Open Communication Interface for Road Traffic Control Systems

Offene Schnittstellen für die Straßenverkehrstechnik

OCIT-C Center to Center

Transport log

OCIT-C_Protocol_V2.0_R1_D1

OCIT Developer Group (ODG)&Partner

OCIT-C Center to Center

Transport log

Document: OCIT-C_Protokoll_V2.0_R1_D1

Issued by: ODG & Partner

Contact: www.ocit.org

Contents

Document history

Version State	Date	Distribution List	Comments
V1.1_R1	**30.10.2014**	**PUBLIC**	**Version 1.1 Issue 1**
V2.0_R1_D1	2016-04-29	ODG internal	2.3.3: Text correction 2.3.4: New
	2016-09-21		2.2.1: inquireAll Status of the objects in the past (text added)
	2016-09-29		2.2.1 inquireAll, text correction 2.3.5 timestamp, text changed 2.3.8.1 Images 7, 8, 9 corrected 2.5.1 Writing fault corrections
V2.0_A01	**2018-05-31**	**PUBLIC**	**For OCIT-C V2.0 ODG Homepage**

1 Introduction

OCIT-C stands for Open Communication Interface for Road Traffic Control Systems - Center to Center. OCIT-C covers the functions for communicating between the central traffic control and traffic guidance systems:

- Traffic control centers and traffic management centers (urban, regional, interregional)
- Traffic engineer work place with traffic control centers
- Parking guidance systems, parking facility systems
- Roadworks management systems
- Local internet users (city information online)

The definition and maintenance of the OCIT-C interface is carried out by the ODG and their partners .

OCIT-C is a standard that supplements OCIT-O perfectly. Using OCIT-C and OCIT-O for the communication from central systems to field devices covers all requirements from traffic control through to primary traffic management.

OCIT-C is therefore geared towards practical requirements. With its low implementation costs, its use is also suitable for solutions with small budgets.

The characteristic properties of OCIT-C are:

- Exchange protocol with a simple request-response communication pattern (direct querying of data) based on the SOAP standard.
- Definition of a comprehensive data model in the process data area containing all partial sections of the traffic control and traffic guidance.
- System integration and desired adaptations are governed in advance by project planning.
- Conformity tests for the protocol are carried out in a test environment provided at www.ocit.org. Tests of entire implementations (protocol and data contents) are carried out on a project-specific basis.
- Expansions to the DATEX II components are possible based on your project requirements.

The communication interface should be implemented the same way in all central units. For this the SOAP protocol is used as the higher-level communication interface, via which all communication takes place. The version described here is designated as the OCIT-C protocol.

This interface is open and can be used in various systems, predominantly in the road traffic technology sector. The aim of this document is to describe the OCIT-C protocol and how its used. The aim of this document is not to describe the data structures of the data to be transmitted. These are described in the document "OCIT-C data".

2 Protocol

All communication via the interface is processed using the SOAP protocol.

This section describes the use of the protocol for the OCIT-C interface. This configuration must be installed on all clients and servers.

The exact description of the data model at the root of this protocol, as well as the elementary description of the attributes and elements takes place completely within the individual schema definitions in the form of XML schema definitions (XSD). These can be machine-processed and readable as text. The schema definitions were written up in English and will not be translated.

2.1 SOAP transmission protocol

This chapter describes the SOAP interface.

2.1.1 Technology

The data to be transferred are encoded as XML. This has the following advantages:

- usual log for all areas,
- independence from type of data,
- platform-independent,
- easy to upgrade.

SOAP based on http is used as the transmission method. SOAP also uses XML to structure its data.

The protocol includes simple commands like "get" and "delete".

2.1.2 Requirements for the protocol

- The protocol is a server-client protocol.
- Data is presented as XML at the output interface.
- Data is accepted as XML at the input interface.
- Commands are embedded in XML.
- Objects are identified by external identifiers.
- Different object types must not be requested at the same time (in a request).
- The protocol has no status within the server. The server does not know anything about the client.

2.1.3 Security

The server contains a list of user names and the passwords associated with them as well as the operations and access to clients permitted to the user.

2.1.4 Required bandwidth

The required bandwidth depends on the number of clients, the object types and objects in the system. Therefore, no statements can be made about bandwidth here. A local area network (LAN) between the central applications will provide sufficient transmission capacity.

2.2 Protocol functions

The protocol allows the reading and configuration of data. Moreover, it is possible to evaluate objects with regard to usability and structure them dynamically during runtime. Every command consists of a "request" and a "response".

With a request, an XML structure is sent from the client to the server. The "result" (also known as the "response structure") will be sent back from the server to the client.

Available methods:

Request	**Response**	**Function**
put	putResponse	Configure objects
get	getResponse	Query modified data since the last query
inquireAll	inquireAllResponse	Query all objects of an object type
delete	deleteResponse	Delete dynamic data
getContentInfo	getContentInfo-Response	Query object contents
wait4Get	wait4GetResponse	Query modified data since the last query (like "get") with the difference that the response is delayed until data are available.

2.2.1 Reading data with the client

All protocol functions for reading data contain a parameter (filter) that marks the objects that should be read.

If the filter is empty, all objects are sent back in the corresponding response. So that the client can re-synchronize in the event of an interruption, the starting information from the previous response of a read operation are transmitted too.

The server provides all readable data from the available objects at its external interface. The available object types can be queried by the client with the command getContentInfo. These can be read by the client with "get" or "inquireAll" (if read access is allowed).

The difference between inquireAll and get is:

- inquireAll (re-synchronization function) delivers all the objects of the queried object type with their last status and the content of the objects. inquireAll must be used for any synchronization (e.g. server or client restart).
- get (used here to read changes) delivers all the changes made to the content of the queried object type since the last query. This mechanism is described in detail in section 2.3.1, data buffering and positions handling.

The following flow chart, figure 1, shows how data are periodically requested from the server using the protocol functions "inquireAll" and "get".

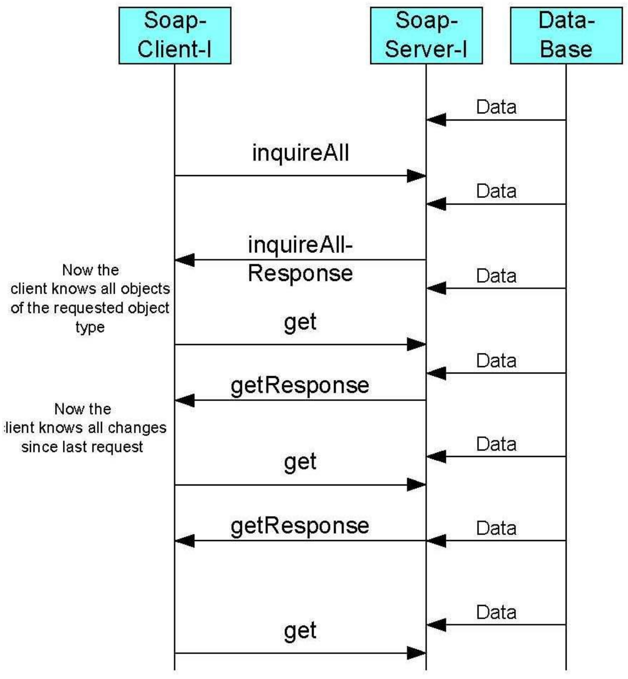


Figure 1*: Usual order in which data are read by the SOAPserverInterface*

2.2.2 Sending data to the server

It is possible to send data to the server. The protocol function "put" is used for this. The behavior of the server depends on the object type. In case of an unknown object in the "put" command, either the object is created or a fault is sent back. To delete objects, the command "delete" can be used.

Relevant data are sent to the server with "put" to configure the interface. The server accepts them or rejects all non-configurable objects and lists them in the putResult-list.

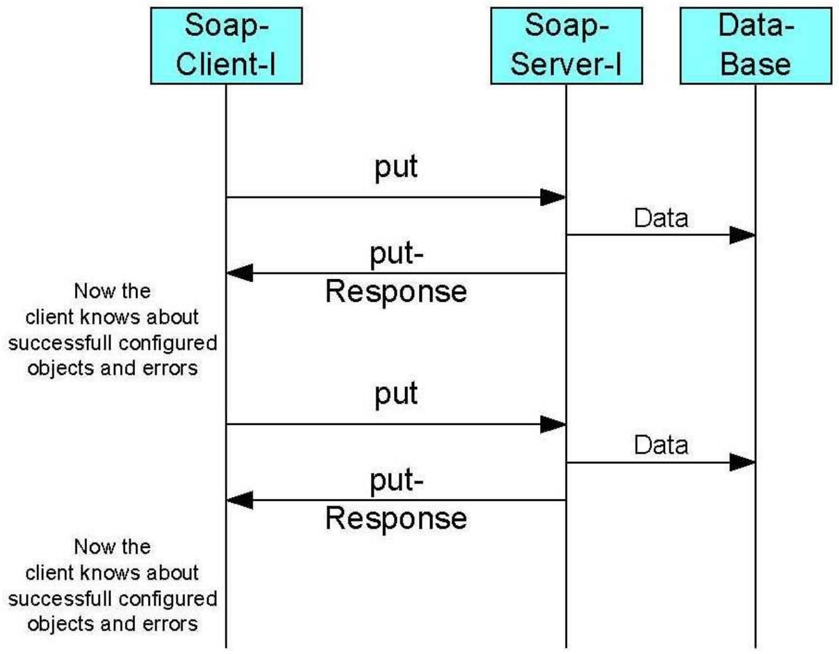
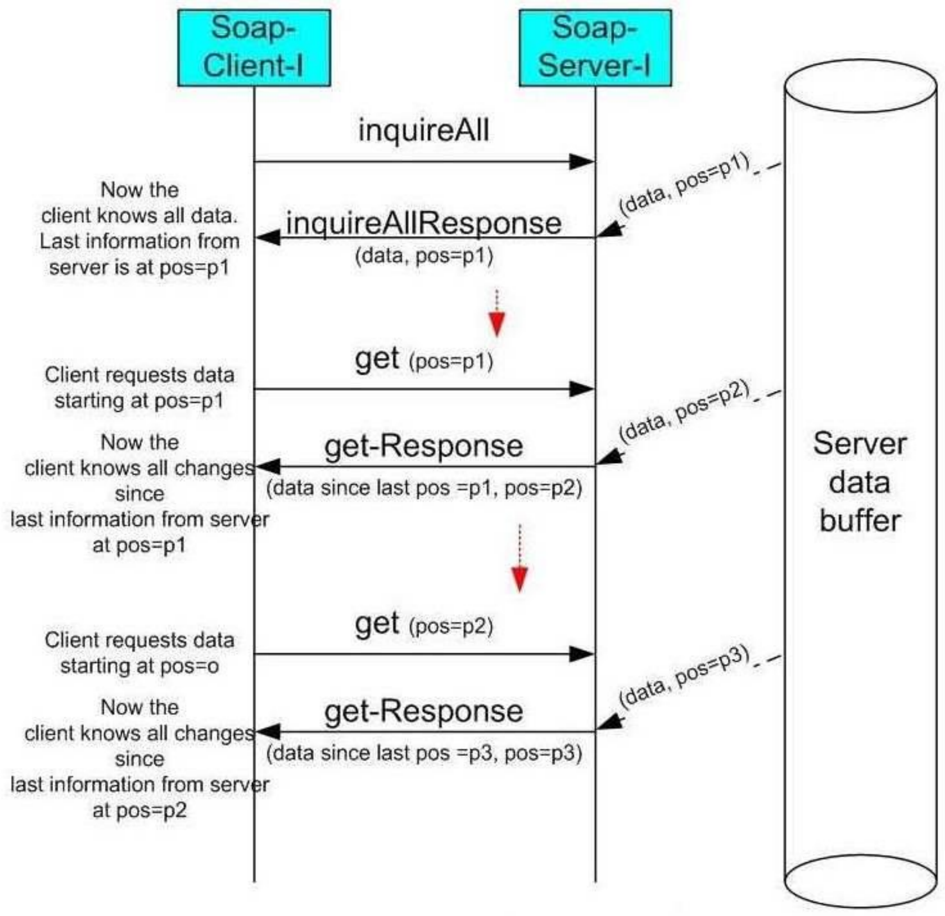

Soap-Client-I
Soap-Server-I
Data-Base
put
Data
put-Response
Now the client knows about successfull configured objects and errors
put
Data
put-Response
Now the client knows about successfull configured objects and errors

Figure 2: Usual order in which data are written or configured on the SOAPserverInterface

2.3 Sequence control

The protocol is connectionless. For sequence control it is enough to wait for the response to a query. This is governed by the http protocol. Additional sequence control is not necessary. Suitable sequences are described in the sections below.

2.3.1 Data buffering and position handling



Soap-Client-I
Soap-Server-I
inquireAll
Now the client knows all data. Last information from server is at pos=p1
inquireAllResponse
(data, pos=p1)
(data, pos=p1)
Client requests data starting at pos=p1
get (pos=p1)
(data, pos=p2)
Now the client knows all changes since last information from server at pos=p1
get-Response
(data since last pos =p1, pos=p2)
Server data buffer
Client requests data starting at pos=o
get (pos=p2)
(data, pos=p3)
Now the client knows all changes since last information from server at pos=p2
get-Response
(data since last pos =p3, pos=p3)

Figure 3: Data buffering on the server and position handling on the client

The server stores the data requested by the client from the server's database into a ring buffer. After carrying out an inquireAll, the client knows the last status of the database. Besides the inquireAll response, the client also receives the last position number and displays the last information from the server (pointer to the last received data in the data buffer). Using this position number, the client generates the next request (get). With the position number, the server knows which data must be made available to deliver all changes that came up. A received response includes the desired data and a new position number, which is used for the following query.

If a client wants to query more than one object type, multiple queries must be used.

Each query line must carry out its own position handling to receive delta values regarding the object type. This also applies to the use of different filter lists and the use of parameters with the optional XML element "get→data".

2.3.2 Excessively long transaction time

Normally, a response contains the data requested. If the server needs more time than permitted (more than 60s), an empty response is sent back to the client. An error code will show that transaction time has been exceeded. Nevertheless, the server continues to get other requested data from its database (or from the archive). The client repeats its query after a certain amount of time. After this time, the server should be able to deliver the data. If the server still cannot deliver, an error code with be displayed again.

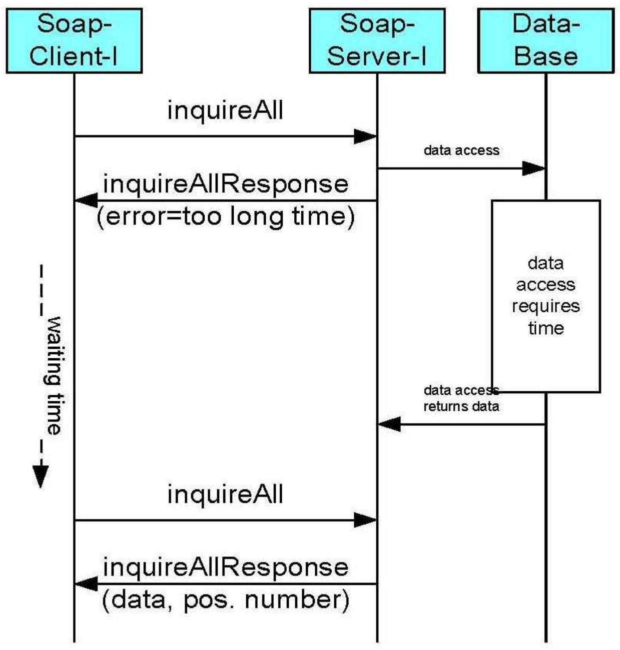

Soap-Client-I
Soap-Server-I
Data-Base
inquireAll
data access
inquireAllResponse (error=too long time)
data access requires time
waiting time
data access returns data
inquireAll
inquireAllResponse (data, pos. number)

Figure 4: Handling long transition times (similar order also applies to "get")

2.3.3 Excessively long queries

The server returns an error code if the content of a response exceeds a certain threshold. This can happen if

- too many historical data are requested (time range too large), or
- too many objects are requested.

The client must accordingly reduce its request by decreasing the number of objects or the query timeframe.

2.3.4 Too many changes

If too many changes have been made since the last query, the query will be satisfied to the extent possible by either returning not all the objects or not the entire timeframe.

The incomplete return will be indicated by the errorCode (in the protocol.xsd, error-Code: missingDatasets)

2.3.5 Handling queries of historical data

- The command "get" is used if requesting access to the historical data.

- The element "storetime" is used to define the start time for the saved historical data.
- The element "endStore" is used to define the end time for the saved historical data.
- The response from the server contains the status changes (no initial values) from "storetime" to "endStore" as long as not too long a query interval was defined.
- If too long a query interval was defined, the query will be satisfied to the extent possible by either returning not all the objects or not the entire timeframe. The in-complete return will be indicated by the errorCode (in the protocol.xsd, errorCode: missingDatasets)
- If "storetime" is the same as "endStore", the status at this time is sent. The value in this case contains, in the form of a timestamp, either exactly the original timestamp of the value or, if this is not available, the initial timestamp will not be listed (timestamp is an optional element in protocol.xsd). If the initial value cannot be returned or returning it makes no sense (e.g. for object type "operatingMes-sages"), then no value is returned.

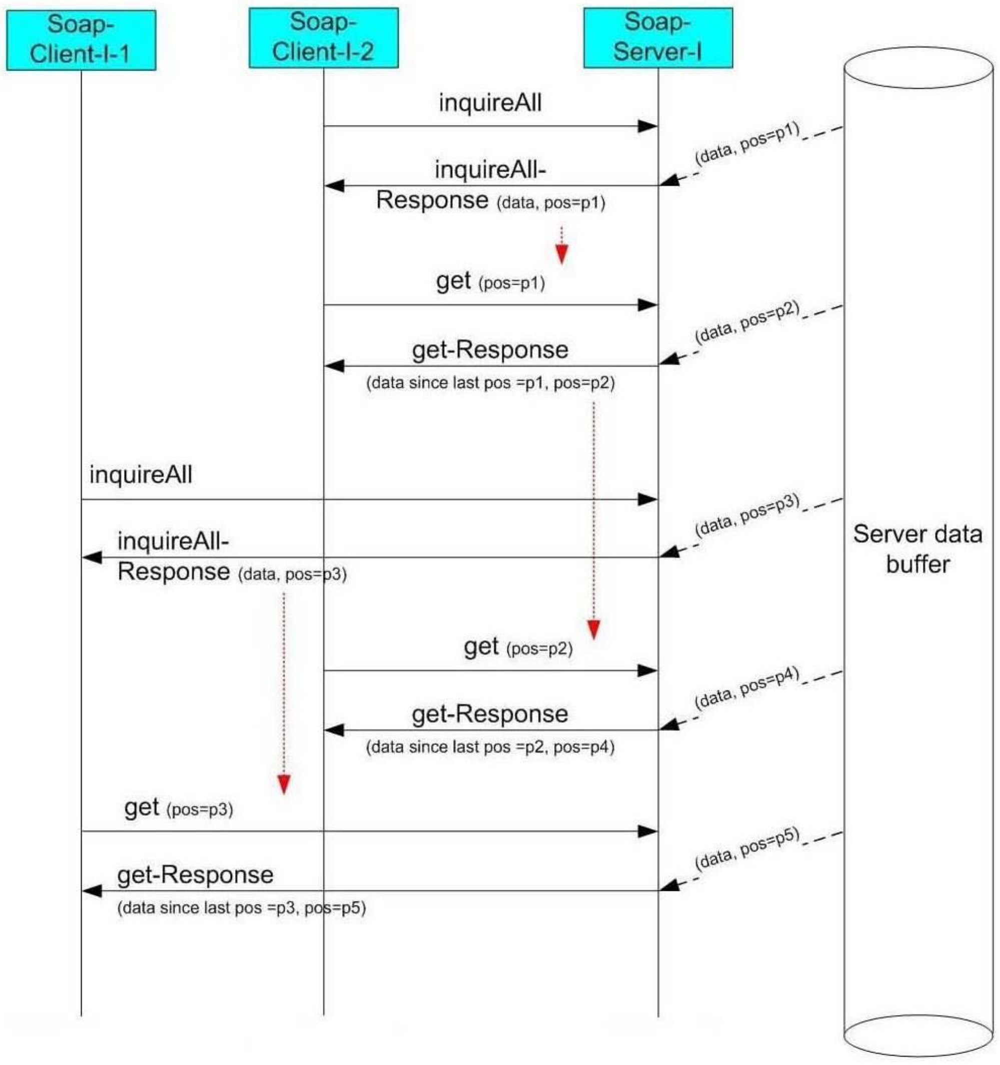
Recommended:

- select as short a range of time as possible
- use filters and thereby reduce the amount of objects to be queried.

2.3.6 Multi-client capability

The server is connectionless. Due to the position number, it is not necessary to build up knowledge about the client.

In implementing OCIT-C it must be ensured that the libraries used allow multiple-cli-ent access for lower-layer communication.



Soap-Client-I-1
Soap-Client-I-2
Soap-Server-I
inquireAll
(data, pos=p1)
inquireAll-Response (data, pos=p1)
get (pos=p1)
(data, pos=p2)
get-Response (data since last pos =p1, pos=p2)
inquireAll
(data, pos=p3)
inquireAll-Response (data, pos=p3)
Server data buffer
get (pos=p2)
(data, pos=p4)
get-Response (data since last pos =p2, pos=p4)
get (pos=p3)
(data, pos=p5)
get-Response (data since last pos =p3, pos=p5)

Figure 5: Multi-client capability

2.3.7 Resynchronizing

There are various reasons for resynchronization:

- Restarting the client
  - The client recognizes this and therefore resynchronizes with "inquireAll".
  - The server may identify the client restart using a watchdog that can optionally be put in place.
- Restarting the server
  - The client may identify that the server is not reachable via "socket timeout".
  - The server reacts to any possible protocol function (inquireAll, get or put), including "lastStart", which displays when the server was started up.

- The client must resynchronize with "inquireAll" if "lastStart" is different from the previous one.

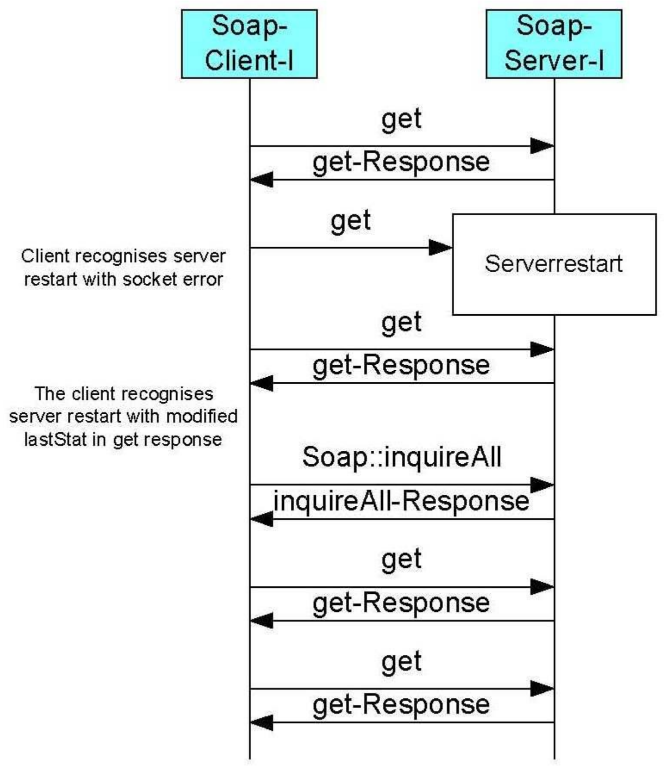


Figure 6: Server restart

2.3.8 Bidirectional communication

The following cases require bidirectional communication:

- TSS control or sign control:
    - Switching command is transmitted from control center to controller server
    - Switching state is transmitted asynchronously from controller server to control center

- CCTV
    - PTZ (pan/tilt/zoom) command is transmitted from control center to con-troller server
    - - Current state is transmitted asynchronously from controller server to control center

The following sections describe possible configurations. "Sign control" is used as an example.

2.3.8.1 Bidirectional communication with client and server pair

Switching a sign (process order in case of proper functioning)

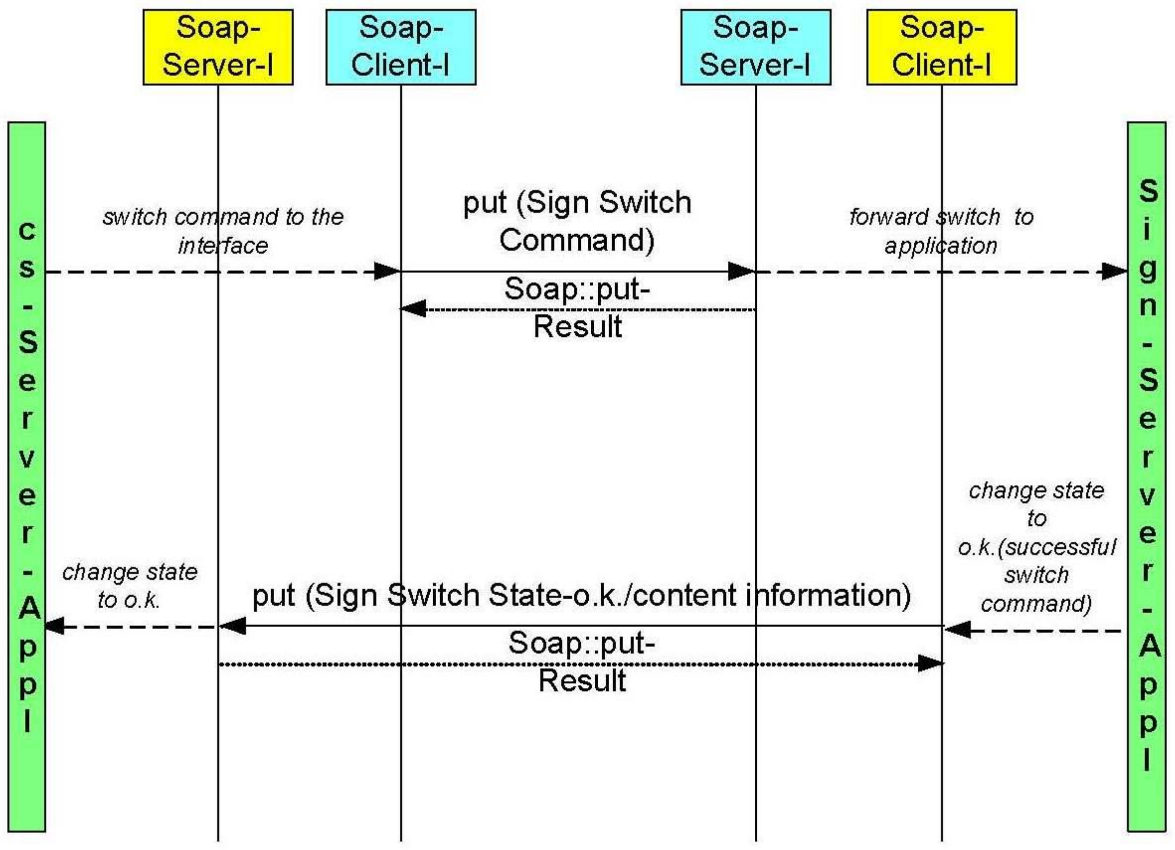

Soap-Server-I
Soap-Client-I
Soap-Server-I
Soap-Client-I
cs-Server-Appl
Sign-Server-Appl
switch command to the interface
put (Sign Switch Command)
forward switch to application
Soap::put-Result
change state to o.k.(successful switch command)
change state to o.k.
put (Sign Switch State-o.k./content information)
Soap::put-Result

Figure 7: Switching a sign (process order in case of proper functioning*)*

Switching a sign (process order in case of faults)

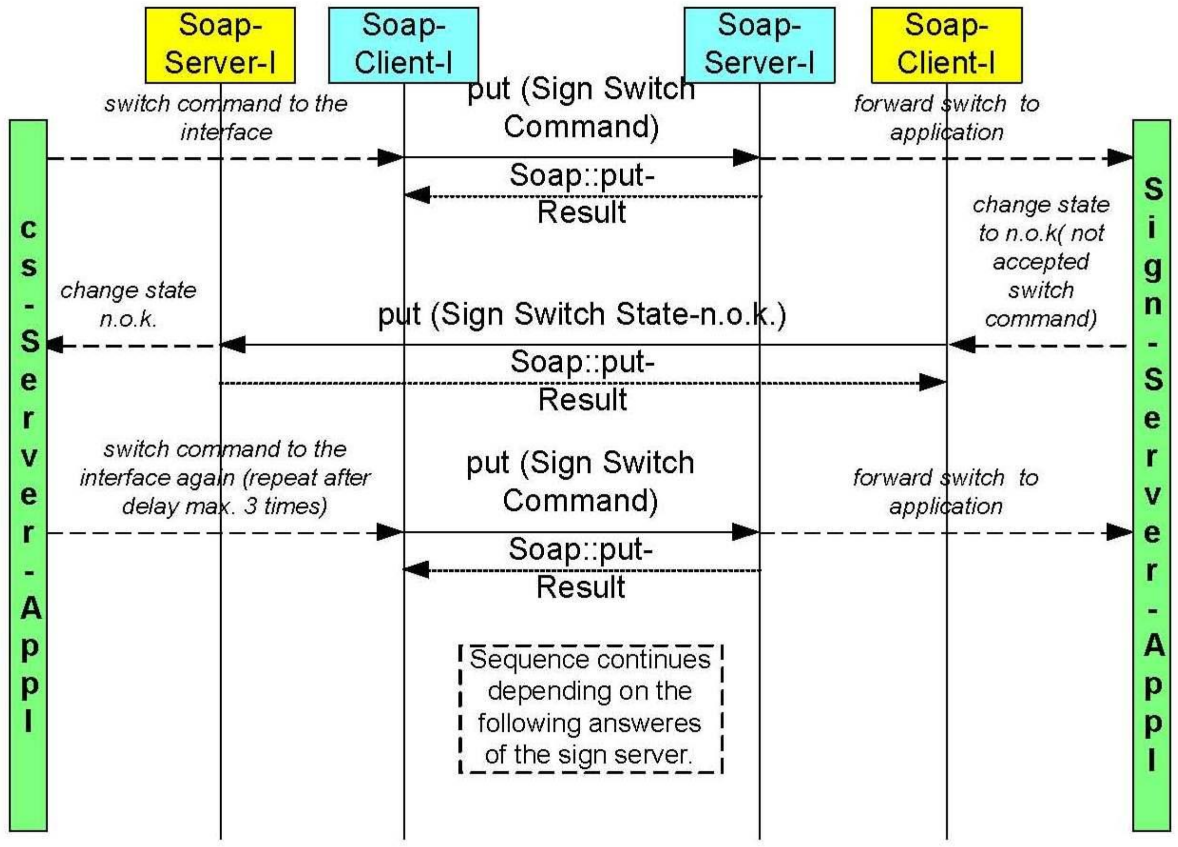

Soap-Server-I
Soap-Client-I
Soap-Server-I
Soap-Client-I
cs-Server-Appl
Sign-Server-Appl
switch command to the interface
put (Sign Switch Command)
forward switch to application
Soap::put-Result
change state to n.o.k(not accepted switch command)
change state n.o.k.
put (Sign Switch State-n.o.k.)
Soap::put-Result
switch command to the interface again (repeat after delay max. 3 times)
put (Sign Switch Command)
forward switch to application
Soap::put-Result
Sequence continues depending on the following answeres of the sign server.

Figure 8: Switching a sign (process order in case of faults)

In case of a missing response, the switching commands are repeated after a configu-rable delay. This applies regardless of where the fault arises (no status transition oc-curs to status "busy" or "ok"). Repeating is stopped after three unsuccessful tries. Then, the current status on the side of the central system changes to "nok".

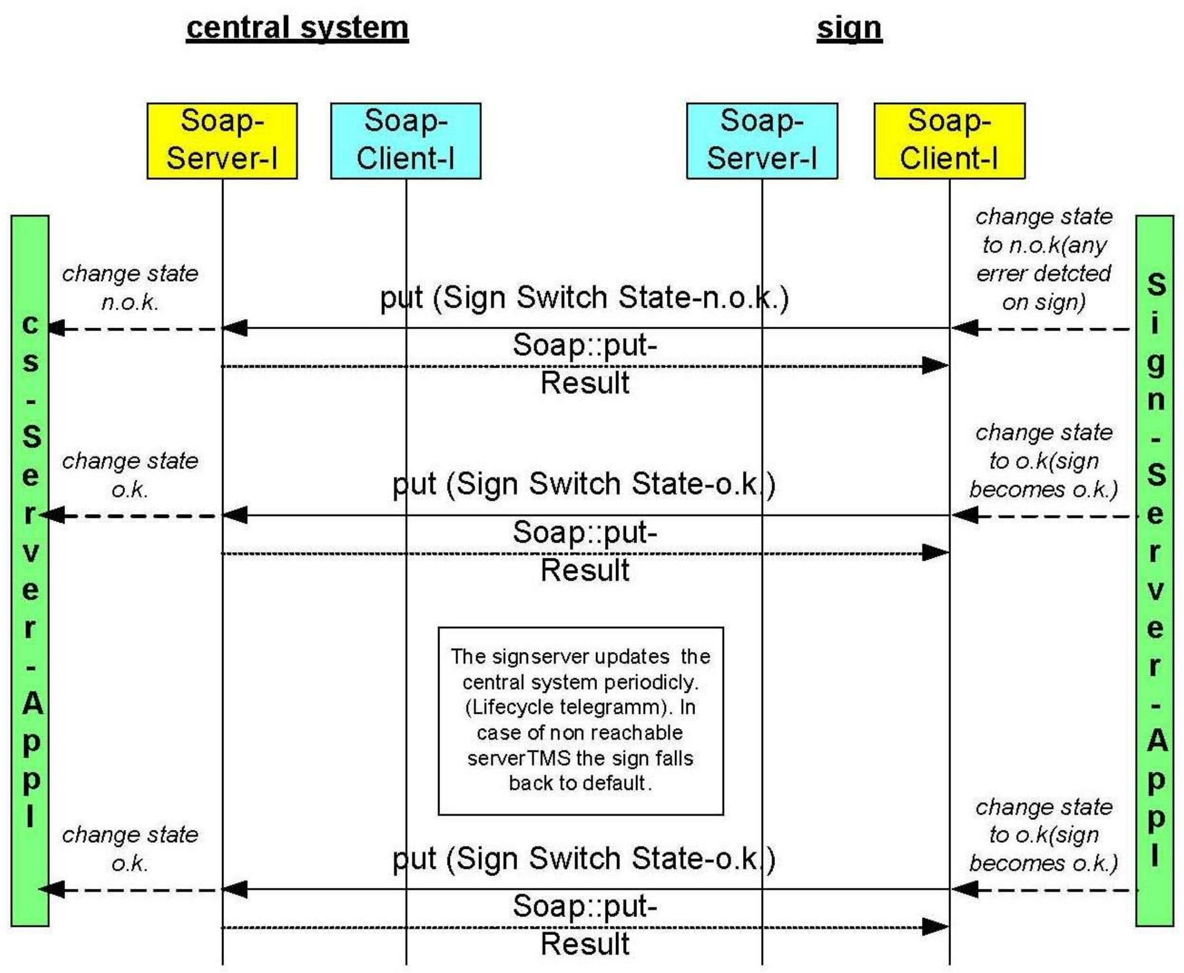

central system
sign
Soap-Server-I
Soap-Client-I
Soap-Server-I
Soap-Client-I
cs-Server-Appl
Sign-Server-Appl
change state n.o.k.
change state to n.o.k(any errer detcted on sign)
put (Sign Switch State-n.o.k.)
Soap::put-Result
change state o.k.
change state to o.k(sign becomes o.k.)
put (Sign Switch State-o.k.)
Soap::put-Result
The signserver updates the central system periodicly. (Lifecycle telegramm). In case of non reachable serverTMS the sign falls back to default.
change state o.k.
change state to o.k(sign becomes o.k.)
put (Sign Switch State-o.k.)
Soap::put-Result

Figure 9: Switching state

The sign server updates the central system periodically even if the switching state does not change. As a result, the central system receives notifications:

- if a sign connected to the sign server changes its status
- if a sign changes its content
- if the sign server is unreachable (the max. wait time is specified by the central system)

If the central system is unreachable, the sign server sets the signs to a predefined display status.

2.3.8.2 Bidirectional communication with regular status inquiry (polling)

The evaluation of states takes place as described in section 2.3.8.1.

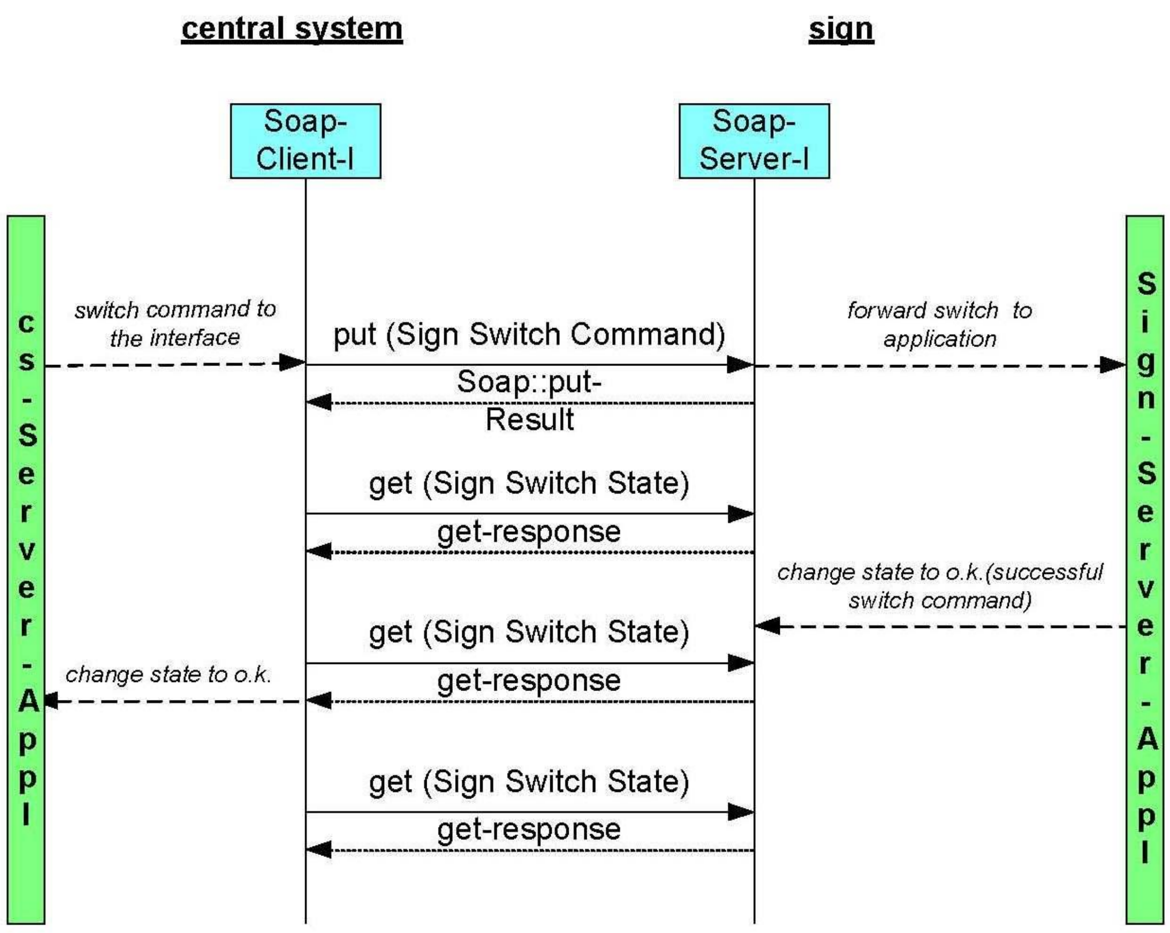


Figure 10: Polling

2.3.9 Avoiding sampling delays

To avoid undesirably long sampling delays due to cyclical queries from the client to the server, a new function is introduced in the protocol. This function works according to the following principle:

- Structure like function "get"
  (i.e. same parameters, just different function name)

  Name of the function "wait4Get"

- The new function "wait4Get" works like the function "get" with the difference that in case of no data present on the server (i.e. the return list is empty), the response from "wait4Get" is delayed until either a timeout occurs or data are available on the server.

- The client would, in this case, call the "wait4Get" function again immediately after receiving a response in order to indirectly keep the return channel permanently open.

- To avoid multiple queries within one second, the server can take protective measures. A potential protective measure would be to permit only one response per second and return the values in the response collected in this second.

- The server can restrict the number of clients or object types / objects that can be queried this way.
- To avoid too many open sockets, the function "wait4Get" makes it possible to query various object types at the same time. This was accordingly taken into consideration in the data structures of the function.

The following sequence diagram illustrates the process (simplified for the query of an object type:

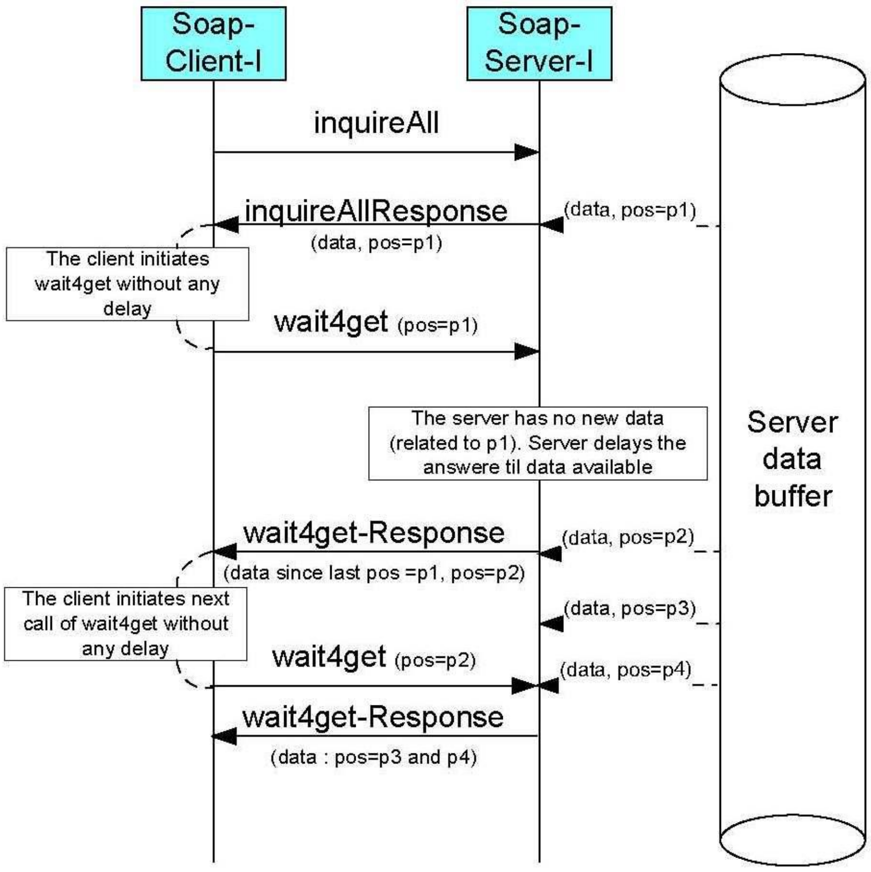


Figure 11: Communication layers - client and server

2.4 OSI layers

The server and client have layers subdivided according to the OSI model into different sections with different functions.

The lowest protocol layer is the http protocol, which is responsible for data transfer in the network.

Above that is the SOAP protocol in the form of a client or a server.

The protocol manager has the task of providing all commands including the required data buffers for the functioning of the server.

The application layer represents the connection to the database.

The following figure shows the layer model.

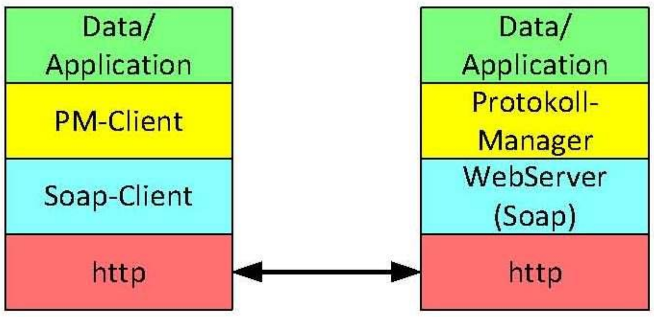
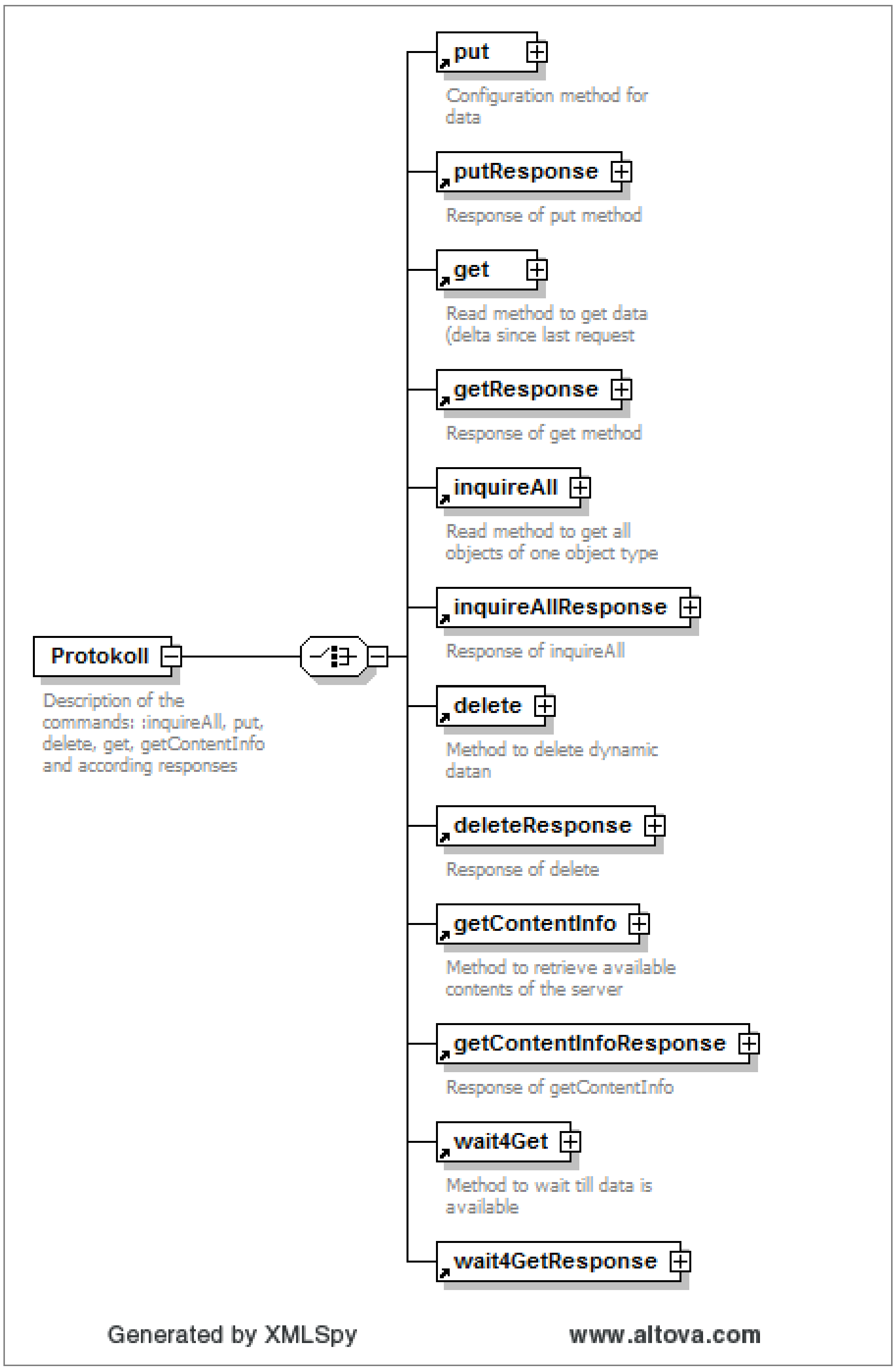


Figure 12: Communication layers - client and server

2.5 Protocol functions in detail

Available methods:



Protokoll
Description of the commands: :inquireAll, put, delete, get, getContentInfo and according responses
put
Configuration method for data
putResponse
Response of put method
get
Read method to get data (delta since last request
getResponse
Response of get method
inquireAll
Read method to get all objects of one object type
inquireAllResponse
Response of inquireAll
delete
Method to delete dynamic datan
deleteResponse
Response of delete
getContentInfo
Method to retrieve available contents of the server
getContentInfoResponse
Response of getContentInfo
wait4Get
Method to wait till data is available
wait4GetResponse
Generated by XMLSpy
www.altova.com

Figure 13: Available methods

2.5.1 Standard parameters

The standard parameters of the methods are:

Input parameters

- **UserName** and **UserPasswd** authenticate the user
  UserName and UserPasswd are transmitted as normal text. This authentication should therefore not be used for high security requirements.

- **watchdog** is a structure with which the client informs the server when the next call can be expected. This can be used by the server for monitoring the client's time (timeout).

- **storetime** identifies the start of the requested or sent data. This method is only used for access to saved historical data.

- **endStore** identifies the end of the requested data. This method is only used for access to saved historical data.

- **position** identifies the position of a pointer in the server's buffer.
  The position is received using the method inquireAll or getResponse and used for the next "get" request (see chapter 2.3.1, Data buffering and position handling).

- **filterList** is a list of objects that should be read
  This makes it possible to reduce the amount of data.

Output parameters

- **lastStart** is the timestamp of the last server start-up. If "lastStart" changes from one response to the next from the server, this is a sign that the data must be resynchronized. The client thus resynchronizes with the method inquireAll.

- **errorCode** is an error code generated in case of incorrect commands. It is not used for improper XML structures. For this there is a message from the SOAP protocol (fault).

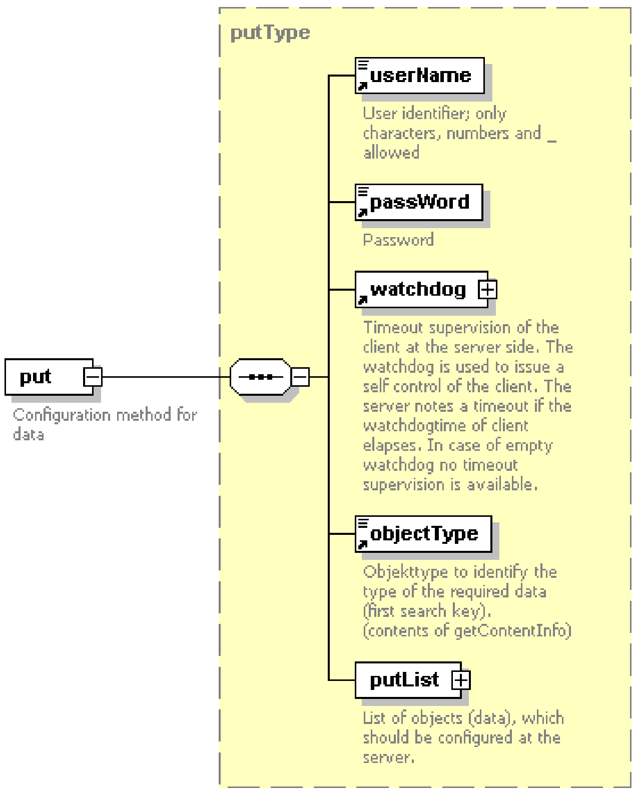
- **errorTxt** is a description of the errorCode that is readable to humans.

- **position** is the position of last data access. It must be used for subsequent data access.

- **dataList** is a list with the data requested.

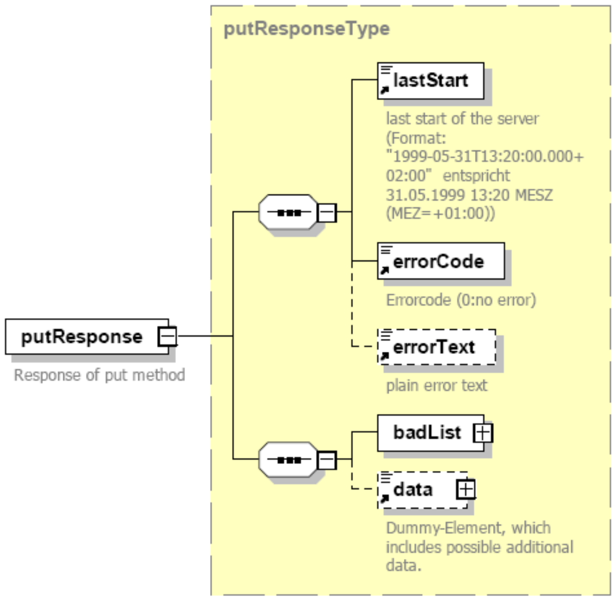
2.5.2 put

The method "put" is used to configure objects. It contains all the instances of the data that should be configured.


putType
userName
User identifier; only characters, numbers and _ allowed
passWord
Password
watchdog
Timeout supervision of the client at the server side. The watchdog is used to issue a self control of the client. The server notes a timeout if the watchdogtime of client elapses. In case of empty watchdog no timeout supervision is available.
objectType
Objekttype to identify the type of the required data (first search key). (contents of getContentInfo)
putList
List of objects (data), which should be configured at the server.
put
Configuration method for data

"putResponse" is the response to "put".

It contains all the non-configurable data, usually none.

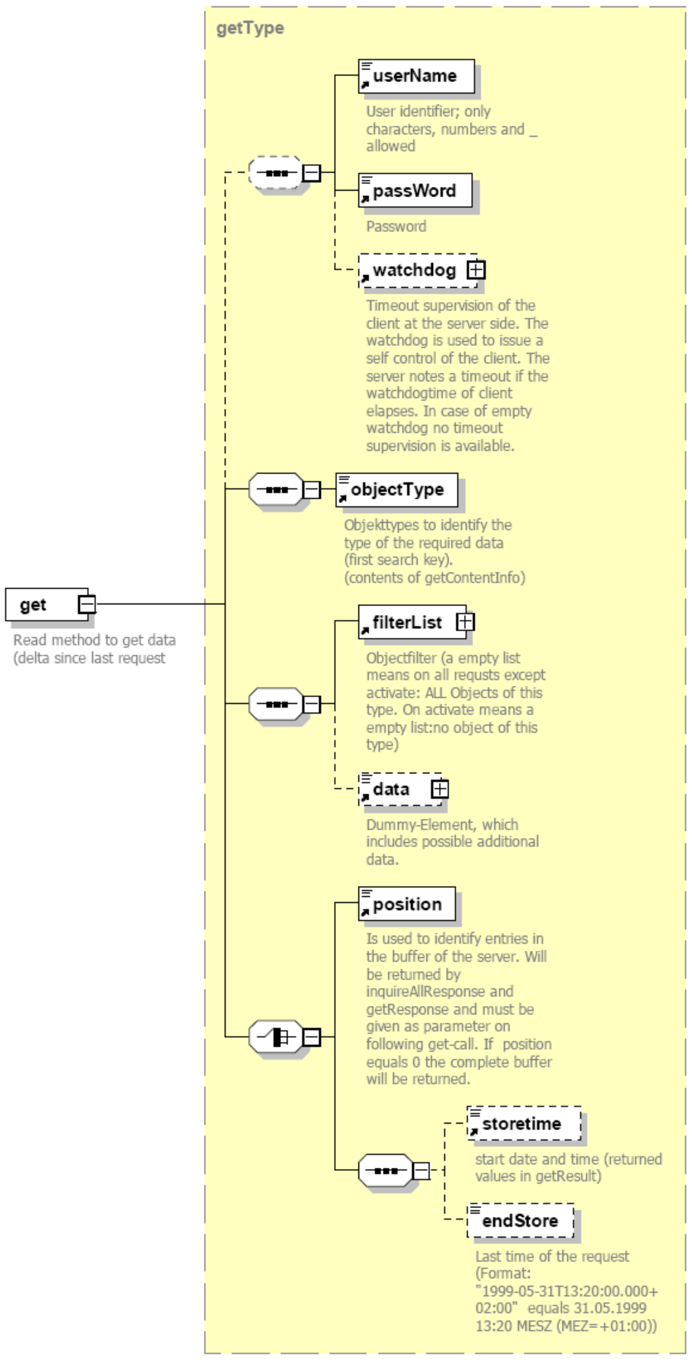

putResponseType
lastStart
last start of the server (Format: "1999-05-31T13:20:00.000+02:00" entspricht 31.05.1999 13:20 MESZ (MEZ=+01:00))
errorCode
Errorcode (0:no error)
errorText
plain error text
badList
data
Dummy-Element, which includes possible additional data.
putResponse
Response of put method

2.5.3 get

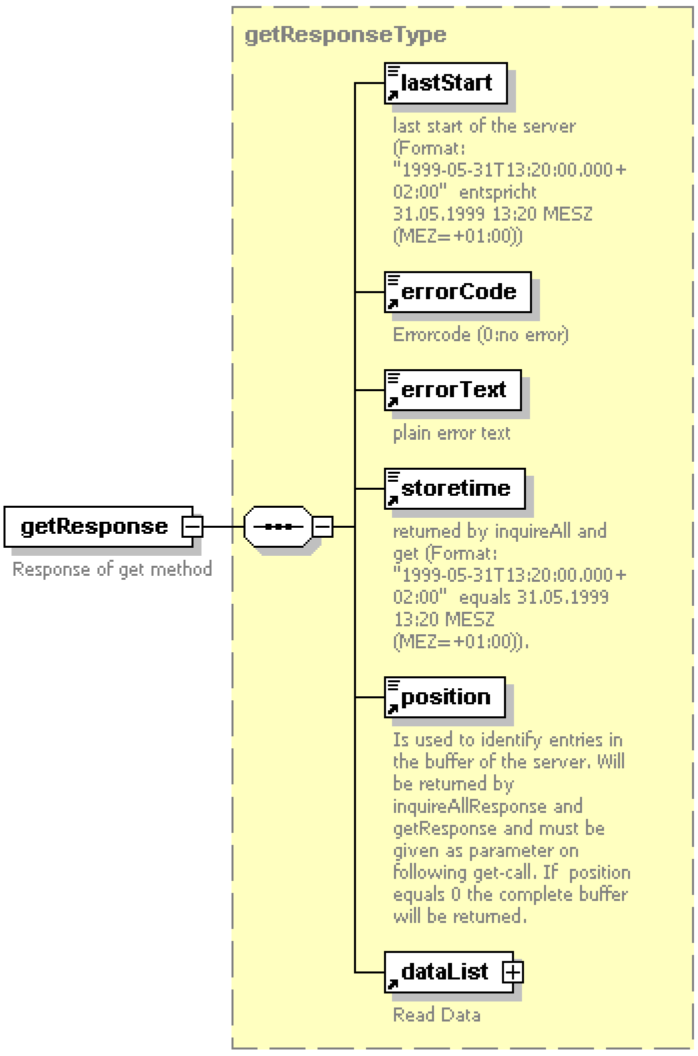
This method is used to query data.

In addition to the standard parameter

- it has either start and end time to receive all the values within this time range
- or the position number of the data to be queried. Normally, this is the position number that was returned by the last method "inquireAllResponse" or "getResponse".


getType
userName
User identifier; only characters, numbers and _ allowed
passWord
Password
watchdog
Timeout supervision of the client at the server side. The watchdog is used to issue a self control of the client. The server notes a timeout if the watchdogtime of client elapses. In case of empty watchdog no timeout supervision is available.
objectType
Objekttypes to identify the type of the required data (first search key). (contents of getContentInfo)
get
Read method to get data (delta since last request
filterList
Objectfilter (a empty list means on all requsts except activate: ALL Objects of this type. On activate means a empty list:no object of this type)
data
Dummy-Element, which includes possible additional data.
position
Is used to identify entries in the buffer of the server. Will be returned by inquireAllResponse and getResponse and must be given as parameter on following get-call. If position equals 0 the complete buffer will be returned.
storetime
start date and time (returned values in getResult)
endStore
Last time of the request (Format: "1999-05-31T13:20:00.000+02:00" equals 31.05.1999 13:20 MESZ (MEZ=+01:00))

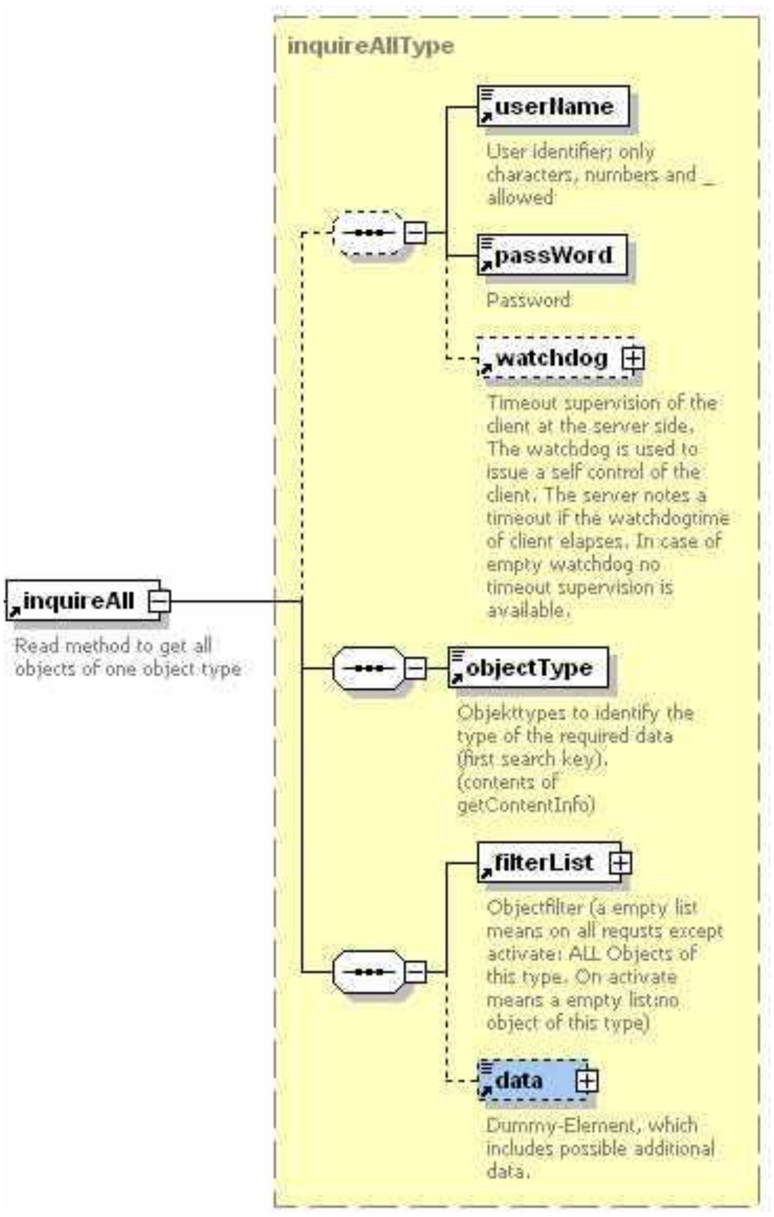
"getResponse" is the response to "get"


getResponseType
lastStart
last start of the server (Format: "1999-05-31T13:20:00.000+02:00" entspricht 31.05.1999 13:20 MESZ (MEZ=+01:00))
errorCode
Errorcode (0:no error)
errorText
plain error text
storetime
returned by inquireAll and get (Format: "1999-05-31T13:20:00.000+02:00" equals 31.05.1999 13:20 MESZ (MEZ=+01:00)).
position
Is used to identify entries in the buffer of the server. Will be returned by inquireAllResponse and getResponse and must be given as parameter on following get-call. If position equals 0 the complete buffer will be returned.
dataList
Read Data
getResponse
Response of get method

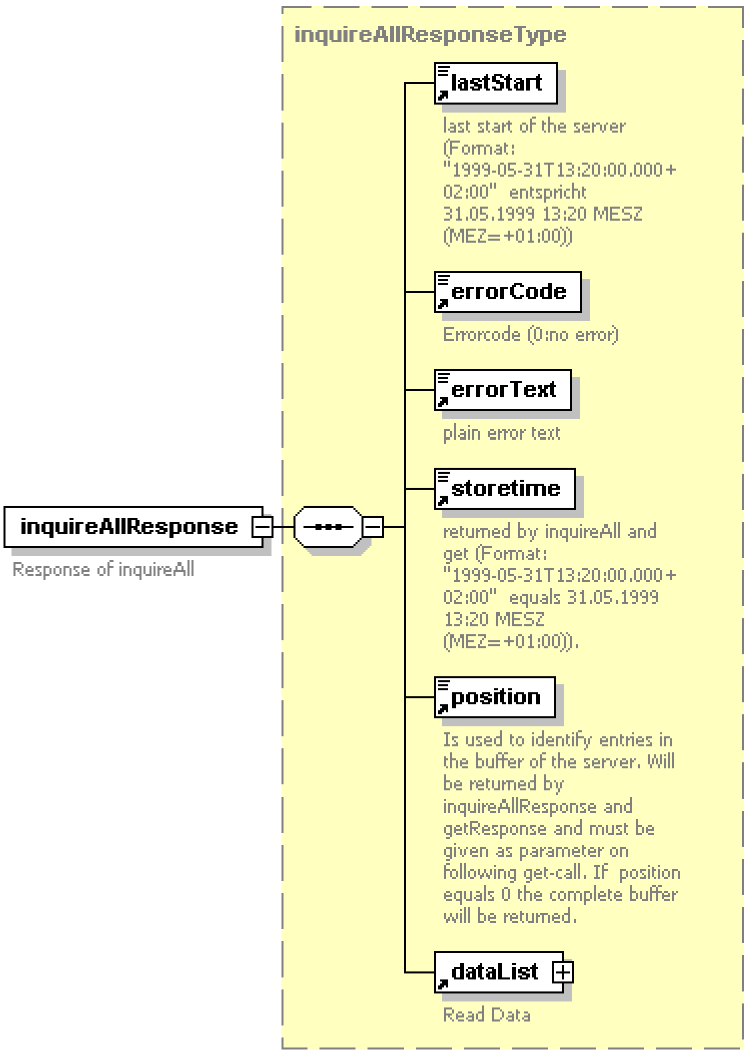
2.5.4 inquireAll

Method for querying all the data of an object type with the last status / the content of the object. This method guarantees to the client that all queried objects are included in the response.

The method "InquireAll" contains only one standard parameter.

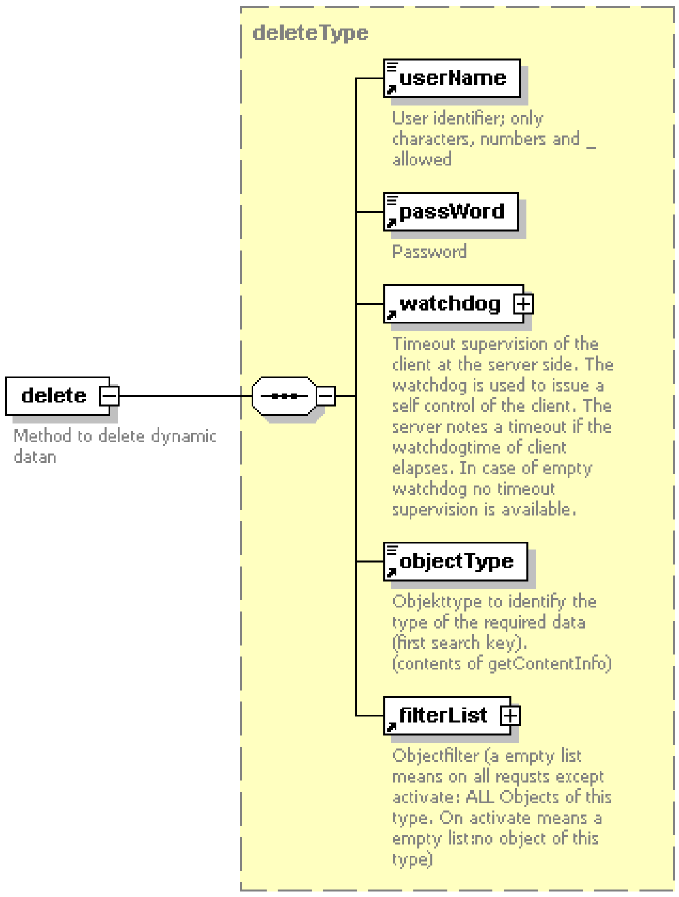

inquireAllType
userName
User identifier; only characters, numbers and _ allowed
passWord
Password
watchdog
Timeout supervision of the client at the server side. The watchdog is used to issue a self control of the client. The server notes a timeout if the watchdogtime of client elapses. In case of empty watchdog no timeout supervision is available.
inquireAll
Read method to get all objects of one object type
objectType
Objekttypes to identify the type of the required data (first search key). (contents of getContentInfo)
filterList
Objectfilter (a empty list means on all requsts except activate: ALL Objects of this type. On activate means a empty list:no object of this type)
data
Dummy-Element, which includes possible additional data.

"inquireAllResponse" is the response with all the requested content.

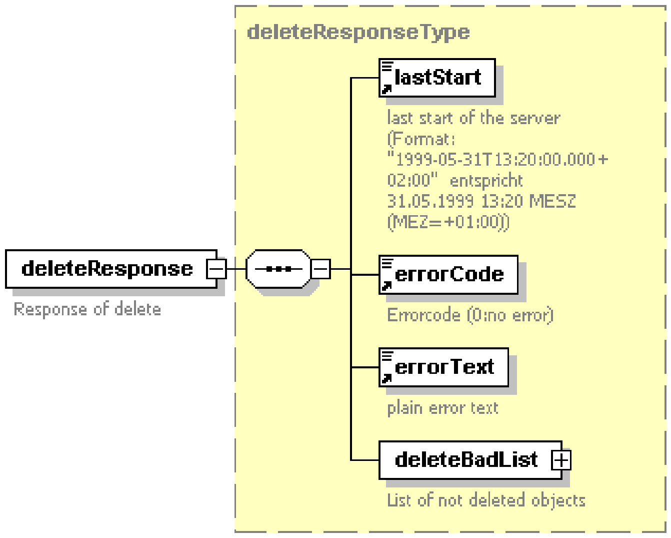

inquireAllResponseType
inquireAllResponse
Response of inquireAll
lastStart
last start of the server (Format: "1999-05-31T13:20:00.000+02:00" entspricht 31.05.1999 13:20 MESZ (MEZ=+01:00))
errorCode
Errorcode (0:no error)
errorText
plain error text
storetime
returned by inquireAll and get (Format: "1999-05-31T13:20:00.000+02:00" equals 31.05.1999 13:20 MESZ (MEZ=+01:00)).
position
Is used to identify entries in the buffer of the server. Will be returned by inquireAllResponse and getResponse and must be given as parameter on following get-call. If position equals 0 the complete buffer will be returned.
dataList
Read Data

2.5.5 delete

The method "delete" is used to delete dynamic data (for which this is permitted). Instances of the data to be deleted must be entered in the filter list.

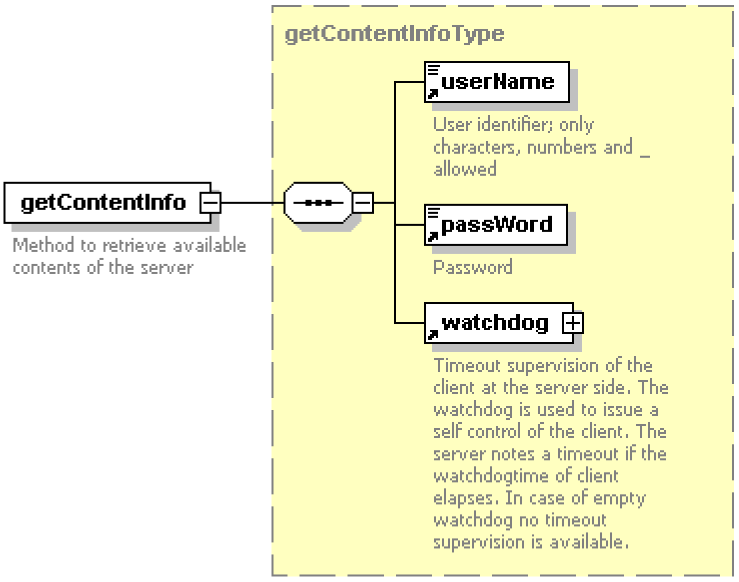

deleteType
delete
Method to delete dynamic datan
userName
User identifier; only characters, numbers and _ allowed
passWord
Password
watchdog
Timeout supervision of the client at the server side. The watchdog is used to issue a self control of the client. The server notes a timeout if the watchdogtime of client elapses. In case of empty watchdog no timeout supervision is available.
objectType
Objekttype to identify the type of the required data (first search key). (contents of getContentInfo)
filterList
Objectfilter (a empty list means on all requsts except activate: ALL Objects of this type. On activate means a empty list:no object of this type)

"deleteResponse" is the response to "delete". It contains the instances of the data that could not be deleted.

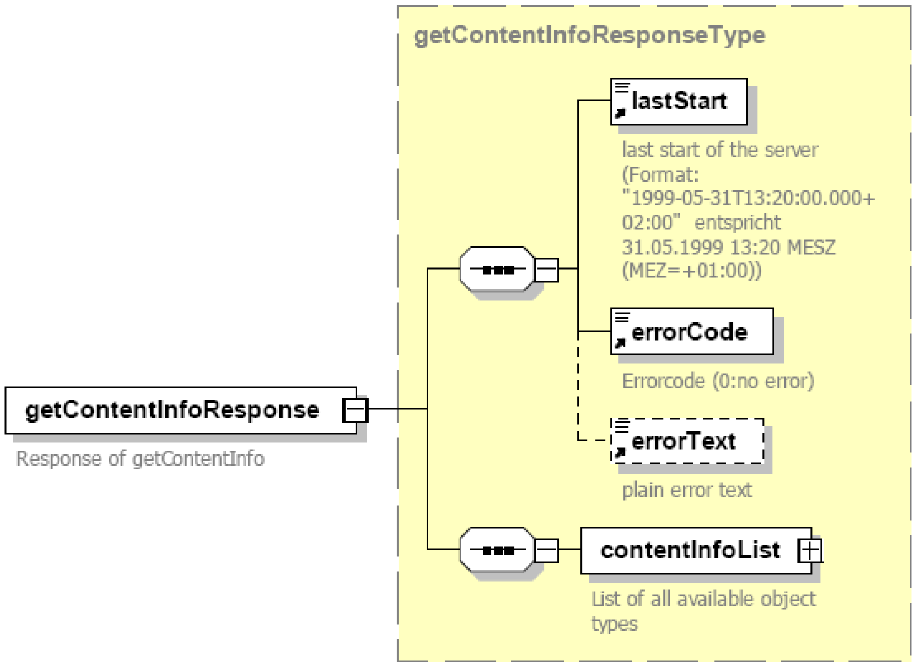

deleteResponseType
deleteResponse
Response of delete
lastStart
last start of the server (Format: "1999-05-31T13:20:00.000+02:00" entspricht 31.05.1999 13:20 MESZ (MEZ=+01:00))
errorCode
Errorcode (0:no error)
errorText
plain error text
deleteBadList
List of not deleted objects

2.5.6 getContentInfo

Method for querying the server's object contents. This method's parameters are the name of the client, the password and optionally the time for the watchdog.

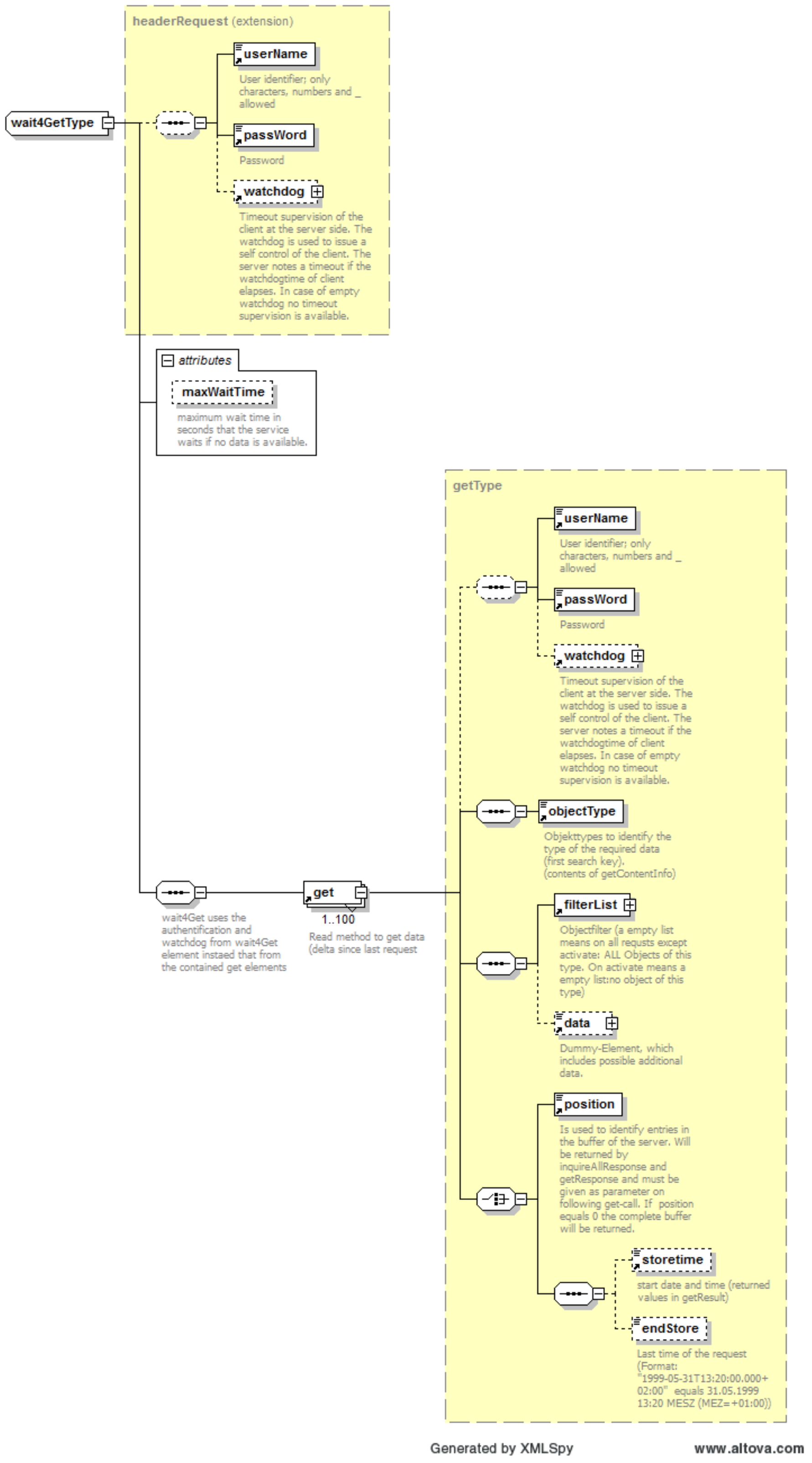

getContentInfoType
userName
User identifier; only characters, numbers and _ allowed
getContentInfo
Method to retrieve available contents of the server
passWord
Password
watchdog
Timeout supervision of the client at the server side. The watchdog is used to issue a self control of the client. The server notes a timeout if the watchdogtime of client elapses. In case of empty watchdog no timeout supervision is available.

The response "getContentInfoResponse" contains a list of the available object types with their access permissions and recommended update cycles.

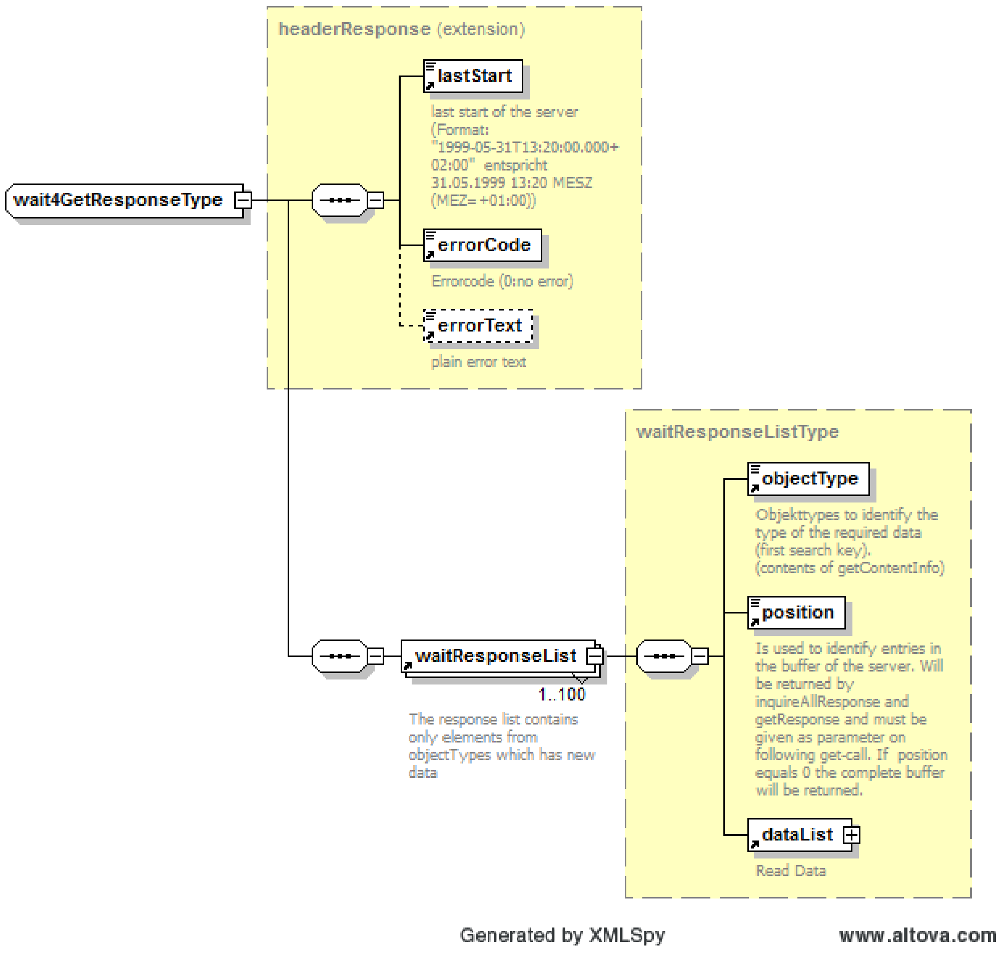

getContentInfoResponseType
lastStart
last start of the server (Format: "1999-05-31T13:20:00.000+02:00" entspricht 31.05.1999 13:20 MESZ (MEZ=+01:00))
errorCode
Errorcode (0:no error)
errorText
plain error text
getContentInfoResponse
Response of getContentInfo
contentInfoList
List of all available object types

2.5.7 wait4Get

Method for querying data from the server. This method's parameters are the same parameters as "get", but it makes it possible to query multiple object types at once.


wait4GetType
headerRequest (extension)
userName
User identifier; only characters, numbers and _ allowed
passWord
Password
watchdog
Timeout supervision of the client at the server side. The watchdog is used to issue a self control of the client. The server notes a timeout if the watchdogtime of client elapses. In case of empty watchdog no timeout supervision is available.
attributes
maxWaitTime
maximum wait time in seconds that the service waits if no data is available.
wait4Get uses the authentification and watchdog from wait4Get element instaed that from the contained get elements
get
1..100
Read method to get data (delta since last request
getType
userName
User identifier; only characters, numbers and _ allowed
passWord
Password
watchdog
Timeout supervision of the client at the server side. The watchdog is used to issue a self control of the client. The server notes a timeout if the watchdogtime of client elapses. In case of empty watchdog no timeout supervision is available.
objectType
Objekttypes to identify the type of the required data (first search key). (contents of getContentInfo)
filterList
Objectfilter (a empty list means on all requsts except activate: ALL Objects of this type. On activate means a empty list:no object of this type)
data
Dummy-Element, which includes possible additional data.
position
Is used to identify entries in the buffer of the server. Will be returned by inquireAllResponse and getResponse and must be given as parameter on following get-call. If position equals 0 the complete buffer will be returned.
storetime
start date and time (returned values in getResult)
endStore
Last time of the request (Format: "1999-05-31T13:20:00.000+02:00" equals 31.05.1999 13:20 MESZ (MEZ=+01:00))
Generated by XMLSpy
www.altova.com

"Wait4GetResponse" is the response with the changes since the last query.


headerResponse (extension)
lastStart
last start of the server (Format: "1999-05-31T13:20:00.000+02:00" entspricht 31.05.1999 13:20 MESZ (MEZ=+01:00))
errorCode
Errorcode (0:no error)
errorText
plain error text
wait4GetResponseType
waitResponseListType
objectType
Objekttypes to identify the type of the required data (first search key). (contents of getContentInfo)
position
Is used to identify entries in the buffer of the server. Will be returned by inquireAllResponse and getResponse and must be given as parameter on following get-call. If position equals 0 the complete buffer will be returned.
dataList
Read Data
waitResponseList
1..100
The response list contains only elements from objectTypes which has new data
Generated by XMLSpy
www.altova.com

2.6 Data structures

Data structures added to the protocol, e.g. the methods "put", "inquireAllResponse", etc., are defined in separate schema files for the protocol.

2.7 Definition ErrorCodes

The following errorCodes are defined in protocol.xsd:

errorCode	Meaning
0	no error
1	access error
2	buffer overflow
10	requested data unavailable
11	requested data cannot be sent
12	requested data cannot be deleted
13	values cannot be set
14	found empty object type
15	object type not found
16	error writing data
17	error creating data
18	error deleting data
19	missing filter for deletions
20	server shortly unavailable
21	missing parameters to execute the method
22	internal error
23	other registered accessing client
30	one file cannot be accessed
31	error opening a file
32	error reading a file
33	internal error, reading the archive
34	internal error, parsing the archive
35	error, parsing the archive
36	error activating
37	error deactivating
38	error reading data
39	object not found
40	invalid time range.
41	time range complete (no error)
42	missing data sets
43	returned time range incomplete

2.8 Recommended use

This section describes how the server and client can be handled in various cases. These are only recommendations from which project-specific deviations can be made.

2.8.1 Data supply with multiple subscribers

In cases in which there are multiple subscribers to the server's data that occasionally request data and do not need a real-time connection, the architecture described in the following is recommended.

- The data source contains the SOAPServerInterface, which has functions for data buffering.
  The data sink contains the SOAPClientInterface, which enables access to the SOAP server (methods inquireAll and get).

This way the client can only access the server on request. Moreover, the protocol can be implemented with little effort. In the case of an internet connection, the data source is the server and an internet utility is the client.

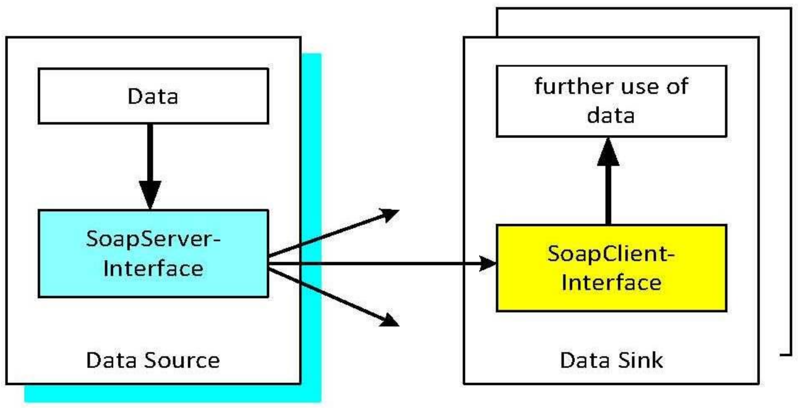


Figure 14: Data supply with multiple subscribers

2.8.2 Configuration interface

If a system (data source) needs a configuration interface to the data sink, the following architecture is recommended:

- The data source is the client
- The data sink is the server

Properties:

- Data transfer only on request
- Real-time configuration, no polling required

- Multiple configuration interfaces possible with standard communication interface to a data sink.

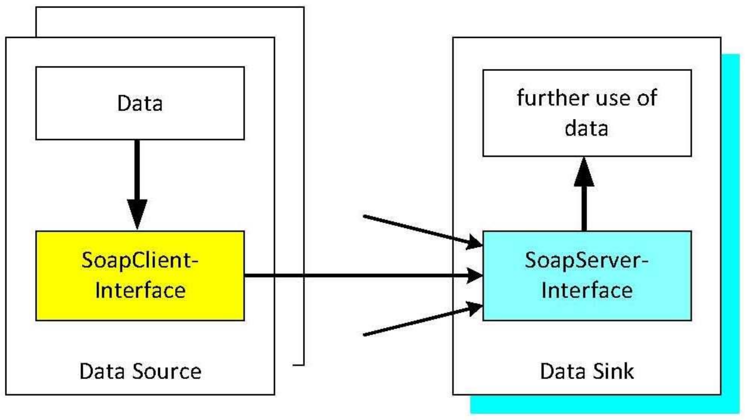


Figure 15: Multiple configuration interfaces

A case for this would be subsystems (client) that forward the data they collect, e.g. data on traffic problems. This configuration should only be used to avoid overload situations.

2.8.3 Data update between central equipment (unidirectional)

In this case, it is recommended that:

- The data source is the server
- The data sink is the client

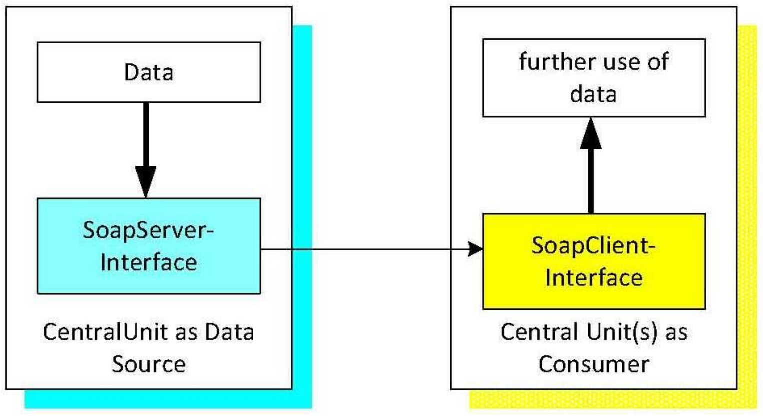


Figure 16: Configuration for data supply to the client

This is practically the same configuration as for data supply with multiple subscribers.

2.8.4 Data update between central equipment (bidirectional)

If a bidirectional interface is needed, the interface can be doubled because the central equipment is client and server at the same time.

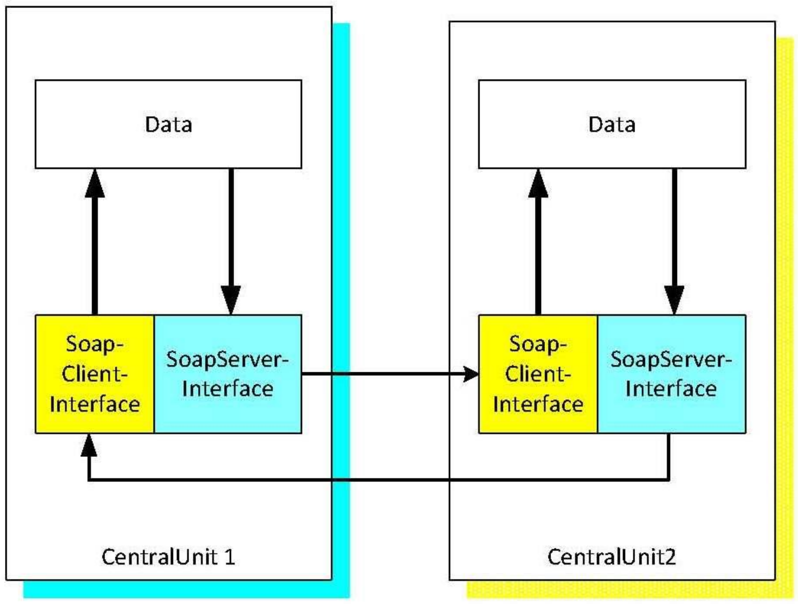


Figure 17: Configuration for data exchange: Server <-> Server

List of figures

Terms and abbreviations

Term/abbreviation	Description
AP	User program
Client	A program which wishes to use services offered by other (servers) and actively opens them to do so.
DATEX II	Specifications of Technical Committee 278 of the European Committee for Standardization (CEN) for the exchange of traffic-related data between traffic control centers.
FTP	File Transfer Protocol, a network protocol for transferring files
http	HyperText Transfer Protocol, a protocol for transferring data over a network.
TSL	Traffic signal light system
Method	The algorithms assigned to a class of objects. Also used as a synonym for function, procedure, command, action.
PT	Public Transport
OCIT	Open Communication Interface for Road Traffic Control Systems.
OCIT-C	Open Communication Interface for Road Traffic Control Systems - Center to Center. OCIT-C covers the functions for communicating between the central traffic control and traffic guidance systems.
OCIT-O	OCIT Outstations Interface between traffic control centres and traffic signal controllers for controlling and supplying the traffic signal controllers.
ODG	OCIT Developer Group
OSI	Open Systems Interconnection Reference Model, a communication model of the International Organization for Standardization (ISO) for communication protocols in computer networks.
OTS 2	Open Traffic Systems, Version 2
Server	A program that offers certain services and passively waits on incoming calls (from clients) to do so.
SOAP	Simple Object Access Protocol, it is a protocol which enables data to be exchanged between systems. SOAP uses the "Remote Procedure Call", through which it enables the functions in other computers to be called. See http://www.w3.org/TR/SOAP
SSL	Secure Socket Layer.

Soap-Server-Interface	Soap and Protocolmanager on the server side
Soap-Client-Interface	Soap and Protocolmanager on the client side
Protocolmanager	Protocol layer used for implementing commands in the buffer
TLS	Technical delivery terms for roadway stations. The TLS are a standard for the structure of traffic control systems on major German Federal high-ways. Issued by: German Federal Highway Research Institute
TCP / IP	Transmission Control Protocol / Internet Protocol, a family of network pro-tocols for the Internet.
VDV	Association of German Transportation Companies
WSDL	Web Services Description Language, a platform / programme language and protocol-independent description language for network services (web services) for exchanging messages based on XML.
XML	Extensible Markup Language, a markup language for presenting struc-tured data in the form of text. XML is used among other things for a plat-form and implementation-independent exchange of data between com-puter systems. An XML document is made up of text characters, in the most basic case in ASCII coding, and is therefore machine-readable. It does not contain binary data. The XML specification is published by the World Wide Web Consortium (W3C) as a recommendation.
XSD	XML schema, a recommendation of the World Wide Web Consortium (W3C) for defining structures for XML documents. The structure is de-scribed in the form of an XML document. Furthermore, it supports a large number of data types. The XSD schema language describes data types, individual XML schema instances (documents) and groups of such in-stances. A specific XML schema is called an XSD (XML Schema Defin-tion) and the file usually has the ending ".xsd".

Further explanations about the technical terms and abbreviations used in this document can be found in “OCIT – O Glossary V3.0”.