl*" error *shall* be returned in case a url can't be accessed with the current credentials.

Format of the `<Authenticate phase="challenge"/>` command:

```
<Req>
  <Authenticate phase="challenge"/>
</Req>
```

Example:

```
<Req id="555">
   <Authenticate phase="challenge"/>
</Req>
```

Response format for `<Authenticate phase="challenge"/>`:

```
<Rsp {id="nr"}
        {status="[ok|error|syntaxError|internalError|
                  notImplemented]"}
        {msg="text"}>
  <Challenge nonce="xsd:base64Binary"/>
</Rsp>
```

Response attributes and elements:

Attribute	Description

status	Optional attribute that returns the processing status of the request. The ok and error status values apply as described in section 3.3.3.
	• ***notImplemented***: The <*Authenticate/*> command is not supported by the server.
	--- 	
	`<Rsp id="555">`
	`  <Challenge nonce="SGFucy1DaHJpc3RpYW4gRnJpY2tl"/>`
	`</Rsp>`

challenge	Response for and <*Authenticate/*> request that contains an element that contains the attribute "*nonce*".
nonce	Contains one-time generated random data of 16 bytes encoded as `xsd:base64Binary`..

Example (not actual data):

```
<Rsp id="555">
  <Challenge nonce="SGFucy1DaHJpc3RpYW4gRnJpY2tl"/>
</Rsp>
```

3.5.12.5 Authentication "response" phase

Optional command that is the second step in the challenge/response digest authentication (*shall* be implemented when `<Authenticate phase="challenge"/>` is implemented).

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Format of the command:

```
<Req>
  <Authenticate phase="response"
      {user="xsd:string"}
      {cnonce="xsd:base64Binary"}
      {digest="xsd:base64Binary"}/>
</Req>
```

Request attributes and elements:

Attribute	Description
cnonce	Attribute specifying a client nonce (random generated 16 bytes used in the challenge-response) represented in `xsd:base64Binary` encoding. Default: Empty string.
user	Attribute specifying a user-id to authenticate against (e.g. `"public"`). Default: Empty string.
digest	Attribute specifying a calculated response digest represented in `xsd:base64binary` encoding. Default: Empty string.

Example (not actual data):

```
<Req id="56">
  <Authenticate phase="response"
                user="admin"
                cnonce="SGFucy1DaHJpc3RpYW4gRnJpY2tl"
                digest="SmVucyBLcnVlZ2Vy"/>
</Req>
```

The digest *shall* be calculated in the following way (all parameters in the MD5 method call are represented as 7-bit strings in UTF-8).

1. `HA1 = MD5(user+":"+password)`
2. `HA2 = MD5(AsciiHex(nonce)+":"+AsciiHex(cnonce))`
3. `DIGEST = MD5(AsciiHex(HA1)+":"+AsciiHex(HA2)).`

Note: The digest *shall* be calculated on both sides in the same fashion. See RFC 2617 for more information concerning the underlying concept (see also http://tools.ietf.org/html/rfc2617 and http://en.wikipedia.org/wiki/Digest_access_authentication).

Response format for `<Authenticate phase="response"/>`:

```
 <Rsp {id="nr"}
          {status="[ok|syntaxError|internalError|error|
                    authenticationFailed]"}
          {msg="text"}/>
```

Response attributes and elements:

Attribute	Description
status	Optional attribute that returns the processing status of the request. The ok and error status values apply as described in section 3.3.3. • ***`authenticationFailed`***: The server *shall* respond with this status code when the authentication failed (i.e. `password` / `user` combination is incorrect, etc.) or *`user`*, *`cnonce`* or *`digest`* attributes are not present or empty.
	`<Rsp id="56" status="authenticationFailed"/>`

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3.6 Data envelope - <Dat/>

A *<Dat/>* envelope is generated in response for an issued *<Subscribe/>* command when the data object represented by the *url* has been updated or directly after a the *<Subscribe/>* request.

- Note 1: If a *<Dat/>* is generated for a *<Subscribe/>* command with the option `content="false"`, no content in the body of the envelope *shall* be supplied. The envelope is just a notification that the content has changed.
- Note 2: The `content="false"` option shall be considered if the content has a large footprint / size. The client can request the updated data via the *<Get>* command.

Format of a *<Dat/>* message:

```
<Dat url="url" {timeStamp="timeStamp"}>
    {data-encoding}
</Dat>
```

Attributes and elements:

Attribute	Description
data-encoding	Contains the payload transferred in the *<Dat/>* envelope (also see section 3.4.2).
url	Denotes the url of the data object (e.g. `url="VehicleSpeed"`) that is updated.
timeStamp	Optional attribute that holds the time of creating the data object in the server. The date/time is represented in the same format as the data type *<Tim/>*. A timestamp is sent, when a *<Subscribe/>* command is issued with a time stamp option, as described in section 3.5.4). Implementors note: If possible, the time *should* specify the creation time of the data object within the resolution of 1 millisecond.

	`<Dat url="VehicleSpeed" timeStamp="2012-09-13T10:11:00.534Z">`
	`  <Abs name="VehicleSpeed" val="100.34"/>`
	`</Dat>`

Example:

```
<Dat url="WGS84Position">
  <Abs name="Latitude" val="52.43801"/>
  <Abs name="Longitude" val="10.75102"/>
  <Enm name="Height" val="61"/>
</Dat>

<Dat url="WheelSpeed">
  <Abs name="FrontLeft" val="45.67"/>
  <Abs name="FrontRight" val="45.64"/>
  <Abs name="RearLeft" val="45.63"/>
  <Abs name="RearRight" val="45.69"/>
</Dat>

<Dat url="ConnectableNetworks">
  <List name="ConnectableNetworks">
    <Elem>
      <Txt name="NetworkLinkName" val="AT&amp;T 3G Network"/>
      <Txt name="NetworkLinkIdentifier" val="1"/>
      <Txt name="NetworkIdentifier" val="1"/>
      <Txt name="ProviderName" val="AT&amp;T USA"/>
      <Txt name="NetworkDeviceName" val="umts0"/>
      <Enm name="LinkType" val="UMTS"/>
      <Enm name="EncryptionState" val="unknown"/>
    </Elem>
  </List>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
</Dat>
```

3.7 Status envelope - <Status/>

The EXLAP *<Status/>* envelopes are used for asynchronous status signalling from the server to the client.

3.7.1 Fields and attributes

The following listed attributes *shall* apply for all *<Status/>* envelopes.

Fields and attributes:

Attribute	Description
msg	Optional attribute that may contain a text (`xsd:string`, encoded in UTF-8) including a detailed textual description regarding the status message.
	`<Status msg="Connection initialization"><Init/></Status>`

3.7.2 Status <Bye/>

This asynchronous status message is used to signal the client that the server will terminate the connection. Immediately after sending the bye status, the server *should* close the underlying transport connection (e.g. TCP/IP socket, WebSocket or Bluetooth connection).

- Note: After receiving the *<Bye/>* status message the client *shall* not process any further communication from the server and *should* also close the connection to the server.

Format:

```
<Status {msg="text"}>
  <Bye/>
</Status>
```

Example:

```
<Status><Bye/></Status>
```

3.7.3 Status <Init/>

This is an asynchronous status to signal the initialization of the connection. The *<Init>* status *shall* be sent after the initial setup of the connection from the server to the client or when the connection is newly initialized on side of the server (e.g. the server restart while holding the communication channel open).

Implementation notes:

- *<Init/>* also indicates that the server has no active subscriptions. Also a previously active alive signalling (*<Alive/>* command) will be deactivated and authenticated resources (*<Auth/>* command) will be reset.
- If a *<Init/>* status message is received by a client, the client *should* send its initialization sequence (e.g. *<Protocol/>* and *<Subscribe/>* command) or any other command (e.g. *<Alive/>*) within 10 seconds to the server.
- If the server receives no request from the client within 10 seconds after sending of an *<Init/>* status message, it *shall* send the *<Init/>* status message again. The server *shall* retry the sending of a *<Init/>* status message at least two times. If the server receives still no request from the client after two retries the server may terminate the transport connection to the client and *may* regard the connection attempt as erroneous.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Format:

```
<Status {msg="text"}>
  <Init/>
</Status>
```

Example:

```
<Status><Init/></Status>
```

3.7.4 Status <Alive/>

Asynchronous status send to the client in response to an active heartbeat signalling (i.e. *<Heartbeat/>* command). Please refer for additional restrictions and conditions to the *<Heartbeat/>* command in section 3.5.9.

Format:

```
<Status {msg="text"}>
  <Alive/>
</Status>
```

Example:

```
<Status><Alive/></Status>
```

3.7.5 Status <Dataloss/>

Asynchronous status sent to the client, signalling that the server experiences an overload or congestion condition, which may result in the loss of data. This status does not make a qualified statement about the how or why of the causing condition. The root cause might depend on the special server implementation, the transport channel and/or the OS under that the server runs.

Therefore the *<Dataloss/>* status *should* only be understood as a rough indication that there is some problem on side of the server, for example:

- The kind or amount of subscribed data objects might produce too much load at the side of the server.
- The sending buffer or queue of the server experiences an overflow, as the underlying connection (i.e. serial link) cannot transport the data fast enough to the client.
- The client might not be able to receive or process the generated data fast enough, so that a congestion at the server occurs.

Format:

```
<Status {msg="text"}>
  <Dataloss/>
</Status>
```

Example:

```
<Status><Dataloss/></Status>
```

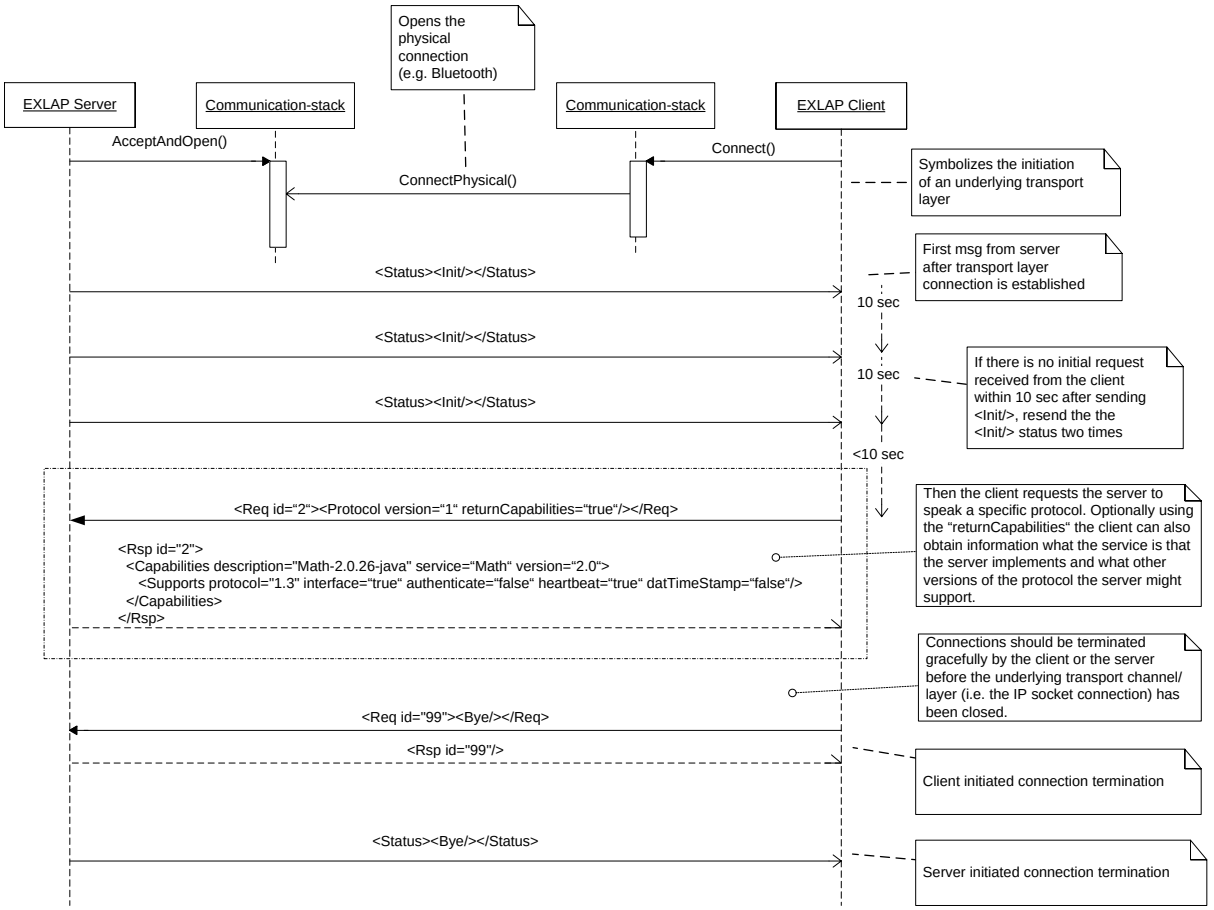
3.8 Overview on practical communication scenarios

The following sections are aimed to give a real-world overview on typical or practical protocol flow or sequences. The data provided in the examples are syntactically correct and reflect the actual data that is transferred “on the wire” between the EXLAP server and the EXLAP client.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3.8.1 Connection initialization (<Protocol/>)

This protocol flow gives an overview of how connections are established and how useful initialization scenarios *should* look like. Special focus is put on the determination of the protocol version and the capabilities of the server.

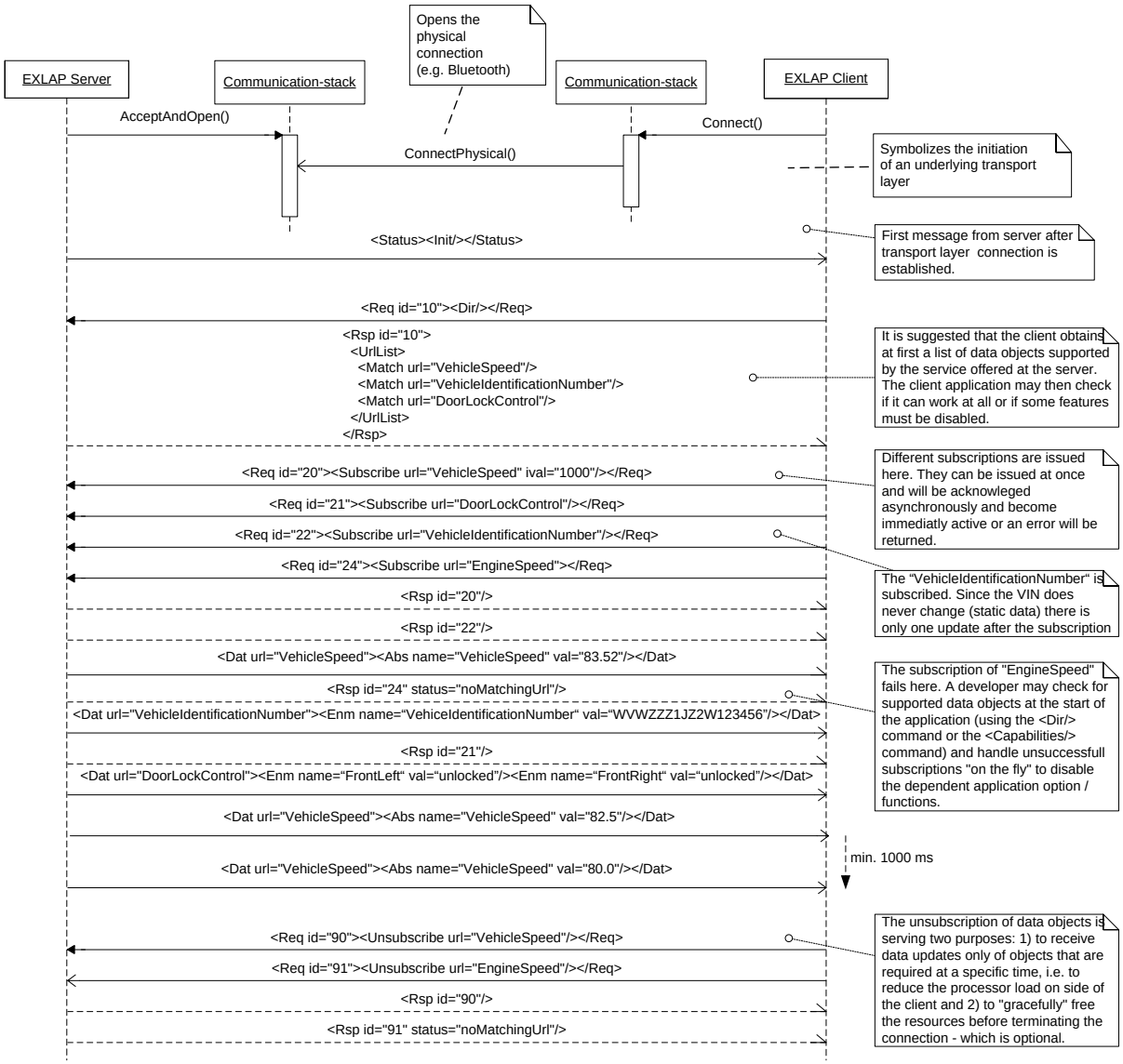


Sequence 2: EXLAP connection initialization protocol flow

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3.8.2 Subscription scenario (<Subscribe/> and <Unsubscribe/>)

Working with event driven data from an EXLAP server is the main aspect of EXLAP. The following sequence chart provides a practical view on how the subscription *should* be handled by a client, with special regard to a robust application that determines at the start of the connection, if the server is able to provide the required object data.

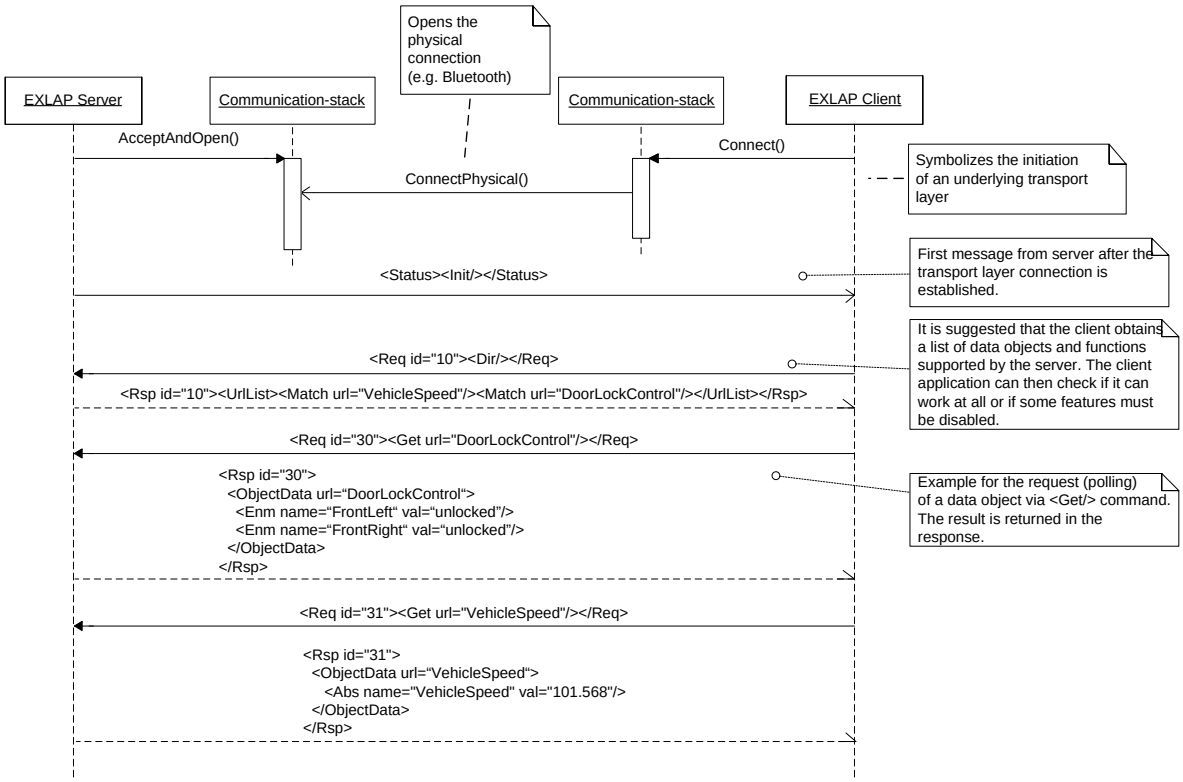


Sequence 3: EXLAP subscribe and unsubscribe protocol flow

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3.8.3 Polling of data objects (<Get/>)

For some scenarios the manual request of data object contents (i.e. polling) can be useful. The following sequence chart gives an overview of practical use-cases for the <Get/> command and its prerequisites.

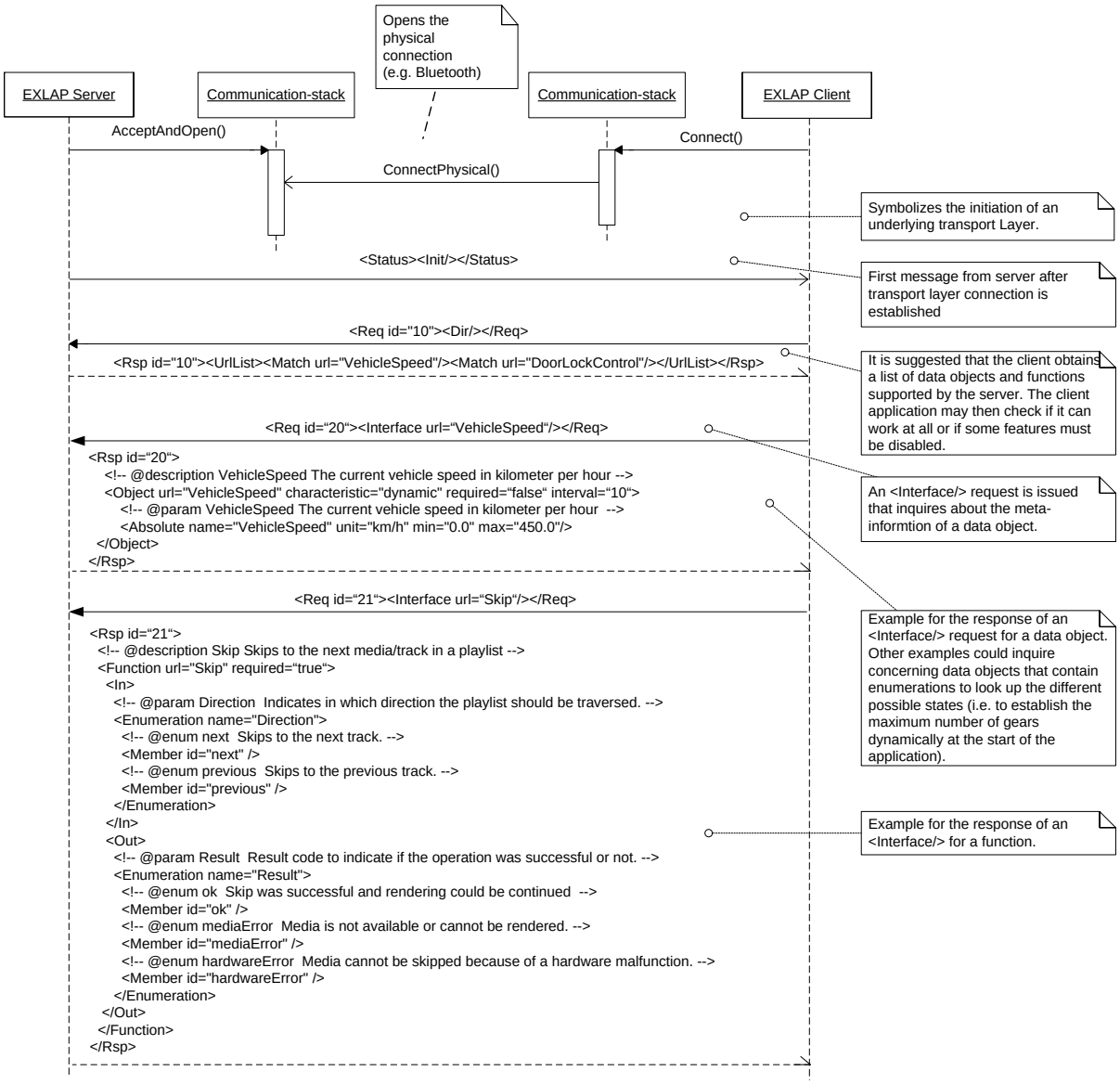


Sequence 4: EXLAP polling of data object protocol flow

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

3.8.4 Retrieving of interface information (<Interface/>)

To enable generic client applications to with a wide range of EXLAP servers, a client can (optionally) request meta-information regarding every data object provided by the server. In most common cases this enables generic tools to display data objects without knowing theirs structure beforehand on base of a specification.

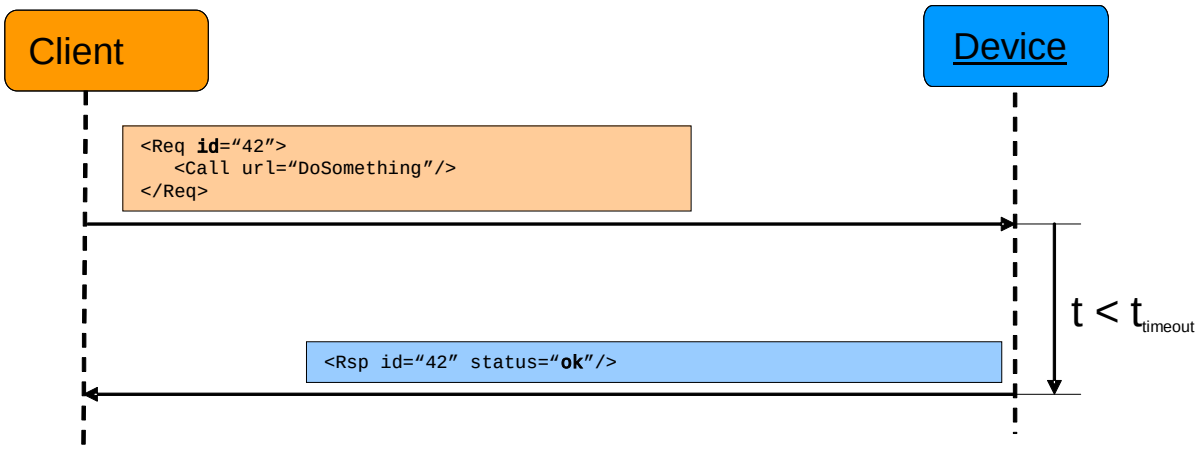


Sequence 5: EXLAP requesting meta information protocol flow

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

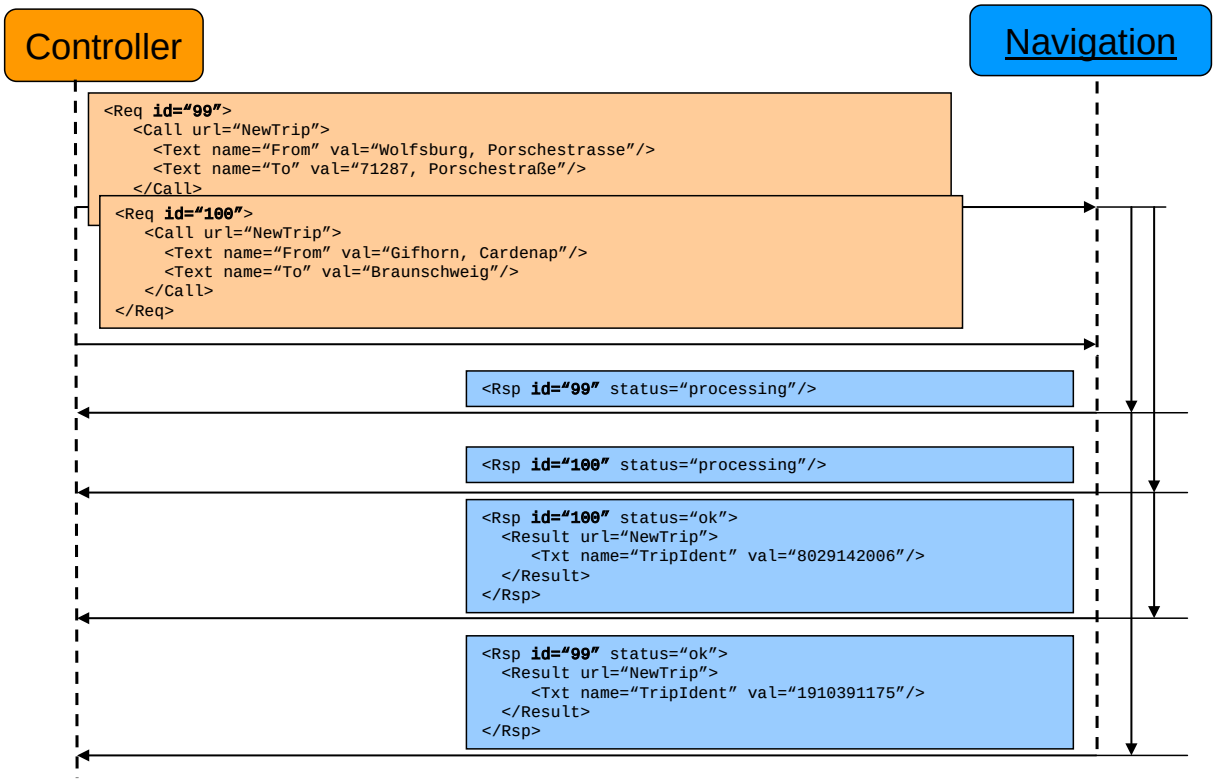
3.8.5 Calling of a RPC style method at the server (<Call/>)

To invoke a remote method in an asynchronous fashion the *<Call/>* command *shall* be used. Following some simple scenarios are shown.



Sequence 6: Simple call with no request and return arguments

The concurrent invocation of a long running method is shown below. Please pay special attention to the intermediate "*processing*" response that is sent because the final return of the method can't be guaranteed within the timeout.

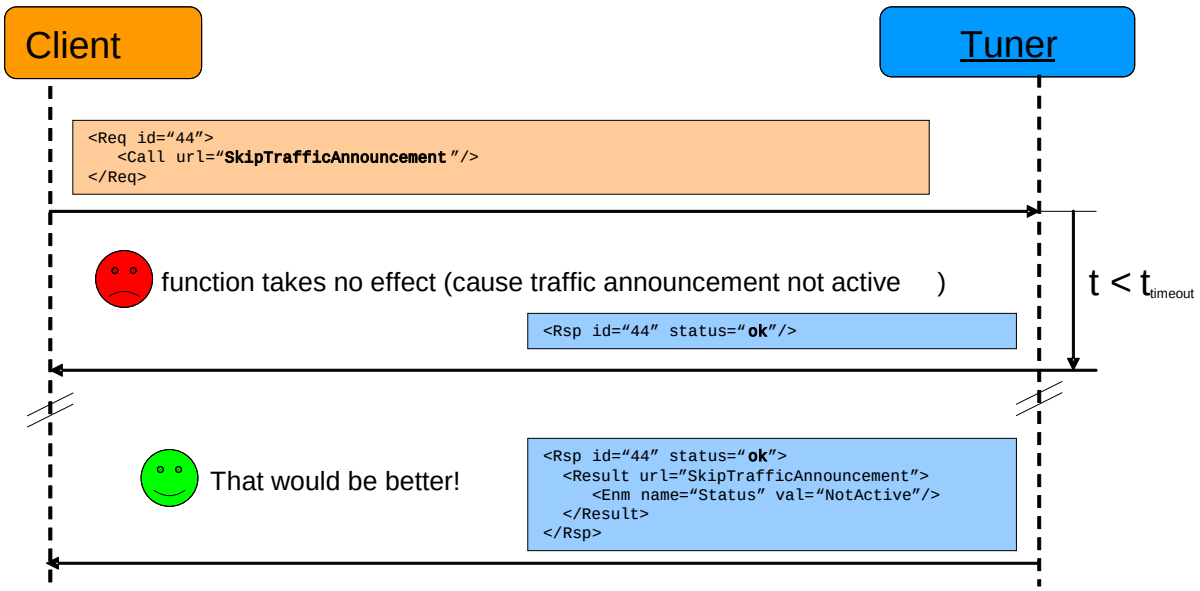


Sequence 7: Concurrent <Call/> with intermediate processing response

Important when working with *<Call/>* and designing the interfaces of the methods is not to be confused with the meaning of the *status* attribute in the *<Rsp/>*. The purpose of the *status* attribute is just to signal if the communication handshake went ok, is in process, has errors or has missing or invalid parameters.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

In the following example the function was successfully executed (`status="ok"`), but takes no effect because there was nothing to skip. The function call in sequence 9 obviously needs to define a return parameter that indicates if the function takes effect or not. (In case traffic announcement is active the call of `SkipTrafficAnnouncement` will instruct the tuner to revert to the original program).



Sequence 8: Example showing how to return results in a <Call/>

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

4 Service interface profiles

This chapter introduces in how services can be described using EXLAP service interface profiles. The so called *service interface profile,* as introduced in section 2.4, will be explained in detail in the following sections.

4.1 Data and meta-data representation

To make the protocol representation as compact and simple as possible, EXLAP distinguishes between the actual encoding of function signatures and data objects on the wire (i.e. `<Abs/>`, `<Rel/>`, etc. within `<Dat/>` envelopes and as introduced in section 2.2.5 and 3.6) and their describing meta-data.

The EXLAP data type system is consciously more abstract/different than known from programming languages (e.g. `float`, `double`, `char`, `string`, ...) The data will get a higher semantics/meaning and thus becomes easier to comprehend. The abstract definition of the member types, as discussed in this section, shall encourage the realization of the definitions in this document in different programming languages, platforms and systems.

The type system used to describe the members in data object and function signatures is based on only seven different basic data types and three compositional types. All types have a name that is used to address the specific information. There are optional or required attributes for each type (e.g. `min`, `max`, `unit`) that give supplemental information to distinguish the meaning of the contained data. Each data type has its own set of attributes. Table 7 provides an overview of the supported data types in EXLAP.

Data type	Description
`<Absolute/>`	An absolute numeric value with a unit (e.g. “km/h”), representing a measurement or a floating point value. Example: A distance, a length, a voltage, a temperature, a pressure etc. Using a resolution of “1” an `Absolute` can also represent an integer number.
`<Activity/>`	A boolean activity (e.g. “`true`”, “`false`”), something that can be activated/deactivated. Example: An on/off switch, a button, an indicator lamp etc. Use a set of activity elements to represent a bitfield.
`<Alternative/>`	An alternative represents one out of a predefined choice of referenced data objects.
`<Binary/>`	An array of bytes that conform to a specified IANA mime type as defined in RFC 2064. Example: An image of the content type `image/jpeg`.
`<Enumeration/>`	A collection of at least two or more predefined constants/literals. Example: A system status (“`deactivated`”, “`idle`”, “`operating`”, “`suspended`”), a choice (“`Otto`”, “`Diesel`”), a direction (“`forward`”, “`backward`”) etc.
`<Relative/>`	A percentage within a minimum/maximum range to describe a fraction of the maximum. The minimum and maximum are distinguished by a label e.g. minimum = “empty”. Example: A level (between “empty” and “full”), a position (between “opened” and “closed”), an intensity (between “dark” and “bright”), a leveling (between “short” and “wide”) etc.
`<ObjectEntity/>`	A object entity is a represents a compositional data type, that is based upon the structure of a referenced data object.
`<ListEntity/>`	A list entity represents compositional data type that can contain 0..n list elements, based upon the structure of a referenced data object.
`<Text/>`	A `Text` represents a variable, an identifier or a fixed string. Example: An identifier (Vehicle Identification Number “WVWZZZ1JZ2W123456”), a code (ISO 639 language code “de”, “en”; colouring code “RR22”), or a formatted number (“+49-5361-9-0”) etc.
`<Time/>`	A `Time` represents a date (4-digit year, month, day) and time (hour, minute, seconds, millis, micros) and a mandatory GMT time offset if a time information is present.

Table 7: Overview of the EXLAP service interface member element data types

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

The following sections will explain the concept of how to specify a service interface profile based upon an explanation of the meta-data describing types like *<Absolute/>*, *<Relative/>* etc.

4.2 Service interface profiles

Services, as introduced in section 2.4, are a set of functions and/or a set of data objects that are implemented by an EXLAP server. To describe a service a so-called service interface profile should be defined.

The basis for such a profile is the XML schema as defined in section 6.5. The result looks like the following example:

```
<Profile name="Servicename"
         version="2.0"
         implementation="derivative"
         xmlns="http://exlap.de/v1/profile">

  <About>A textual description of the service</About>

  <Object url="Name" ...>
    ...
  </Object>

  <Type url="Name" ...>
    ...
  </Type>

  <Function url="Name" ...>
    ...
  </Function>

</Profile>
```

A *<Profile/>* defines the service *name* and a *version* plus an optional *implementation* (i.e. an artifact identifier that marks a derivate). It contains an optional *<About/>* element describing the service and any number of *<Object/>, <Type/>* and *<Function/>* elements, defining the available function set and the data objects. Each of the functions and data objects has a uniqe *url* so that applications (and human readers) can easily find the information that they want.

Attribute	Description
name	The mandatory name of the service
version	The optional version of the service in the format [major].[minor] with the default of “0.0”.
implementation	The optional name of the implementation artifact in case this is not the core service definition that is (partially) implemented. The default is and empty string.
<About/>	A single and optional element that can contain a small or comprehensive service description.
<Object/>	0..n data object definitions.
<Type/>	0..n data type definitions.
<Function/>	0..n function definitions.

Best practice: A *url* shall have at least 3 and not more than 32 characters (see section 6.4). The use of acronyms and abbreviations *should* be avoided for the ease of understanding.

To provide additional documentation within the service interface profile it is possible to add XML comments (i.e. *<!-- -->* sections) with descriptive markers like that of JavaDoc or Doxygen. The currently supported comments are listed in Table 8.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Attribute	Parameter	Description
@description	The name of the type, function or object to document	Documents an *<Object/>*, *<Type/>* or *<Function/>*.
@param	The name of the member element to document	Documents the member element of an *<Object/>*, *<Type/>* or parameter of a *<Function/>*.
@enum	The name of the enumeration member id to document	Documents an *<Enumeration/>* member value *id* (see also section 4.6.5).
@choice	The name of the alternative choice to document	Documents an *<Alternative>* choice *typeRef*.

Table 8: Optional XML description comment markers

Example of a potential `Play` function of a media related service:

```
<!-- @description Play Play a track -->
<Function url="Play" required="true">
  <In>
    <!-- @param Playlist  A list of playlist entries containing track
                          identifiers to be played.  -->
    <ListEntity name="Playlist" typeRef="PlaylistEntry" />
    <!-- @param Index  The track index within the playlist where playing
                       should start; index "1" marks first track -->
    <Text name="Index" regExp="[1-9][0-9]*" />
    <!-- @param Position  Determines the seek position of the track where
                          playing should start. -->
    <Absolute name="Position" unit="s" />
  </In>
  <Out>
    <!-- @param Result  Result code to indicate if the operation was
                        successful or not. -->
    <Enumeration name="Result">
      <!-- @enum ok  Rendering started successfully. -->
      <Member id="ok" />
      <!-- @enum mediaError  Media with the specified track identifier is
                             not available or cannot be rendered. -->
      <Member id="mediaError" />
      <!-- @enum hardwareError  Media cannot be rendered because of a
                                hardware malfunction. -->
      <Member id="hardwareError" />
    </Enumeration>
  </Out>
</Function>
```

Using these XML comment annotations code generating tools can create source code comments for generated code.

4.3 Data object definition - <Object/>

A service can describe any number of data objects using *<Object/>* elements in a service *<Profile/>*. A data object has a unique *url* name and additional attributes, describing the object, based upon the profile XML schema as defined in section 6.5.

Note: EXLAP before version 1.3 allowed 0..n *<Description/>* elements within the *<Object/>* element. They should be gracefully ignored to allow EXLAP 1.3 implementations the processing of older service profile definitions.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Format for the definition of an <Object/>:

```
<Object url="name"
        {context="[global|session]"}
        characteristic="[static|dynamic|event]"
        {interval="xsd:positiveInteger-canonical(0)"}
        {required="[true|false]"}>

  1..n * <[Absolute|Activity|Alternative|Binary|Enumeration|
           ListEntity|ObjectEntity|Relative|Text|Time]/>
</Object>
```

Example:

```
<Object url="VehicleSpeed" context="global" characteristic="dynamic"
        interval="10" required="false">
  <Absolute name="VehicleSpeed" unit="km/h"/>
</Object>
```

The example shows an *<Object/>* representing the speed of the car that is represented by a *<Relative/>* element. Applications are able to subscribe (i.e. "*read*") these information and "*dynamic*" means the values contained are subject of change.

Attribute or element	Use	Default	Description
url	required		Unified identifier to access the desired data object e.g. "EngineSpeed" from a client. Uniqueness of the string used in *url* shall be assured in the context of a single service interface profile.
required	optional	false	The optional attribute *required* denotes if an object must be implemented by a service provider or not. If an object is not mandatory to implement, a service consumer *shall* check via the *<Dir/>* command if that object is available and can be used.
interval	optional	0	The optional attribute *interval* is designed to give subscribing client applications an idea how often the data will be updated. This attribute describes the minimum time in milliseconds between two consecutive updates e.g. the specified mean cycle time of the data source if applicable. The default interval "0" does mean that there is no information regarding the update interval available.
context	optional		Defines the operating context of this object. • ***global***: this data object exists once per service. An update to this object will be send to all sessions.. • ***session***: this data object exists once per session. An update to this object will be send to the corresponding session only.
characteristic	required		Describes the characteristic of the object value. • ***static***: The values of this object remain constant during the connection. • ***event***: A signal where the appearance / update of the object is the information itself, other to dynamic where the information is within the content. • ***dynamic***: Data from sources that produce a continuous stream of sensor information (e.g. a sensor that measures a vehicles speed all

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

			10ms). Updates will be received when for example new sensor information is measured or generated, even if the value content did not change. A data object with this attribute set cannot be throttled (see *ival* attribute in <*Subscribe/*> command in section 3.5.4).
<*Absolute/*> <*Activity/*> <*Alternative/*> <*Binary/*> <*Enumeration/*> <*ListEntity/*> <*ObjectEntity/*> <*Relative/*> <*Text/*> <*Time/*>	required		Choice of *1..n* member type elements defining the content that object contains as described in section 4.6.

Table 9: Elements and attributes of element <Object/>

Multiple member elements *may* be necessary where multiple instances exists e.g. a potential `DoorPosition` object, where each door requires an own position information. Multiple members are also required when the data types of the members are heterogeneous resp. of different type. An example for that is a geographical position where two physical values (latitude, longitude), two hemisphere information (North/South and East/West and additional a *time of generation* information) are needed.

In addition data objects can have a global session context or a local session context (i.e. <*Object context="global "/*> or <*Object context="session "/*>). A global session context denotes that a data object within the services is applicable for all sessions together. Any update to that object will be sent to all the sessions. If a data object context is per session then each session will have an individual value for that object maintained in the service.

Example:

```
<Object url="DoorPosition" context="global" characteristic="event"
        required="false">
  <Enumeration name="FrontLeft">
    <Member id="opened"/>
    <Member id="closed"/>
    <Member id="ajar"/>
  </Enumeration>
  <Enumeration name="FrontRight">
    <Member id="opened"/>
    <Member id="closed"/>
    <Member id="ajar"/>
  </Enumeration>
  <Enumeration name="RearLeft">
    <Member id="opened"/>
    <Member id="closed"/>
    <Member id="ajar"/>
  </Enumeration>
  <Enumeration name="RearRight">
    <Member id="opened"/>
    <Member id="closed"/>
    <Member id="ajar"/>
  </Enumeration>
</Object>

<Object url="WGS84Position" context="global" characteristic="dynamic"
        required="false">
  <Absolute name="Latitude" unit="arcDegree"/>
  <Absolute name="Longitude" unit="arcDegree"/>
  <Absolute name="Height" unit="m"/>
</Object>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

4.4 Type definitions - <Type/>

Using a type, frequently used structures of member elements can be reused (i.e. referenced) in *<Object/>* and *<Function/>* definitions. While a type is quite similar to an *<Object/>* definition, regarding its *url* and the definition of member elements, it cannot be subscribed. A syntactical complete definition can be found in the XML schema as defined in section 6.5.

Note: EXLAP before version 1.3 allowed 0..n *<Description/>* elements within the *<Type/>* element. They should be gracefully ignored to allow EXLAP 1.3 implementations the processing of older service profile definitions.

Format for the definition of an <Type/>:

```
<Type url="name">
  1..n * <[Absolute|Activity|Alternative|Binary|Enumeration|
          ListEntity|ObjectEntity|Relative|Text|Time]/>
</Type>
```

Example:

```
<Type url="Track">
  <Text name="Artist" regExp=".+"/>
  <Text name="Title" regExp=".+"/>
  <Text name="Album" regExp=".*"/>
  <Text name="Year" regExp="[0-9]*"/>
</Type>

<Function url="TrackList">
  <ListEntity name="Tracks" typeRef="Track"/>
</Function>
```

The example shows an *<Type/>* representing a track of a media service, that is then used as base for a track list in a list using the "*typeRef*" attribute.

Attribute or element	Use	Default	Description
url	required		Unified identifier for the type (e.g. *WGS84Position*). Uniqueness of the string used in *url* shall be assured in the context of a single service interface profile.
<Absolute/> *<Activity/>* *<Alternative/>* *<Binary/>* *<Enumeration/>* *<ListEntity/>* *<ObjectEntity/>* *<Relative/>* *<Text/>* *<Time/>*	required		Choice of *1..n* member type elements defining the content that the type contains as described in section 4.6.

Table 10: Elements and attributes of element <Type/>

The usage of types is generally limited to the *<ObjectEntity/>* and *<ListEntity/>* members which must reference a type to defines their content structure.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

4.5 Function definition - <Function/>

A service can describe any number of functions using *<Function/>* elements in a service *<Profile/>*. A function has a unique *url* and additional attributes, describing the object, based upon the profile XML schema as defined in section 6.5.

Note: EXLAP before version 1.3 allowed 0..n *<Description/>* elements within the *<Function/>* element. They should be gracefully ignored to allow EXLAP 1.3 implementations the processing of older service profile definitions.

Format for the definition of an <Function/>:

```
<Function url="name" required="[true|false]">
   <In>
      0..n * <[Absolute|Activity|Alternative|Binary|Enumeration|
               ListEntity|ObjectEntity|Relative|Text|Time]/>
   </In>

   <Out>
      0..n * <[Absolute|Activity|Alternative|Binary|Enumeration|
               ListEntity|ObjectEntity|Relative|Text|Time]/>
   </Out>
</Function>
```

Example:

```
<!-- @description Seek Sets the 'Position' of the current track. -->
<Function url="Seek" required="false">
  <In>
    <Absolute name="Position" unit="s"/>
  </In>
  <Out>
    <Enumeration name="Result">
      <Member id="ok" />
      <Member id="mediaError" />
      <Member id="hardwareError" />
    </Enumeration>
  </Out>
</Function>
```

The example shows an *<Function/>* representing a seek function of a potential media service that has an *<Absolute/>* input element defining the position to seek to and one output element defining the result of the operation.

Attribute	Use	Default	Description
url	required		Unified identifier to access the desired function e.g. "Seek" from a client. Uniqueness of the string used in *url* shall be assured in the context of a single service interface profile.
required	optional	false	The optional attribute *required* denotes if a function or object must be implemented by a service provider or not. If a function or object is not mandatory to implement a service consumer *shall* check via the *<Dir/>* command if that function or object is available and can be used.
<In/>	required		This element section contains the input (i.e. calling) arguments of the function in for from 0..n *<Absolute/>*, *<Relative/>*, *<Text/>*, … member element definitions.
<Out/>	required		This element section contains the output (i.e. return) arguments of the function in for from 0..n *<Absolute/>*, *<Relative/>*, *<Text/>*, … member

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

			element definitions.
<Absolute/> *<Activity/>* *<Alternative/>* *<Binary/>* *<Enumeration/>* *<ListEntity/>* *<ObjectEntity/>* *<Relative/>* *<Text/>* *<Time/>*	optional		0..n object member type elements defining the content of the *<In/>* and *<Out/>* arguments elements as described in section 4.6.

Table 11: Attributes of element <Function/>

Best practice note: Functions *should* return a processing status in the respose (i.e. *<Out/>* element section) using a *<Enumeration/>* with the name "*Result*" and and the member "*ok*" and optionally "*error*" if no specific error cases may be defined.

4.6 Service interface member element data types

Member element types are *<Absolute/>*, *<Activity/>*, *<Alternative/>*, *<Binary/>*, *<Enumeration/>*, *<ObjectEntity/>*, *<ListEntity/>*, *<Relative/>*, *<Text/>* and *<Time/>* as introduced in section 4.1 and are explained in detail in following sections.

The following attributes are applicable to all member element types.

Attribute	Use	Default	Description
name	required		Alias name to access the member element. Name uniqueness shall be assured within a single *<Object/>*, a single *<Type/>* or *<In/>* and *<Out/>* element section of a single *<Function/>*.
required	optional	true	Stating that this member is required (`true`) or can be omitted (`false`) in the actual service implementation. If *required* is *false*, the consuming service or client must handle the absence of this member in function calls, function results and subscribed data objects. The service and member description shall give additional information how to handle the absence or presence of optional arguments or values.

4.6.1 Absolute

An *<Absolute/>* is used to represent a measurement or a number value and will be represented by a *<Abs/>* in the encoding on the wire. For example: A representation of the engine speed in revolutions per minute or the exterior temperature in degree Celsius.

The data member shall provide a physical *unit*. The recommended units of measurement are predefined in section 6.4. Since the *unit* attribute is required for the definition of *<Absolute/>* members the unit "1" is used for instances that describe a number (e.g. number of cylinders). The optional attributes *min* and *max* specify the minimum and maximum range of the value. The minimum and maximum values indicated for the specific *<Absolute/>* instances are intended to give users an idea of a reasonable value. The optional attribute *resolution* informs client applications about the resolution of measured values. It specifies the minimum value change between two consecutive updates (e.g. 1 for integer numbers).

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Attribute	Use	Default	Description
unit	required		Describing the physical unit corresponding to the value of this <*Absolute/*> instance if applicable.
min	optional	-INF	Minimum value this <*Absolute/*> can become.
max	optional	INF	Maximum value this <*Absolute/*> can become.
resolution	optional	0	The smallest possible value change.

Table 12: Attributes of element <Absolute/>

Format for the definition of an <*Absolute/*>:

```
<Absolute name="name"
          {required="true|false"}
          unit="[xsd:string]"
          {min="xsd:double(-INF)"}
          {max="xsd:double(INF)"}
          {resolution="xsd:double(0)"}/>
```

Example:

```
<Object url="NumberOfAxles" context="global" characteristic="static">
  <Absolute name="NumberOfAxles" unit="1" min="1" max="5"/>
</Object>

<Object url="EngineSpeed" context="global" interval="10"
        characteristic="dynamic">
  <Absolute unit="1/min" name="EngineSpeed" resolution="0.25"/>
</Object>
```

The example `NumberOfAxles` is a static data object, describes the number of axles of a vehicle. The object `EngineSpeed` is a `dynamic` value that may have an update interval around `10` milliseconds. The unit is *1* since it is only a integer number. Other units could be *1*, *m*, *km*, *rad*, *km/h*, etc.

Example of more complex data objects:

```
<Object url="WheelSpeed" context="global" characteristic="dynamic">
  <Absolute unit="km/h" name="FrontLeft" min="0.0" max="450.0"/>
  <Absolute unit="km/h" name="FrontRight"/>
  <Absolute unit="km/h" name="RearLeft"/>
  <Absolute unit="km/h" name="RearRight"/>
</Object>

<Object url="BatteryVoltage" context="global" characteristic="dynamic">
  <Absolute unit="V" name="Main" min="5.0" max="20.0"/>
  <Absolute unit="V" name="Aux" min="5.0" max="20.0"/>
</Object>
```

Implementation note: It is not advised to use an <*Absolute/*> to represent identifiers. In this context the <*Text/*> type offers more flexibility and expressiveness. Regarding units it is recommended to define reusable unit strings for a set of services in the same context.

4.6.2 Activity

An <*Activity/*> represents an instance similar to a boolean that can be “activated” or “deactivated” e.g. the brake applied status or a lamp on-off switch and will be represented by a <*Act/*> in the encoding on the wire. This data type has just a name and doesn’t have any further descriptive attributes. A set of activity elements shall be used to represent a bit field structure.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Format for the definition of an <*Activity/*>:

```
<Activity name="name"
          {required="true|false"}
/>
```

Example:

```
<Object url="BrakeActuation" characteristic="dynamic">
  <Activity name="BrakeActuation"/>
</Object>
```

The <Activity/> presented in the example indicated if a brake pedal is pressed or not. To represent if the brake system is in order (i.e. a potential object BrakeSystemState) a <*Enumeration/*> may be used and to provide the percentage the brake pedal is pressed a <*Relative/*> data type should be used.

4.6.3 Alternative

An <*Alternative/*> is a data structure that is similar to a variant or C-style union. Using an alternative, it is possible to embed one out of at a predefined set least two or types. These types can then have different elements, so that it is possible to dynamically construct variable data structures and lists that hold different items.

The 2..n <*Choice/*> elements define the possible <*Type/*> elements that later can be transported and appear within the <*Alt/*> element in the encoding on the wire.

Attribute or element	Use	Default	Description
<*Choice/*>	required		Choice element that contains one of the types that are valid choices within its *typeRef* field.
typeRef	required		The attribute value must hold the name of a <*Type/*> that is defined in the same interface service profile.

Table 13: Attributes of element <Enumeration/>

Format for the definition of an <*Alternative/*>:

```
<Alternative name="name"
             {required="true|false"}>
  2..n * <Choice typeRef="url-of-a-type"/>
</Alternative>
```

Example:

```
<Type url="Animal">
  <Text name="Name" regExp="*."/>
  <Alternative name="Animal">
    <Choice typeRef="Dog"/>
    <Choice typeRef="Cat"/>
    <Choice typeRef="Horse"/>
  </Alternative>
</Type>

<Type url="Dog">
  <Enumeration name="Race">
    <Member id="Golden Retriever"/>
    <Member id="Greyhound"/>
    <Member id="German Sheppard"/>
  </Enumeration>
  <Relative name="BarkVolume"/>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
</Type>

<Type url="Cat">
  <Enumeration name="Race">
    <Member id="Siamese"/>
    <Member id="Manx"/>
    <Member id="Felix catus"/>
  </Enumeration>
  <Relative name="PurrVolume"/>
</Type>
```

4.6.4 Binary

An *<Binary/>* represents an array of bytes and will be represented by a *<Bin/>* in the encoding on the wire. The only additional attribute of a binary is the *contentType* field, which describes the format of the binary content.

Attribute	Use	Default	Description
contentType	required		The IANA mime type of the content as defined in RFC 2064. Example: An image of the content type *image/jpeg.* If the content is application specific, *application/octet-stream* shall be used.

Table 14: Attributes of element <Activity/>

Format for the definition of an *<Binary/>*:

```
<Binary name="name"
        {required="true|false"}
        contentType="rfc-2064-mime-type"/>
```

Example:

```
<Type url="Person">
  <Text name="Firstname"/>
  <Text name="Lastname"/>
  <Binary name="AvatarImage" contentType="image/jpeg"/>
</Object>
```

The <Binary/> presented in the example later holds an image in the jpeg format of an avatar image that should be used in context of the person. While this method of image embedding is not suggested, a HTTP url to a location where the image can be downloaded may be more appropriate, it is a workable example.

4.6.5 Enumeration

An *<Enumeration/>* is a collection of at least two or more predefined constants/literals and will be represented by a *<Enm/>* in the encoding on the wire. The current state of the *<Enumeration/>* is represented by one constant of the predefined set.

In 2..n *<Member/>* elements a set of predefined constants denoted by attribute *id* must be defined. These members then make up the enumeration values. The uniqueness of a member identification string (*id*) within the same enumeration context shall be assured.

Attribute or element	Use	Default	Description
<Member/>	required		Enumeration element that contains one of the states of the enumeration within its *id* field.
id	required		The attribute value is a unique constant that represents

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

			one of the states of the enumeration. Uniqueness *shall* be assured in a single enumeration.

Table 15: Attributes of element <Enumeration/>

Format for the definition of an *<Enumeration/>*:

```
<Enumeration name="name"
             {required="true|false"}>
  2..n * <Member id="id"/>
</Enumeration>
```

Example:

```
<Object url="AntiLockBrakeSystem" context="global" characteristic="dynamic">
  <Enumeration name="AntiLockBrakeSystem">
    <Member id="deactivated"/>
    <Member id="idle"/>
    <Member id="operating"/>
  </Enumeration>
</Object/>
```

4.6.6 Relative

A *<Relative/>* type is a percentage within a minimum/maximum range (usually 0.0 up to 1.0) to describe a fraction of the maximum and will be represented by a *<Rel/>* in the encoding on the wire. The intention of the minimum and maximum values is described by two additional attributes *minLabel* and *maxLabel*. The optional attribute *resolution* informs client applications about the resolution of measured values. It specifies the minimum value change between two consecutive updates.

Attribute	Use	Default	Description
min	required		Minimum value of this *<Relative/>* instance.
max	required		Maximum value of this *<Relative/>* instance.
minLabel	required		Qualifies the minimum value e.g. “empty”.
maxLabel	required		Qualifies the maximum value e.g. “full”.
resolution	optional	0	The smallest possible value change.

Table 16: Attributes of element <Relative/>

Format for the definition of an *<Relative/>*:

```
<Relative name="name"
          {required="true|false"}
          {min="xsd:double"}
          {max="xsd:double"}
          {minLabel="name"}
          {maxLabel="name"}
          {resolution="xsd:double(0)"}/>
</Relative>
```

Example:

```
<Object url="WaterInGlasLevel" context="global" characteristic="dynamic">
  <Relative name="WaterInGlasLevel"
            min="0.0" minLabel="empty"
            max="1.0" maxLabel="full"/>
</Object>
```

In the example the data object `WaterInGlasLevel` defines the minimum value (0.0) as “`empty`” and the maximum value (1.0) as “`full`”.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

4.6.7 Text

A *<Text/>* represents a variable or fixed string and will be represented by a *<Txt/>* in the encoding on the wire.

For example: A vehicle identification number (VIN), a name of a radio station, a telephone number or a title of a song. An optional attribute *regExp* is intended to specify a valid format for a *<Text/>* (using the Perl 5 regular expression syntax).

Attribute	Use	Default	Description
regExp	optional	.*	Specifies the allowed format of the *<Text/>* string by using the Perl 5 regular expression syntax.

Table 17: Attributes of element <Text/>

Format for the definition of an *<Text/>*:

```
<Text name="name"
     {required="true|false"}
     {regExp="regular-expression"}/>
```

Example:

```
<Type url="Track" context="global">
  <Text name="Artist" regExp=".+"/>
  <Text name="Title" regExp=".+"/>
  <Text name="Album" regExp=".*"/>
  <Text name"Year" regExp="[0-9]*"/>
</Type>
```

4.6.8 Time

A *<Time/>* is used to represent date and/or time information e.g. manufacturing date or the local time and will be represented by a *<Tim/>* in the encoding on the wire.

The required attribute *isLocalTime* describes if the time zone *shall* be ignored or is unreliable (`true`) or is reliable (`false`). The background for this attribute is that the ISO 8601 does not specify a time without a time zone. This attribute now defines that the time given is the local time at the point of origin, whereby the time zone at the origin is not known or not available for the service that produces the time information.

Attribute	Use	Default	Description
isLocalTime	required		This attribute *shall* be *true*, if the time shall be interpreted only as a time while the time zone information is ignored (even it it might be present in form of a 'Z' (zulu) or "+00:00", etc.).

Table 18: Attributes of element <Time/>

Format for the definition of an *<Time/>*:

```
<Time name="name"
      {required="true|false"}
      isLocalTime="[true|false]"/>
```

Example:

```
<Object url="LocalTime" context="global" characteristic="dynamic">
  <Time name="LocalTime" isLocalTime="true"/>
</Object
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

In the example a service provides its time which is only available without time zone information. This could be a vehicle that has only a digital clock but no option to enter or specify a time zone.

4.6.9 ObjectEntity

A <*ObjectEntity/*> is a composite type that is used to reference to existing type <*Type/*> definitions within other objects and will be represented by a <*Obj/*> in the encoding on the wire.

The *typeRef* attribute points (i.e. references) to the data object <*Type/*> definition that shall be the template for the data structure that shall be represented.

Attribute	Use	Default	Description
typeRef	required		A name for a <*Type/*> definition that will be used as a structural template (i.e. like a reference to a `struct` in the C programming language).

Table 19: Attributes of element <ObjectEntity/>

Format for the definition of an <*ObjectEntity/*>:

```
<ObjectEntity name="name"
              {required="true|false"}
              typeRef="object-name"/>
```

Example:

```
<Type url="WGS84Position">
  <Absolute name="Longitude" unit="arcDegree" min="-180" max="180"/>
  <Absolute name="Latitude" unit="arcDegree" min="-90" max="90"/>
  <Absolute name="Height" unit="m" min="-1000" max="18000"/>
</Type>

<Object url="CurrentCarPosition" characteristic="dynamic"
        context="global">
  <ObjectEntity name="CurrentCarPosition" typeRef="WGS84Position"/>
  <Absolute name="Heading" unit="arcDegree" min="0" max="360"/>
</Object>
```

In this example the data object `CurrentCarPosition` reuses the data object definition `WGS84Position` and adds another data member `Heading` to the new composite type.

Resulting <*Dat/*> encoding on the wire for the example above:

```
<Dat url="CurrentCarPosition"/>
  <Obj name="WGS84Position"/>
    <Abs name="Longitude" val="10.34560"/>
    <Abs name="Latitude" val="52.886351"/>
    <Abs name="Height" val="34.65"/>
  </Obj>
  <Abs name="Heading" val="130.76"/>
</Object>
```

4.6.10 ListEntity

A <*ListEntity/*> is a composite type that is used to model lists based upon <*Type/*> definitions and will be represented by a <*List/*> and <*Elem/*> in the encoding on the wire.

The *typeRef* attribute points (i.e. references) to the data <*Type/*> that shall be the template for the list elements.

Attribute	Use	Default	Description

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

typeRef	required		A name to an <*Type/*> definition hat will be used as a structural template (i.e. like a reference to a `struct` in the C programming language) for the content.

Table 20: Attributes of element <ListEntity/>

Format for the definition of an <*ListEntity/*>:

```
<ListEntity name="name"
            {required="true|false"}
            typeRef="object-name"/>
```

Example:

```
<Type url="Track" characteristic="dynamic" context="type">
  <Text name="TrackIdentifier" regExp=".*"/>
  <Text name="Title" regExp=".*"/>
  <Text name="Artist" regExp=".*"/>
  <Absolute name="Length" unit="s"/>
  <Text name="Genre" regExp=".*"/>
  <Text name="Album" regExp=".*"/>
  <Text name="Year" regExp="((19|20)[0-9][0-9])?"/>
</Track>

<Function url="GetTracks" context="global">
  <In>
    <Text name="StartWith" regExp="[1-9][0-9]*"/>
    <Text name="MaxElements" regExp="[1-9][0-9]*"/>
  </In>
  <Out>
    <ListEntity name="Tracks" typeRef="Track"/>
  </Out>
</Function>
```

In this example the function `GetTracks` reuses the data object definition `Track` in its output parameters to provide a list of `Track` that matches the criteria provided in the input parameters of the function (i.e. `StartWith` and `MaxElements`).

Resulting function <*Result/*> encoding on the wire for the example above:

```
<Result url="GetTracks">
  <List name="Tracks">
    <Elem>
      <Txt name="TrackIdentifier" val="64ghGS"/>
      <Txt name="Title" val="Whole Lotta Love"/>
      <Txt name="Artist" val="Led Zeplin"/>
      <Abs name="Length" val="333"/>
      <Txt name="Genre" val="Classic Rock"/>
      <Txt name="Album" val="Led Zeplin II"/>
      <Txt name="Year" val="1969"/>
    </Elem>
    <Elem>
      <Txt name="TrackIdentifier" val="54hjkZ"/>
      <Txt name="Title" val="Ramble On"/>
      <Txt name="Artist" val="Led Zeplin"/>
      <Abs name="Length" val="275"/>
      <Txt name="Genre" val="Classic Rock"/>
      <Txt name="Album" val="Led Zeplin II"/>
      <Txt name="Year" val="1969"/>
    </Elem>
  </List>
</Result>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

4.7 Notes regarding the “required” attribute in functions, objects and members

The *required* attribute owes its existence in the heritage of the protocol – the vehicle environment, where the vehicles follow a general set of principles but are still differently equipped. Using this attribute a service developer can define if a function or object is required (i.e. mandatory or essential) for the functioning and usage of the service. The same is valid for the members in objects and function calls – or results. There might be members that are required and other that are optional. The optional members can have an assumed or prior defined default value.

Using the *required* attribute when designing a service, results in a *master* service interface that defines the mandatory superset of all functions, objects and types. When implementing a *derived* real world service, based upon this *master* interface, this interface must then omit all (non-required) parts in its service interface *subset* which it does not implement.

A client that connects to a (*subset*) real world service can still use this service like any other, but must expect that functionality that is not required is not present. To inquire beforehand which functionality is not present the <*Dir/*> and <*Interface/*> commands can be used.

Summary: Non required functions, objects or members that are not supported by the specific implementation are in consequence not present in the directory or in the interface description.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

4.8 Example Service “Math”

This example shows the service interface profile for a very simple service: The `Math` service – offering two functions `Add` and `Div` plus a data object that provides dynamic updated information regarding the `Statistics` of the service.

```
<Profile xmlns="http://exlap.de/v1/protocol"
         name="Math" version="1.1">

  <About>A simple mathematical service</About>

  <!-- @description Statistics Object that represents the statistics of the service
                              operations. -->
  <Object context="global" url="Statistics" characteristic="dynamic">
    <!-- @param TotalSum The total sum of all operation results. -->
    <Absolute name="TotalSum" unit="1"/>
    <!-- @param OperationsCount The number of performed operations. -->
    <Absolute name="OperationsCount" unit="1" resolution="1"/>
  </Object>

  <!-- @description Add Simple function to add two values. -->
  <Function url="Add">
    <In>
      <!-- @param SummandA The first summand -->
      <Absolute name="SummandA" unit="1"/>
      <!-- @param SummandB The second summand -->
      <Absolute name="SummandB" unit="1"/>
    </In>
    <Out>
      <!-- @param Sum The resulting sum (summand a + summand b) -->
      <Absolute name="Sum" unit="1"/>
      <!-- @param Result Result status of the operation -->
      <Enumeration name="Result">
        <!--  @enum ok The operation was successful -->
        <Member id="ok"/>
        <!--  @enum error An error has occurred during the operation -->
        <Member id="error"/>
      </Enumeration>
    </Out>
  </Function>

  <!-- @description Div Simple function to divide two values. -->
  <Function url="Div">
    <In>
      <!-- @param Divident The divident -->
      <Absolute name="Divident" unit="1"/>
      <!-- @param Divisor The divisor -->
      <Absolute name="Divisor" unit="1"/>
    </In>
    <Out>
      <!-- @param Quotient The division result (i.e. the quotient) -->
      <Absolute name="Quotient" unit="1"/>
      <!-- @param Result Result status of the operation -->
      <Enumeration name="Result">
        <!--  @enum ok The operation was successful -->
        <Member id="ok"/>
        <!--  @enum divisionbyzero A division by zero error has happened -->
        <Member id="divisionByZero"/>
        <!--  @enum error An other error has happened -->
        <Member id="error"/>
      </Enumeration>
    </Out>
  </Function>

</Profile>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

5 Transport layer binding

While EXLAP is in principal independent of the underlying transport layer still some definitions have to take place to build real-world systems, since connection setup, start and termination and discovery of communications partners have to be defined.

Implementations using EXLAP are not bound to the recommended bindings and can use different service discovery mechanisms and underlying transport protocols to suite their specific needs.

5.1 General transport connection error handling

5.1.1 Client side view

From view of a correct functioning client, connection errors can occur on two layers within the EXLAP protocol suite:

1) On the level of the EXLAP connection (timeout for requests exceeded, no response for an <*Alive/*> request or no reply for a <*Heartbeat/*> request) due to an EXLAP server that is malfunctioning.
2) An error or congestion on the transport layer.

In both cases the client *should* close the underlying transport channel and try to reconnect to the EXLAP server, if the EXLAP server does not answer within the command timeout of 2-3 retries.

5.1.2 Server side view

An EXLAP server *should* not terminate the connection on his behalf. The server *should* close the EXLAP protocol session only, if the underlying transport connection is closed (i.e. on behalf of the client) or the client requests this action from the server via <*Bye/*> command.

In addition a server *may* close the transport connection to the client on his behalf if the server encounters an internal error or is signalled a system shutdown. In every case, the server *shall* send a <*Bye/*> status message before it closes its connections to the client.

5.2 TCP/IP socket binding

TCP/IP sockets are the most common and a very efficient way for communication between two parties. To make TCP/IP sockets work with EXLAP some definitions, like the port number usage have to be defined beforehand. To determine the port on which an EXLAP service are offered, the EXLAP service discovery may be used. Additionally a service can define a standard port as fall-back when service discovery cannot be used.

5.2.1 EXLAP data representation using TCP/IP sockets

When sockets are used, the EXLAP based XML dialect is directly written to the socket and no special characters or opening and closing sequences are used. How the XML data is formatted (i.e. with indents and/or whitespaces) does not matter as long as the XML generated is valid XML. The data stream on the socket follows exactly the communication examples presented in this document.

5.2.2 Connection termination

Before the socket connection is closed, it is recommended to terminate the logical session via EXLAP <Bye/> command sent by the client or a status <Bye/> from the server side, as described in section 3.5.8.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

5.2.3 Discovery of EXLAP services over IP

The EXLAP transport binding definitions for IP support an automatic service discovery definition. The goal is that an EXLAP service consumer (i.e. client) can look up one or more services without prior configuration.

The following definitions apply for the so-called *service beacon*. There are two roles in the automatic discovery context:

- **Server**: *Should* send a *service beacon* at least every t_{beacon}= 5 seconds, or more frequently, in form of a UDP datagram broadcast on port 28500.
- **Client:** *Should* expect an above described UDP broadcast datagram *service beacon* from a server on port 28500. If a client does not receive a *service beacon* for at least $t_{beacon\text{-}timeout}$ = (t_{beacon} * 2.5) seconds a service is considered "*offline*".

A discovery service beacon content is composed out of the attributes and elements described in the table below.

Attribute and Elements	Use	Description
<ServiceBeacon>		XML element that holds the attribute below.
address	required	An *address* that holds information of how to connect to this service in the form of "socket://[host]:[port]" • Intention is that this discovery process can also be used for future transport schemes (http, datagram, etc.) • The address is included in case devices or API's do not support the access to the destination field of the datagram. Example: "*socket://192.168.0.30:28500*" for a service that is reachable at the IP address 192.168.0.30 at port 28500 or "*socket://localhost:28500*" to connect to the local machine at port 28500.
id	optional	A unique alphanumerical ID (A-Z, a-z, 0-9, '-') that *may* be used to identify a service instance. Since the same server instance can offer its service using different envelopes (i.e. ports at the same IP address), the *id* can be used to detect that multiple service announcements (one for each IP address) belong to the same service instance on a server.
service	optional	An optional service identifier, identifying the service specification implemented by the EXLAP service server. The name of the service is in the scope of the service definition.
version	optional	The version of the "service" specification implemented by the server in the format [major].[minor]. The default value, if this attribute is not present, is "0.0".
status	optional	The availability status of the service. • ***active***: The service is active and ready to accept requests, which is also the default value if this attribute is not present in the *service beacon*. • ***unavailable***: The service is active but does not accept any new requests. Active sessions *may* be served until the service goes "*offline*". • ***offline***: The service is shutting down, active session/connections will be terminated.
description	optional	A free text description for the service, aimed for logging and debugging purposes.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

remoteManagementAddress	optional	This optional attribute tells the client at which address it can reach the EXLAP remote management service for this service instance. If this attribute is not present, no remote management service is available. Note: The remote management service is a user defined service that is not part of this specification. Since the IP binding is only a recommendation and this field is optional – it can be left out in inplementations.

Additional notes: UTF-8 encoding of the data is used. The data representation follows the XML specification. After the closing tag "/>" or *"</ServiceBeacon>*" no additional characters *should* be sent or appended in the UDP packet.

Format of a <ServiceBeacon/> announcement:

```
<ServiceBeacon address="socket://address:port"
         {id="[...]"}
         {service="[...]"}
         {version="major(0).minor(0)"}
         {status="[active|unavailable|offline]"}
         {description="desc"}
         {remoteManagementAddress="socket://address:port"}/>
```

Example: A UDP broadcast to port 28500 that announces a *Math* service.

```
<ServiceBeacon address="socket://192.168.0.30:28500" id="math-test-hcf-1"
service="Math" version="1.2" status="active"/>
```

Note on the service discovery definition:

Since the Java 2 Micro Edition (J2ME) configurations and profiles (MIDP2.0, CLDC 1.1) do not allow for multicast, a simple mechanism was employed. Future versions of this specification *may* define an additional discovery mechanism that leverages the more versatile IP-multicast.

5.3 WebSocket binding

The WebSocket protocol [5] allows bi-directional communication between web applications, commonly executed in web browsers, and web servers on top of the HTTP protocol [6]. In contrast to a HTTP communication, which is solely request-response based, a WebSocket communication is based upons asynchronous and message based communication between the client (i.e. web browser) and server (i.e. web server) part.

5.3.1 WebSocket connection establishment

To enable web applications and browsers to access EXLAP based services the WebSocket transport binding the EXLAP server has to implement the WebSocket protocol as defined in [5]. There is only one extra item defined: The key word for the `Sec-WebSocket-Protocol`, which is `exlap`. If a client does not provide any Sec-WebSocket-Protocol information, the EXLAP WebSocket server shall still accept the incoming WebSocket communication.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Example: EXLAP WebSocket client request

```
GET /exlap/math HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: exlap
Sec-WebSocket-Version: 13
```

Example: EXLAP WebSocket server response

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo
Sec-Websocket-Protocol: exlap
```

5.3.2 EXLAP messages and WebSocket payload data

The mapping of EXLAP messages to WebSocket data frames is very simple. Any complete EXLAP envelope (i.e. `<Req/>`, `<Rsp/>`, `<Dat/>`, `<Status/>`) will be put in a single WebSocket data frame payload (see also section 5.2 – *Base Framing Protocol* – in [5]).

No additional and no special characters or opening and closing sequences are used within the payload. How the XML data is formatted (i.e. with indents and/or whitespaces) does not matter as long as the XML generated is valid XML.

The payload data within the WebSocket data frames follows exactly the communication examples presented in this document.

5.3.3 Connection termination

Before the WebSocket connection is closed it is recommended to terminate the logical session via EXLAP `<Bye/>` command from the client or a status `<Bye/>` from the server side, as described in section 3.5.8.

5.4 Bluetooth binding

The Bluetooth protocol offers an efficient and convenient way to communicate wirelessly between a mobile device and for example – a vehicle. To enable the availability of EXLAP services as Bluetooth services the Serial Port Profile (SPP) [7] is used.

5.4.1 Service representation and discovery

To differentiate between multiple offered services from a device or server, any Bluetooth service has a so-called UUID assigned, a unique 128 bit number, which identifies a service + version for the consumers of services when they are doing a service discovery.

Important to understand in combination with Bluetooth is that a specific service UUID from a service, that leverages the Serial Port Profile, works on top of the SPP. After registering a service to the Bluetooth stack the Bluetooth stack announces the service and takes care of the initial connection and discovery management. When a client then connects to the service (i.e. its Channel/Port as announced), the Bluetooth stack notifies the service implementation.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Example: Bluetooth Service Discovery Record

```
Sequence
        Attribute 0x0000 - ServiceRecordHandle
                UINT32 0x00010000
        Attribute 0x0001 - ServiceClassIDList
                Sequence
                        UUID128 0xa0b15218-a1ab-45c8-8ccf-80b3ea81-1469
                        UUID16 0x1101 - SerialPort
        Attribute 0x0004 - ProtocolDescriptorList
                Sequence
                        Sequence
                                UUID16 0x0100 - L2CAP
                        Sequence
                                UUID16 0x0003 - RFCOMM
                                UINT8 0x06 - Channel/Port
        Attribute 0x0005 - BrowseGroupList
                Sequence
                        UUID16 0x1002 - PublicBrowseGroup
        Attribute 0x0008 - ServiceAvailability
                UINT8 0xff
        Attribute 0x0100
                String Math
        Attribute 0x0101
                String Simple Math service
        Attribute 0x0102
                String EXLAP Bluetooth proxy
```

5.4.2 EXLAP messages and Bluetooth payload data

The mapping of EXLAP messages to a Bluetooth SPP connection is very simple: The EXLAP based XML dialect is directly written to the SPP connection (or socket) and no special characters or opening and closing sequences are used. How the XML data is formatted (i.e. with indents and/or whitespaces) does not matter as long as the XML generated is valid XML.

The data stream on the SPP connections follows exactly the communication examples presented in this document.

5.4.3 Connection termination

Before the Bluetooth SPP connection is closed it is recommended to terminate the logical session via EXLAP `<Bye/>` command from the client or a status `<Bye/>` from the server side, as described in section 3.5.8.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

6 Appendix

6.1 Abbreviations

A list of used abbreviations follows to make sure that all readers are familiar with the technical terms. Note that multiplicity of an acronym or abbreviation can be expressed by a 's' character at the end (e.g. MDs means multiple media devices).

Abbreviation	Description
BT	Bluetooth
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
EXLAP	Extensible lightweight asynchronous protocol
GMT/UTC	Greenwich Mean Time / Universal Time Coordinated
HTTP	Hypertext Transport Protocol
IDL	Interface definition language
IP	Internet Protocol
ISO	International Organization for Standardization
J2ME	Java 2 Micro Edition
JSR	Java Specification Request
MD5	Message-Digest Algorithm 5
OS	Operating System
REST	Representational State Transfer
RFC	Request For Comment
RPC	Remote procedure call
SOAP	Simple Object Access Protocol
SPI	Serial Programmable Interface
SPP	Serial Port Profile (Bluetooth)
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UTF-8	8-Bit Universal character et Transformation Format
UUID	Universal Unique Identifier
WebSocket	The WebSocket protocol
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

6.2 Change history of EXLAP

The version 1.3 of the protocol specification is a further consolidated version with some extensions on top of the 1.2, where some public feedback was integrated.

New and changed in version 1.3:

Most important in the version 1.3 is the capability for so-called alternative data structures (friends of the C language would call them ‘unions’) and the direct support for binary data members. Furthermore the subscribe handling was simplified and the documentation attribution of services got unified.

Command / Issue	What is new/changed?
Added the `<Alternative>` member type.	Support for alternative data structures (to a high degree similar to variant or union types).
Added the `<Binary>` member type.	Support for binary data structures (i.e. array of byte[]) with optional mime type.
No `<Subscribe/>` command before a `<Get/>` required.	The need to subscribe a data objects before it could be polled via the `<Get/>` command was removed.
Removal of the `notification` attribute in `<Subscribe/>` requests.	A subscription of a data object will now always request updates. The former options of `never` and `onceAvailable` have been removed.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

Removal of the *noSubscription* and *contentNotAllowed* error in <*Subscribe/*> requests.	Error states that have been removed due to the changes to the <*Subscribe/*> command.
Move of the <*Description/*> element to the comment section.	The description for functions, types and objects have been modes to the XML comment section (i.e. `<!-- @description name Text -->`) to unify the method of how descriptive comments are embedded in the interface service profile definition.
Introduction of the *implementation* attribute in a service interace <*Profile/*> element.	To denote the differences in a service interface profile a) the core definition and b) an (partial) implementation of a service, the *implementation* attribute was added.

New in version 1.2:

Protocol version 1.2 featured the first public release using a creative commons license. In addition the specification got streamlined and lists plus sub-object structures got introduced.

Command / Issue	What is new/changed?
<*Obj*> and <*List/*> within data object structures <Dat/>. <Response/> etc.	The <*Obj/*> and <*List/*> elements can now contain the *state* and *msg* attributes like all other simple data types.
Removal of deprecated sections	Removal of all deprecated section of the specification in V1.1, the "*typeFilter*" attribute in the <*Dir/*> command and the "*subscriptionLimit*" attribute in the <*ServerData/*> response.
ival attribute in <*Subscribe/*> command.	Clarified that the *ival* attribute *shall* only be applicable for data objects that have a *characteristic* of "*dynamic*".
Removal of *subscription* and *throttleLimit* attributes in <*Object*> response of <*Interface/*> command.	Those attributes are legacy to EXLAP V1.0 and where only of descriptive matter. Consequently also the *subscription* attribute in an <*Dir/*> command response (i.e. <*Url*>) was removed.
Introduction of the *context* attribute in <*Object/*>.	To denote that data object updates can be client *session* specific or *global* the context attribute was introduced.
Removal of *access=none|read* attribute in <*Object*> and introduction of a <Type/> element that is now used for objects that are referenced in lists and objects.	The change was required to denote that a data object is either a pure type definition that can be references from <*ListEntity/*> and <*ObjectEntitiy*> via the *typeRef* attribute or is defined as a subscribable entity.
Removal of the optional *ival* attribute from the <*Absolute/*> and <*Relative/*> member types to the <*Object/*> level using the name *interval*.	A interval applies to the object level and not the member level.
Added *required* attribute to the <*Object/*> and <*Function/*> definition.	Denotes if a function and/or object is mandatory or optional in a service context.
Added *required* attribute to the member definitions used in a service interface definition (i.e. <*Absolute/*>, <*Activity/*>, <*Enumeration/*>*…).*	Denotes that a member in a function or object is optional and can be left out.
Removal of the <*Capabilities/*> command and replacement within the functionality of the	The former <*ServerData/*> response is now included in the <*Protocol/*> response. To get access to the server and service capabilities a client has to explicitly issue a <*Protocol/*> command request.

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

<*Protocol/*> command.	
Changed the discovery beacon envelop element to <*ServiceBeacon/*>.	Renamed the <*...lapService/*> envelope for IP discovery to <*ServiceBeacon/*>. Also renamed the former "*url*" attribute to "*address*" and the "*remoteManagementPort*" attribute to "*remoteManagementAddress*" to better differentiate the *url* of data obejcts and functions from network addresses.
Removed the plain, pretty and padded XML transport envelope definition.	Legacy from EXLAP V1.0.

New in version 1.1:

In contrast to version 1.2, the EXLAP specification version 1.1 was a substantial extension to the concepts of version 1.0. While the version 1.0 was primarily aimed to allow for access of structured vehicle data, the version 1.1 aimed to be a general data exchange protocol and be used as a common communication middleware in a service oriented architecture.

Command / Issue	What is new/changed?
<*Call*>	New command to execute RPC style methods at the side of the server.
Complex data objects	Objects now feature two new member types: Lists (<*List/*>) and (sub) objects (<*Obj/*>). This extension allows the representation of complex data structures while still keeping the whole protocol definition very compact. These types are supported everywhere – in data objects, function <*Call/*>'s, the <*Get/*> command, etc.
Service definitions	EXLAP V1.1 supports the definition, naming and versioning of services. Whereas EXLAP V1.0 was directly focussed on a vehicle specific service with a variable set of data objects - EXLAP V1.1 introduces the concept of multiple services.

6.3 EXLAP protocol xml schema

This XML schema is the validation schema for the EXLAP communication protocol on the wire and the service definitions. For reasons of length and readability descriptive XML annotations are stripped out of this document.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 * EXLAP 1.3 - Core schema definition
 *
 * Authors:
 * Jens Krueger, Volkswagen AG
 * Hans-Christian Fricke, Volkswagen AG
 *
 * This work is licensed under the Creative Commons "Namensnennung-Weitergabe unter gleichen
 * Bedingungen 3.0 Deutschland License": http://creativecommons.org/licenses/by-sa/3.0/de/
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://exlap.de/v1/protocol"
targetNamespace="http://exlap.de/v1/protocol" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:include schemaLocation="definitions.xsd"/>
   <xs:include schemaLocation="object.xsd"/>
   <xs:element name="Exlap">
      <xs:complexType>
         <xs:choice maxOccurs="unbounded">
            <xs:element ref="Req"/>
            <xs:element ref="Status"/>
            <xs:element ref="Rsp"/>
            <xs:element ref="Dat"/>
         </xs:choice>
      </xs:complexType>
   </xs:element>
   <xs:element name="Status">
      <xs:complexType>
         <xs:choice>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
            <xs:element name="Init">
               <xs:complexType/>
            </xs:element>
            <xs:element name="Bye">
               <xs:complexType/>
            </xs:element>
            <xs:element name="Dataloss">
               <xs:complexType/>
            </xs:element>
            <xs:element name="Alive">
               <xs:complexType/>
            </xs:element>
         </xs:choice>
         <xs:attribute name="msg" type="xs:string" use="optional" default=""/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Dat">
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="basicElements">
               <xs:attribute name="url" type="nameType" use="required"/>
               <xs:attribute name="timeStamp" type="xs:dateTime" use="optional" default="1970-
01-01T00:00:00.000Z"/>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>
   </xs:element>
   <xs:element name="Req">
      <xs:complexType>
         <xs:choice>
            <xs:element ref="Protocol"/>
            <xs:element ref="Dir"/>
            <xs:element ref="Subscribe"/>
            <xs:element ref="Unsubscribe"/>
            <xs:element ref="Call"/>
            <xs:element ref="Get"/>
            <xs:element ref="Bye"/>
            <xs:element ref="Alive"/>
            <xs:element ref="Heartbeat"/>
            <xs:element ref="Interface"/>
            <xs:element ref="Authenticate"/>
         </xs:choice>
         <xs:attribute name="id" use="optional" default="0">
            <xs:simpleType>
               <xs:restriction base="xs:nonNegativeInteger">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="999999999"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
      </xs:complexType>
   </xs:element>
   <xs:element name="Interface">
      <xs:complexType>
         <xs:attribute name="url" type="nameType" use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Protocol">
      <xs:complexType>
         <xs:attribute name="version" type="xs:positiveInteger" use="required"/>
         <xs:attribute name="returnCapabilities" type="xs:boolean" use="optional"
default="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Call">
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="basicElements">
               <xs:attribute name="url" type="nameType" use="required"/>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>
   </xs:element>
   <xs:element name="Get">
      <xs:complexType>
         <xs:attribute name="url" type="nameType" use="required"/>
      </xs:complexType>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
   </xs:element>
   <xs:element name="Alive">
      <xs:complexType/>
   </xs:element>
   <xs:element name="Heartbeat">
      <xs:complexType>
         <xs:attribute name="ival" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:nonNegativeInteger">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="60"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
      </xs:complexType>
   </xs:element>
   <xs:element name="Bye">
   </xs:element>
   <xs:element name="Subscribe">
      <xs:complexType>
         <xs:attribute name="url" type="nameType" use="required">
         </xs:attribute>
         <xs:attribute name="ival" use="optional" default="0">
            <xs:simpleType>
               <xs:restriction base="xs:nonNegativeInteger">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="60000"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="content" type="xs:boolean" default="true"/>
         <xs:attribute name="timeStamp" use="optional" default="false">
            <xs:simpleType>
               <xs:restriction base="xs:token">
                  <xs:enumeration value="true"/>
                  <xs:enumeration value="false"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
      </xs:complexType>
   </xs:element>
   <xs:element name="Authenticate">
      <xs:complexType>
         <xs:attribute name="phase" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:token">
                  <xs:enumeration value="challenge"/>
                  <xs:enumeration value="response"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="user" type="xs:string" use="optional" default=""/>
         <xs:attribute name="cnonce" type="xs:base64Binary" use="optional" default=""/>
         <xs:attribute name="digest" type="xs:base64Binary" use="optional" default=""/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Unsubscribe">
      <xs:complexType>
         <xs:attribute name="url" type="nameType" use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Dir">
      <xs:complexType>
         <xs:attribute name="urlPattern" use="optional" default="*">
            <xs:simpleType>
               <xs:restriction base="xs:token">
                  <xs:maxLength value="32"/>
                  <xs:pattern value="\*?[A-Za-z0-9_]*\*?"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="fromEntry" type="xs:positiveInteger" use="optional" default="1"/>
         <xs:attribute name="numOfEntries" type="xs:positiveInteger" use="optional"
default="999999999"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Rsp">
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
    <xs:complexType>
       <xs:choice minOccurs="0">
          <xs:element ref="Type"/>
          <xs:element ref="Function"/>
          <xs:element ref="Object"/>
          <xs:element ref="ObjectData"/>
          <xs:element ref="UrlList"/>
          <xs:element ref="Result"/>
          <xs:element ref="Challenge"/>
          <xs:element ref="Capabilities"/>
       </xs:choice>
       <xs:attribute name="id" use="optional" default="0">
          <xs:simpleType>
             <xs:restriction base="xs:nonNegativeInteger">
                <xs:minInclusive value="0"/>
                <xs:maxInclusive value="999999999"/>
             </xs:restriction>
          </xs:simpleType>
       </xs:attribute>
       <xs:attribute name="status" type="statusType" use="optional" default="ok"/>
       <xs:attribute name="msg" type="xs:string" use="optional" default=""/>
    </xs:complexType>
 </xs:element>
 <xs:element name="Capabilities">
    <xs:complexType>
       <xs:sequence>
          <xs:element ref="Supports"/>
       </xs:sequence>
       <xs:attribute name="service" use="required"/>
       <xs:attribute name="description" use="optional" default=""/>
       <xs:attribute name="version" use="optional" default="1.0">
          <xs:simpleType>
             <xs:restriction base="xs:token">
                <xs:maxLength value="5"/>
                <xs:pattern value="[1-9].[0-9]"/>
             </xs:restriction>
          </xs:simpleType>
       </xs:attribute>
    </xs:complexType>
 </xs:element>
 <xs:element name="Supports">
    <xs:complexType>
       <xs:attribute name="protocol" use="required">
          <xs:simpleType>
             <xs:restriction base="xs:token">
                <xs:maxLength value="5"/>
                <xs:pattern value="[0-9].[0-9]"/>
             </xs:restriction>
          </xs:simpleType>
       </xs:attribute>
       <xs:attribute name="interface" type="xs:boolean" use="optional" default="false"/>
       <xs:attribute name="authenticate" type="xs:boolean" use="optional" default="false"/>
       <xs:attribute name="heartbeat" type="xs:boolean" use="optional" default="false"/>
       <xs:attribute name="datTimeStamp" type="xs:boolean" use="optional" default="false"/>
    </xs:complexType>
 </xs:element>
 <xs:element name="ObjectData">
    <xs:complexType>
       <xs:complexContent>
          <xs:extension base="basicElements">
             <xs:attribute name="url" type="nameType" use="required"/>
          </xs:extension>
       </xs:complexContent>
    </xs:complexType>
 </xs:element>
 <xs:element name="Result">
    <xs:complexType>
       <xs:complexContent>
          <xs:extension base="basicElements">
             <xs:attribute name="url" type="nameType" use="required"/>
          </xs:extension>
       </xs:complexContent>
    </xs:complexType>
 </xs:element>
 <xs:element name="Challenge">
    <xs:complexType>
       <xs:attribute name="nonce" type="xs:string" use="optional" default=""/>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
   </xs:complexType>
</xs:element>
<xs:element name="UrlList">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="Match" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="Match">
   <xs:complexType>
      <xs:attribute name="url" type="nameType" use="required"/>
      <xs:attribute name="type" use="optional" default="object">
         <xs:simpleType>
            <xs:restriction base="xs:token">
               <xs:enumeration value="function"/>
               <xs:enumeration value="object"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="isSubscribed" type="xs:boolean" use="optional" default="false"/>
   </xs:complexType>
</xs:element>
<xs:simpleType name="statusType">
   <xs:restriction base="xs:token">
      <xs:enumeration value="ok"/>
      <xs:enumeration value="error"/>
      <xs:enumeration value="internalError"/>
      <xs:enumeration value="syntaxError"/>
      <xs:enumeration value="protocolNotSupported"/>
      <xs:enumeration value="noMatchingUrl"/>
      <xs:enumeration value="accessViolation"/>
      <xs:enumeration value="subscriptionLimitReached"/>
      <xs:enumeration value="processing"/>
      <xs:enumeration value="invalidParameter"/>
      <xs:enumeration value="notImplemented"/>
      <xs:enumeration value="authenticationFailed"/>
   </xs:restriction>
</xs:simpleType>
<xs:simpleType name="commandSupportedType">
   <xs:restriction base="xs:token">
      <xs:enumeration value="supported"/>
      <xs:enumeration value="notSupported"/>
   </xs:restriction>
</xs:simpleType>
<xs:complexType name="baseValueType">
   <xs:attribute name="name" type="nameType" use="required"/>
   <xs:attribute name="state" use="optional" default="ok">
      <xs:simpleType>
         <xs:restriction base="xs:token">
            <xs:enumeration value="ok"/>
            <xs:enumeration value="nodata"/>
            <xs:enumeration value="error"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="msg" type="xs:string" use="optional" default=""/>
</xs:complexType>
<xs:element name="Abs">
   <xs:complexType>
      <xs:attributeGroup ref="elementBasicAttributeGroup"/>
      <xs:attribute name="val" type="xs:double" use="optional" default="0"/>
   </xs:complexType>
</xs:element>
<xs:element name="Act">
   <xs:complexType>
      <xs:attributeGroup ref="elementBasicAttributeGroup"/>
      <xs:attribute name="val" type="xs:boolean" use="optional" default="false"/>
   </xs:complexType>
</xs:element>
<xs:element name="Alt">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="basicElements">
            <xs:attributeGroup ref="elementBasicAttributeGroup"/>
            <xs:attribute name="type" type="nameType" use="required"/>
         </xs:extension>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
            </xs:complexContent>
         </xs:complexType>
      </xs:element>
      <xs:element name="Bin">
         <xs:complexType>
            <xs:attributeGroup ref="elementBasicAttributeGroup"/>
            <xs:attribute name="val" type="xs:base64Binary" use="optional" default=""/>
         </xs:complexType>
      </xs:element>
      <xs:element name="Enm">
         <xs:complexType>
            <xs:attributeGroup ref="elementBasicAttributeGroup"/>
            <xs:attribute name="val" type="identifierType" use="optional" default="default"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="Txt">
         <xs:complexType>
            <xs:attributeGroup ref="elementBasicAttributeGroup"/>
            <xs:attribute name="val" type="xs:string" use="optional" default=""/>
         </xs:complexType>
      </xs:element>
      <xs:element name="Rel">
         <xs:complexType>
            <xs:attributeGroup ref="elementBasicAttributeGroup"/>
            <xs:attribute name="val" type="xs:double" use="optional" default="0"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="Tim">
         <xs:complexType>
            <xs:attributeGroup ref="elementBasicAttributeGroup"/>
            <xs:attribute name="val" use="optional" default="1970-01-01T00:00:00.000Z">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:pattern value="[0-9]{4}-[0-2][0-9]-[0-3][0-9]T[0-2][0-9](:[0-5][0-
9]){2}(\.[0-9]{1,6})?(Z|(\+|-)[0-9]{2}:[0-9]{2})|[0-9]{4}-[0-2][0-9]-[0-3][0-9]|[0-2][0-
9](:[0-5][0-9]){2}(\.[0-9]{1,6})?(Z|(\+|-)[0-9]{2}:[0-9]{2})"/>
                     <!-- this pattern represents the specifictaion for Time -->
                  </xs:restriction>
               </xs:simpleType>
            </xs:attribute>
         </xs:complexType>
      </xs:element>
      <xs:element name="List">
         <xs:complexType>
            <xs:sequence>
               <xs:element ref="Elem" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attributeGroup ref="elementBasicAttributeGroup"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="Elem">
         <xs:complexType>
            <xs:complexContent>
               <xs:extension base="basicElements"/>
            </xs:complexContent>
         </xs:complexType>
      </xs:element>
      <xs:element name="Obj">
         <xs:complexType>
            <xs:complexContent>
               <xs:extension base="basicElements">
                  <xs:attributeGroup ref="elementBasicAttributeGroup"/>
               </xs:extension>
            </xs:complexContent>
         </xs:complexType>
      </xs:element>
      <xs:complexType name="basicElements">
         <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="Abs"/>
            <xs:element ref="Act"/>
            <xs:element ref="Alt"/>
            <xs:element ref="Bin"/>
            <xs:element ref="Enm"/>
            <xs:element ref="Txt"/>
            <xs:element ref="Rel"/>
            <xs:element ref="Tim"/>
            <xs:element ref="List"/>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
         <xs:element ref="Obj"/>
      </xs:choice>
   </xs:complexType>
   <xs:attributeGroup name="elementBasicAttributeGroup">
      <xs:attribute name="name" type="nameType" use="required"/>
      <xs:attribute name="state" use="optional" default="ok">
         <xs:simpleType>
            <xs:restriction base="xs:token">
               <xs:enumeration value="ok"/>
               <xs:enumeration value="nodata"/>
               <xs:enumeration value="error"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="msg" type="xs:string" use="optional" default=""/>
   </xs:attributeGroup>
</xs:schema>
```

6.4 EXLAP definitions xml schema

This XML schema defines general types and their limitations that are used by EXLAP and services and serve as recommendation that *should* be followed by all EXLAP implementations to guarantee interoperability among them.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 * EXLAP 1.3 - Core schema definition
 *
 * Authors:
 * Jens Krueger, Volkswagen AG
 * Hans-Christian Fricke, Volkswagen AG
 *
 * This work is licensed under the Creative Commons "Namensnennung-Weitergabe unter gleichen
 * Bedingungen 3.0 Deutschland License": http://creativecommons.org/licenses/by-sa/3.0/de/
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://exlap.de/v1/protocol"
targetNamespace="http://exlap.de/v1/protocol" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:simpleType name="identifierType">
      <xs:restriction base="xs:token">
         <xs:minLength value="1"/>
         <xs:maxLength value="32"/>
         <xs:pattern value="[A-Za-z0-9_]+"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="nameType">
      <xs:restriction base="xs:token">
         <xs:minLength value="3"/>
         <xs:maxLength value="32"/>
         <xs:pattern value="[A-Z][A-Za-z0-9_]*"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="portableType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="u8"/>
         <xs:enumeration value="s8"/>
         <xs:enumeration value="u16"/>
         <xs:enumeration value="s16"/>
         <xs:enumeration value="u32"/>
         <xs:enumeration value="s32"/>
         <xs:enumeration value="u64"/>
         <xs:enumeration value="s64"/>
         <xs:enumeration value="string"/>
         <xs:enumeration value="float"/>
         <xs:enumeration value="double"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

6.5 EXLAP profile xml schema

This XML schema is the base for the definition of EXLAP service interface profiles.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 * EXLAP 1.3 - Core schema definition
 *
 * Authors:
 * Jens Krueger, Volkswagen AG
 * Hans-Christian Fricke, Volkswagen AG
 *
 * This work is licensed under the Creative Commons "Namensnennung-Weitergabe unter gleichen
 * Bedingungen 3.0 Deutschland License": http://creativecommons.org/licenses/by-sa/3.0/de/
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://exlap.de/v1/protocol"
targetNamespace="http://exlap.de/v1/protocol" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:include schemaLocation="definitions.xsd"/>
   <xs:include schemaLocation="object.xsd"/>
   <xs:element name="Profile">
      <xs:complexType>
         <xs:choice maxOccurs="unbounded">
            <xs:element ref="About" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="Object" maxOccurs="unbounded"/>
            <xs:element ref="Function" maxOccurs="unbounded"/>
            <xs:element ref="Type" maxOccurs="unbounded"/>
         </xs:choice>
         <xs:attribute name="name" type="nameType" use="required"/>
         <xs:attribute name="version" use="optional" default="0.0">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:pattern value="[0-9]*\.[0-9]*"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="implementation" use="optional" default=""/>
      </xs:complexType>
      <xs:key name="objectUrlUniqueness">
         <xs:selector xpath="Object|Function|Type"/>
         <xs:field xpath="@url"/>
      </xs:key>
   </xs:element>
   <xs:element name="About"/>
</xs:schema>
```

6.6 EXLAP object xml schema

This XML schema defines the meta-data types that are used in defined a service interface profile like <*Absolute/*>, <*Relative/*>, etc.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 * EXLAP 1.3 - Core schema definition
 *
 * Authors:
 * Jens Krueger, Volkswagen AG
 * Hans-Christian Fricke, Volkswagen AG
 *
 * This work is licensed under the Creative Commons "Namensnennung-Weitergabe unter gleichen
 * Bedingungen 3.0 Deutschland License": http://creativecommons.org/licenses/by-sa/3.0/de/
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://exlap.de/v1/protocol"
targetNamespace="http://exlap.de/v1/protocol" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:include schemaLocation="definitions.xsd"/>
   <xs:element name="Object">
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="basicInterfaceElementAttributes">
               <xs:group ref="basicMembers" minOccurs="1" maxOccurs="unbounded"/>
               <xs:attribute name="context" type="contextType" use="optional"
default="global"/>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
            <xs:attribute name="characteristic" type="characteristicType" use="required"/>
            <xs:attribute name="interval" type="xs:nonNegativeInteger" use="optional"
default="0"/>
            <xs:attribute name="required" use="optional" default="false"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:unique name="objectNameUniqueness">
      <xs:selector xpath="*"/>
      <xs:field xpath="@name"/>
   </xs:unique>
</xs:element>
<xs:element name="Function">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="basicInterfaceElementAttributes">
            <xs:sequence>
               <xs:element ref="In"/>
               <xs:element ref="Out"/>
            </xs:sequence>
            <xs:attribute name="required" use="optional" default="false"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:element>
<xs:element name="Type">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="basicInterfaceElementAttributes">
            <xs:group ref="basicMembers" minOccurs="1" maxOccurs="unbounded"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:element>
<xs:element name="In">
   <xs:complexType>
      <xs:group ref="basicMembers" minOccurs="0" maxOccurs="unbounded"/>
   </xs:complexType>
   <xs:unique name="FunctionInNameUniqueness">
      <xs:selector xpath="*"/>
      <xs:field xpath="@name"/>
   </xs:unique>
</xs:element>
<xs:element name="Out">
   <xs:complexType>
      <xs:group ref="basicMembers" minOccurs="0" maxOccurs="unbounded"/>
   </xs:complexType>
   <xs:unique name="FunctionOutNameUniqueness">
      <xs:selector xpath="*"/>
      <xs:field xpath="@name"/>
   </xs:unique>
</xs:element>
<xs:element name="Absolute">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="basicMemberAttributes">
            <xs:attribute name="unit" type="xs:string" use="required"/>
            <xs:attribute name="min" type="xs:double" use="optional" default="-INF"/>
            <xs:attribute name="max" type="xs:double" use="optional" default="INF"/>
            <xs:attribute name="resolution" type="xs:double" use="optional" default="0"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:element>
<xs:element name="Activity">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="basicMemberAttributes"/>
      </xs:complexContent>
   </xs:complexType>
</xs:element>
<xs:element name="Alternative">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="basicMemberAttributes">
            <xs:sequence>
               <xs:element ref="Choice"/>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
            <xs:element ref="Choice" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
</xs:element>
<xs:element name="Choice">
<xs:complexType>
   <xs:attribute name="typeRef" type="nameType" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="Binary">
<xs:complexType>
   <xs:complexContent>
      <xs:extension base="basicMemberAttributes">
         <xs:attribute name="contentType" type="xs:string" use="required"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
</xs:element>
<xs:element name="Enumeration">
<xs:complexType>
   <xs:complexContent>
      <xs:extension base="basicMemberAttributes">
         <xs:sequence>
            <xs:element ref="Member"/>
            <xs:element ref="Member" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:unique name="memberIdUniqueness">
   <xs:selector xpath="Member"/>
   <xs:field xpath="@id"/>
</xs:unique>
</xs:element>
<xs:element name="Member">
<xs:complexType>
   <xs:attribute name="id" type="identifierType" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="Relative">
<xs:complexType>
   <xs:complexContent>
      <xs:extension base="basicMemberAttributes">
         <xs:attribute name="min" type="xs:double" use="required"/>
         <xs:attribute name="max" type="xs:double" use="required"/>
         <xs:attribute name="minLabel" type="identifierType" use="required"/>
         <xs:attribute name="maxLabel" type="identifierType" use="required"/>
         <xs:attribute name="resolution" type="xs:double" use="optional" default="0"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
</xs:element>
<xs:element name="Text">
<xs:complexType>
   <xs:complexContent>
      <xs:extension base="basicMemberAttributes">
         <xs:attribute name="regExp" type="xs:string" use="optional" default=".*"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
</xs:element>
<xs:element name="Time">
<xs:complexType>
   <xs:complexContent>
      <xs:extension base="basicMemberAttributes">
         <xs:attribute name="isLocalTime" type="xs:boolean" use="required"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
</xs:element>
<xs:element name="ListEntity">
<xs:complexType>
   <xs:complexContent>
      <xs:extension base="basicMemberAttributes">
         <xs:attribute name="typeRef" type="nameType" use="required"/>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.

```
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>
   </xs:element>
   <xs:element name="ObjectEntity">
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="basicMemberAttributes">
               <xs:attribute name="typeRef" type="nameType" use="required"/>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>
   </xs:element>
   <xs:simpleType name="contextType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="global"/>
         <xs:enumeration value="session"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="characteristicType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="static"/>
         <xs:enumeration value="dynamic"/>
         <xs:enumeration value="event"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:group name="basicMembers">
      <xs:choice>
         <xs:element ref="Absolute"/>
         <xs:element ref="Activity"/>
         <xs:element ref="Alternative"/>
         <xs:element ref="Binary"/>
         <xs:element ref="Enumeration"/>
         <xs:element ref="ListEntity"/>
         <xs:element ref="ObjectEntity"/>
         <xs:element ref="Relative"/>
         <xs:element ref="Text"/>
         <xs:element ref="Time"/>
      </xs:choice>
   </xs:group>
   <xs:complexType name="basicInterfaceElementAttributes">
      <xs:attribute name="url" type="nameType" use="required"/>
   </xs:complexType>
   <xs:complexType name="basicMemberAttributes">
      <xs:attribute name="name" type="nameType" use="required"/>
      <xs:attribute name="required" type="xs:boolean" use="optional" default="true"/>
   </xs:complexType>
</xs:schema>
```

This work is licensed under the Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Deutschland License.