OCIT-C Center to Center

Transport log

OCIT-C_Protocol_V2.0_R1_D1
## Contents

1. Introduction .......................................................................................................................... 6
2. Protocol .................................................................................................................................. 8
   2.1 SOAP transmission protocol ............................................................................................ 8
       2.1.1 Technology .................................................................................................................. 8
       2.1.2 Requirements for the protocol .................................................................................... 8
       2.1.3 Security ...................................................................................................................... 9
       2.1.4 Required bandwidth ................................................................................................... 9
   2.2 Protocol functions ............................................................................................................. 9
       2.2.1 Reading data with the client ....................................................................................... 9
       2.2.2 Sending data to the server ........................................................................................ 10
   2.3 Sequence control .............................................................................................................. 11
       2.3.1 Data buffering and position handling .......................................................................... 12
       2.3.2 Excessively long transaction time ............................................................................. 12
       2.3.3 Excessively long queries ............................................................................................ 13
       2.3.4 Too many changes ..................................................................................................... 13
       2.3.5 Handling queries of historical data ............................................................................ 13
       2.3.6 Multi-client capability ................................................................................................ 14
       2.3.7 Resynchronizing ....................................................................................................... 15
       2.3.8 Bidirectional communication ..................................................................................... 16
           2.3.8.1 Bidirectional communication with client and server pair .................................... 17
           2.3.8.2 Bidirectional communication with regular status inquiry (polling) .................. 19
       2.3.9 Avoiding sampling delays ......................................................................................... 20
   2.4 OSI layers .......................................................................................................................... 21
   2.5 Protocol functions in detail .............................................................................................. 23
       2.5.1 Standard parameters .................................................................................................. 24
       2.5.2 put .............................................................................................................................. 24
2.5.3 get .................................................................................................................. 25

2.6 Data structures ..................................................................................................... 33

2.8 Recommended use .................................................................................................. 35
   2.8.1 Data supply with multiple subscribers .......................................................... 35
   2.8.2 Configuration interface .................................................................................. 35
   2.8.3 Data update between central equipment (unidirectional) ......................... 36
   2.8.4 Data update between central equipment (bidirectional) ......................... 36
## Document history

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

<table>
<thead>
<tr>
<th>Request</th>
<th>Response</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>put</td>
<td>putResponse</td>
<td>Configure objects</td>
</tr>
<tr>
<td>get</td>
<td>getResponse</td>
<td>Query modified data since the last query</td>
</tr>
<tr>
<td>inquireAll</td>
<td>inquireAllResponse</td>
<td>Query all objects of an object type</td>
</tr>
<tr>
<td>delete</td>
<td>deleteResponse</td>
<td>Delete dynamic data</td>
</tr>
<tr>
<td>getContentInfo</td>
<td>getContentInfoResponse</td>
<td>Query object contents</td>
</tr>
<tr>
<td>wait4Get</td>
<td>wait4GetResponse</td>
<td>Query modified data since the last query (like &quot;get&quot;) with the difference that the response is delayed until data are available.</td>
</tr>
</tbody>
</table>

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".

![Flow chart](image)

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.

![Diagram](image)

*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

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.
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.

![Diagram](image)

*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 incomplete 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 "operatingMessages"), 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-client access for lower-layer communication.
2.3.7 Resynchronization

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

![Diagram showing server restart sequence]

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 controller 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)

![Diagram](image)

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

In case of a missing response, the switching commands are repeated after a configurable delay. This applies regardless of where the fault arises (no status transition occurs 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".

Figure 8: Switching a sign (process order in case of faults)
Status, content and connection monitoring

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.
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:

![Sequence Diagram](image)

**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.

![Diagram of communication layers - client and server]

*Figure 12: Communication layers - client and server*
2.5 Protocol functions in detail

Available methods:

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 re-synchronized. 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.
"putResponse" is the response to "put". It contains all the non-configurable data, usually none.

### 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".
"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 (Error 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.
"inquireAllResponse" is the response with all the requested content.
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.

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

The response "getContentInfoResponse" contains a list of the available object types with their access permissions and recommended update cycles.
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.
"Wait4GetResponse" is the response with the changes since the last query.

### 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:

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

![Diagram](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.

![Diagram](image1.png)

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

![Diagram](image2.png)

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

Abbildung 1: Übliche Reihenfolge zum Lesen von Daten vom SOAPserverInterface ..........10
Abbildung 2: Übliche Reihenfolge zum Schreiben oder Konfigurieren von Daten zum
SOAPserverInterface ........................................................................................................11
Abbildung 3: Datenpufferung am Server und Positionsbehandlung am Client ...............12
Abbildung 4: Handhabung langer Transaktionszeiten (ähnliche Reihenfolge gilt auch für
„get“) ................................................................................................................................13
Abbildung 5: Multi Client Fähigkeit ..................................................................................15
Abbildung 6: Neustart des Servers ..................................................................................16
Abbildung 7: Schalten eines Schildes (Ablaufreihenfolge im Gutfall) .........................17
Abbildung 8: Schalten eines Schildes (Ablaufreihenfolge im Fehlerfall) ....................18
Abbildung 9: Schaltzustand .............................................................................................19
Abbildung 10: Regelmäßige Zustandsabfrage .................................................................20
Abbildung 11: Kommunikationschichten Client und Server ........................................21
Abbildung 12: Kommunikationschichten Client und Server ........................................22
Abbildung 13: Verfügbare Methoden ................................................................................23
Abbildung 14: Datenlieferung an mehrere Abnehmer .....................................................35
Abbildung 15: Mehrere Konfigurations-Schnittstellen ..................................................36
Abbildung 16: Konfiguration bei Datenlieferung an Client ..........................................36
Abbildung 17: Konfiguration bei Datenaustausch Server <-> Server .............................37
## Terms and abbreviations

<table>
<thead>
<tr>
<th>Term/abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>User program</td>
</tr>
<tr>
<td>Client</td>
<td>A program which wishes to use services offered by other (servers) and actively opens them to do so.</td>
</tr>
<tr>
<td>DATEX II</td>
<td>Specifications of Technical Committee 278 of the European Committee for Standardization (CEN) for the exchange of traffic-related data between traffic control centers.</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol, a network protocol for transferring files</td>
</tr>
<tr>
<td>http</td>
<td>HyperText Transfer Protocol, a protocol for transferring data over a network.</td>
</tr>
<tr>
<td>TSL</td>
<td>Traffic signal light system</td>
</tr>
<tr>
<td>Method</td>
<td>The algorithms assigned to a class of objects. Also used as a synonym for function, procedure, command, action.</td>
</tr>
<tr>
<td>PT</td>
<td>Public Transport</td>
</tr>
<tr>
<td>OCIT</td>
<td>Open Communication Interface for Road Traffic Control Systems.</td>
</tr>
<tr>
<td>OCIT-C</td>
<td>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.</td>
</tr>
<tr>
<td>OCIT-O</td>
<td>OCIT Outstations Interface between traffic control centres and traffic signal controllers for controlling and supplying the traffic signal controllers.</td>
</tr>
<tr>
<td>ODG</td>
<td>OCIT Developer Group</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection Reference Model, a communication model of the International Organization for Standardization (ISO) for communication protocols in computer networks.</td>
</tr>
<tr>
<td>OTS 2</td>
<td>Open Traffic Systems, Version 2</td>
</tr>
<tr>
<td>Server</td>
<td>A program that offers certain services and passively waits on incoming calls (from clients) to do so.</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol, it is a protocol which enables data to be exchanged between systems. SOAP uses the &quot;Remote Procedure Call&quot;, through which it enables the functions in other computers to be called. See <a href="http://www.w3.org/TR/SOAP">http://www.w3.org/TR/SOAP</a></td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer.</td>
</tr>
<tr>
<td><strong>Soap-Server-Interface</strong></td>
<td>Soap and Protocolmanager on the server side</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td><strong>Soap-Client-Interface</strong></td>
<td>Soap and Protocolmanager on the client side</td>
</tr>
<tr>
<td><strong>Protocolmanager</strong></td>
<td>Protocol layer used for implementing commands in the buffer</td>
</tr>
<tr>
<td><strong>TLS</strong></td>
<td>Technical delivery terms for roadway stations. The TLS are a standard for the structure of traffic control systems on major German Federal highways. Issued by: German Federal Highway Research Institute</td>
</tr>
<tr>
<td><strong>TCP / IP</strong></td>
<td>Transmission Control Protocol / Internet Protocol, a family of network protocols for the Internet.</td>
</tr>
<tr>
<td><strong>VDV</strong></td>
<td>Association of German Transportation Companies</td>
</tr>
<tr>
<td><strong>WSDL</strong></td>
<td>Web Services Description Language, a platform / programme language and protocol-independent description language for network services (web services) for exchanging messages based on XML.</td>
</tr>
<tr>
<td><strong>XML</strong></td>
<td>Extensible Markup Language, a markup language for presenting structured data in the form of text. XML is used among other things for a platform and implementation-independent exchange of data between computer 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.</td>
</tr>
<tr>
<td><strong>XSD</strong></td>
<td>XML schema, a recommendation of the World Wide Web Consortium (W3C) for defining structures for XML documents. The structure is described 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 instances. A specific XML schema is called an XSD (XML Schema Definition) and the file usually has the ending &quot;.xsd&quot;.</td>
</tr>
</tbody>
</table>

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