Description of XML based messages for data exchange between OTT netDL and a HTTP Server

1. Overview 4
2. General principles 5
2.1. Message Types 5
2.2. Basic authentication 5
2.3. Client/Server model 5
2.4. Conventions in XML schemas 6
2.5. Compression 6
2.6. Data Types 6
3. Format of data time series messages 9
3.1. Basic structure of a data time series message 10
3.2. Data messages including events 12
3.3. Data messages including observer information 13
3.4. Data messages including additional station information 14
3.5. Data messages for instantaneous values 15
4. Command messages 16
4.1. Structure of command messages 16
4.1.1. Station specific commands 17
4.1.2. Channel specific commands 29
4.2. Acknowledgement of command messages 32
5. Querying data from the logger 34
5.1. Basic structure of a query message 34
5.2. Refining queries for individual channels 35
5.3. Querying for instantaneous values 36
5.4. Addressing a specific station on a data server 36
5.5. Refining queries for individual events 38
5.5.1. Event Types 38
5.6. Including information about the station 39
5.7. Requesting data in compressed format 39
6. Logger operation use cases 40
6.1. Self-timed data transmission 40
6.1.1. Automatic adjustment of transmission interval 40
6.1.2. Redundant communication paths 40
6.1.3. Redundant servers 41
6.2. Alarm actions 41
6.2.1. Alarm messages 42
6.3. Modifying channel specific settings 43
6.4. Firmware Update 43
6.5. Configuration Update 43
6.6. Time synchronisation 44
7. Logger communication wakeup 44
7.1. Maintenance windows 44
7.1.1. Modes for handling connection requests 45
8. Appendix A: Sequence diagrams 48
8.1. Diagram: Self-timed data transmission 48
8.2. Diagram: Processing general command messages 48
8.2.1. Diagram: Process UpdateFirmware command 49
8.2.2. Diagram: Process UpdateConfiguration command 51

8.2.3. Diagram: Process SaveConfiguration command 52
8.3. Diagram: Processing a query 53
8.4. Diagram: Sending alarm messages 53
8.5. Diagram: Maintenance window (connect to server on RING) 55
8.6. Diagram: Maintenance window (accept IP connection) 56
9. Appendix B: XML Schemas 57
9.1. OTT_Query 57
9.2. OTT_Alarm 57
9.3. OTT_Response 58
9.4. OTT_Data 58
9.4.1. General Structure 58
9.4.2. Channel Data 58
9.4.3. Complete schema 60
9.5. OTT_Command 61
9.5.1. General Structure 61
9.5.2. SetMaintenanceWindow 62
9.5.3. SetIPDataTransmission 63
9.5.4. SetInternetConnection 63
9.5.5. Channel commands 64

1. Overview

The data logger OTT netDL1000 comes with different interfaces and protocols to enable its integration in IT environments that make use of typical IP communication protocols like HTTP, FTP and SMTP.

In a modern environmental monitoring network, the traditional polling of field stations by a proprietary software via modem connection is replaced by a self-timed transmission of data using IP communication with standard internet protocols over different communication media, e.g. GPRS or ADSL.

With OTT netDL1000 data loggers a modern network can be easily set up, where the loggers transmit their data self-timed in regular intervals to a central web server using the well known HTTP protocol and XML based data formats, that can be easily processed.

The web server does not only receive the data, but can also be used to issue certain commands to the logger, e.g. to modify the logger configuration or update the firmware. This is done using so called command messages.

Communication between loggers and servers follows the client/server model. This means that communication is always initiated by the logger (=client), which allows the use of dynamic IP addresses without having the need, that the server knows the logger IP address.

In this document the details of the communication between OTT netDL1000 and an HTTP server are described, i.e. the different types and formats of XML messages that are exchanged between them.

2. General principles

All communication between data logger and central web server described in this document is based on the Hypertext Transfer Protocol (HTTP) in version 1.0 or above. In this document only data transmission via HTTP is explained. The OTT netDL1000 logger however supports in principle also IP based transmission via FTP and SMTP.

2.1. Message Types

Messages exchanged between logger and server are based on XML. Below is a list of message types with the corresponding XML schema file:

Message Type	Schema file	Meaning
Data Message	OTT_Data.xsd	Message from logger to server containing data (measured values, events, ...)
Command Message	OTT_Command.xsd	Message from server to logger containing commands that shall be executed by the logger
Query Message	OTT_Query.xsd	Message from server to logger containing a query for data
Alarm Message	OTT_Alarm.xsd	Message from logger to server containing an alarm message (e.g. a limit was crossed)
Response Message	OTT_Response.xsd	Response/acknowledgement message between logger and server. Used e.g. to acknowledge the execution of a command or a successful file upload.

The messages are typically sent in the body of HTTP POST requests, where the “Content-Type” attribute of the HTTP header is set to “application/xml”.

2.2. Basic authentication

To provide a certain security level, the logger can be configured to use HTTP Basic authentication, when sending HTTP requests to the server. Username and passwort can be defined in the logger configuration using the operating software or via the SetHttpServer command.

2.3. Client/Server model

IP Communication between logger and server works in classical client/server manner, i.e. communication is always initiated by the client. This is especially useful in environments, where dynamic IP addresses are assigned to the loggers and so no straightforward way exists to contact the logger directly.

When the server wants to perform certain actions on the logger, command messages are used. Whenever a logger connects to the server to transmit data, it also checks if a command message is available, which shall be executed. This allows for effective control of the logger without the need to establish directly a communication to the logger.

For cases where this is necessary, special features are implemented. See chapter “Logger communication wakeup” for details.

2.4. Conventions in XML schemas

To ensure consistent naming in the XML schema documents, UpperCamelCase is used for Elements and Types, lowerCamelCase is used for attributes.

Element examples:
```
<StationCommands>
<ConnectMode>
```

Attribute examples:
```
channelId
dataScript
```

2.5. Compression

As the exchanged XML messages are by nature not very compact and thus increase the data transfer volume, an option exists to compress the messages before they are transmitted. This option can be explicitly enabled in the logger configuration and a Query request can explicitly ask for compressed data. OTT netDL1000 supports the “Deflate” compression method.

2.6. Data Types

The OTT netDL1000 data logger manages different kinds of data. These data comprise the following:

- storage values of the different channels
- instantaneous values of the different channels
- different types of channel specific events, that are generated by the data logger
- different types of station specific events, that are generated by the data logger

Instantaneous values are the values that are actual measurement values provided by a sensor, when the logger asks for a current value (sampling value). Typically the instantaneous values are not logged directly, but a storage value is generated averaging a number of instantaneous values, e.g. the logger asks a sensor every minute for an instantaneous value, but stores every 10 minutes the average of the 10 instantaneous values as a storage value.

Finally the data logger creates different kinds of events, e.g. when the value of a channel exceeds a certain limit or when a communication fails. These events can be distinguished into events, which are specific to a certain channel (e.g. the limit event) or are general events of the station (e.g. the communication failure). Events can be further divided into certain categories, e.g. limit events, gradient events or error events.

When configuring a self-timed data transmission, the user can define, which kind of data shall be included in the data transmission. Also when using explicit query commands, to query for data, the user can define which kinds of data shall be included in the response to the query. See also chapters “Format of data time series messages” and “

Querying data from the logger” for details.

3. Format of data time series messages

The OTT netDL1000 can be configured to transmit data self-timed in regular intervals as XML messages to a web server using the HTTP POST method. For details how to configure the OTT netDL1000 using the operating program, see the OTT netDL1000 operating manual and the chapter “SetIPDataTransmission command”. See also chapters “Self-timed data transmission” and “Diagram: Self-timed data transmission” for details about self-timed transmission.

Using Query messages the OTT netDL1000 can be explicitly forced to transmit such a time series independently of the regular transmission interval (see also the chapters:”

Querying data from the logger" and the sequence diagram "Diagram: Processing a query" in Appendix A).

In this chapter, the format of these XML messages, containing time series of data from the logger is described. The expression "data" is very generic here and comprises the actual measured values, as well as additional information, like events that occurred in the logger, or metadata, like current firmware version of the logger.

3.1. Basic structure of a data time series message

The file OTT_Data.xsd contains the XML Schema of data time series messages. This chapter describes the format of these messages including examples.

Below is a simple example to get started with the structure of a data time series message:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<StationDataList>
  <StationData stationId="0123456789" name="TestStation" timezone="+01:00">
    <ChannelData channelId="0010" name="Water level" unit="m">
      <Values>
        <VT t="2009-01-01T00:15:00">3.56</VT>
        <VT t="2009-01-01T00:30:00">3.57</VT>
        <VT t="2009-01-01T00:45:00">3.57</VT>
        <VT t="2009-01-01T01:00:00">3.57</VT>
        <VT t="2009-01-01T01:15:00">3.58</VT>
        <VT t="2009-01-01T01:30:00">3.58</VT>
        <VT t="2009-01-01T01:45:00">3.58</VT>
        <VT t="2009-01-01T02:00:00">3.58</VT>
      </Values>
    </ChannelData>
    <ChannelData channelId="0020" name="Water temperature" unit="Celsius">
      <Values>
        <VT t="2009-01-01T00:30:00">6.3</VT>
        <VT t="2009-01-01T01:00:00">6.3</VT>
        <VT t="2009-01-01T01:30:00" errorcode="5"
                                    errortext="Line Break">0</VT>
        <VT t="2009-01-01T02:00:00">6.2</VT>
      </Values>
    </ChannelData>
  </StationData>
</StationDataList>
```

The root element of a data message is a <StationDataList> element, which contains a number of <StationData> elements as its child elements. Each <StationData> element contains the data of one measuring station. When data are transmitted from one station, only one <StationData> element is existing. This is the case, when data are sent from the OTT netDL1000 logger. However this message format could also be used to transmit data from a data server, that manages data of different stations. In this case multiple <StationData> elements would be used.

The <StationData> element contains additional information about the station as attributes, including the required `stationId`, as well as the `timezone` information and the optional `name`

of the station. By setting the timezone to “00:00” in the logger configuration, the logger timestamps are in UTC time (if synchronized by NTP).

Optionally an attribute `exchangeId` can be used to provide a unique identification of this station, which is independent of the mandatory `stationId` attribute.

When the data message is sent in response to a query, where the optional attribute `ackId` was set, the `ackId` attribute with the same value is used in the <StationData> element to refer to the corresponding query.

Each <StationData> element contains a number of `<ChannelData>` elements, which contain data for the individual channels of a logger. The `channelId` attribute of the `<ChannelData>` element is necessary to identify the channel clearly. Optionally the `name` of the channel and the `unit` of the measured data can be included as attributes, as in the example.

Optionally an attribute `exchangeId` can be used to provide a unique identification of this `<ChannelData>` element, which is independent of the mandatory `channelId` attribute. It is even possible to have `<ChannelData>` elements, which do not contain data from an actual channel, but data derived from an actual channel. E.g. observer values, which are normally stored as events (see next chapter) can be transmitted as a `<ChannelData>` time series, where the `channelId` attribute contains the channel, from which the data were derived and the attribute `exchangeId` is used to uniquely identify this (derived) time series.

There are different kinds of data for every channel available. The most obvious kind of data are the storage values of a channel, which are contained within a `<Values>` element. For every storage value, a `<VT>` element is added as a child element to the `<Values>` element, as illustrated in the example. The `<VT>` element contains a value with associated timestamp.

The content of each `<VT>` element is the storage value. Metadata of the storage value are described as attributes of the `<VT>` element. A mandatory attribute is the timestamp attribute `t`, which contains the local timestamp of the storage value. The combination of the local timestamp `t` and the `timezone`, that is defined in the <StationData> element, results in an unambiguous global timestamp.

In case of an error, when acquiring the storage value, the optional attribute `errorcode` is set to a value>0, indicating which kind of error occurred. The optional attribute `errortext` can provide an error message in plain text. Below is a list of the most common generic error codes. For a more detailed list of possible error codes, see the logger manual.

Errorcode	Meaning
01	A/D conversion fault
02	Communication error on sensor interface
03	Over/Underflow
05	Line break

The format for other kinds of data is explained in the next chapters.

3.2. Data messages including events

The OTT netDL1000 data logger creates different kinds of events, e.g. when the value of a channel exceeds a certain limit or when a communication fails. These events can be distinguished into events, which are specific to a certain channel (e.g. the limit event) or are general events of the station (e.g. the communication failure). Events can be further divided into certain categories, e.g. limit events, gradient events or error events. See chapter “Refining queries for individual events” for a list of event types.

These events can be embedded easily into the data time series messages. For data messages that are automatically transmitted in regular intervals to a server, the inclusion of events can be enabled in the configuration of the logger.

When data are queried explicitly from the logger, it can be defined in detail in the query, which events shall be included in the response. For details see chapter “Refining queries for individual events”.

Station specific events are contained within the `<StationEvents>` element, which is a child element of the `<StationData>` element. For each station specific event, an `<Event>` element is added as a child element to the `<StationEvents>` element.

Channel specific events are contained within the `<Events>` element, which is a child element of the `<ChannelData>` element. For each channel specific event, an `<Event>` element is added as a child element to the `<Events>` element.

Each `<Event>` element has a mandatory timestamp attribute `t` and an attribute `type`, that specifies the type of the event.

Which kinds of events shall be included in a message can be defined in detail in the query message respectively in the configuration of the data logger for automatically transmitted messages.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<StationDataList>
  <StationData stationId="0123456789" name="TestStation" timezone="+01:00">
    <StationEvents>
      <Event t="2009-01-01T12:00:06" type="error.communication">
      Modem on COM1 does not respond
      </Event>
    </StationEvents>
    <ChannelData channelId="0010" name="Water level" unit="m">
      <Events>
        <Event t="2009-01-01T19:10:00" type="info.limit">
        Value 10.1 has exceeded limit (10)
        </Event>
      </Events>
    </ChannelData>
  </StationData>
</ StationDataList>
```

3.3. Data messages including observer information

When an observer in the field creates an observer entry in the logger, this entry will be stored as an event, which can be included in the OTT-ML message.
In the simplest case, when an observer only has a look at the current values, without entering a value, a station event of type “info.observer” will be created as shown below:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<StationDataList>
  <StationData stationId="0123456789" name="TestStation" timezone="+01:00">
    <StationEvents>
      <Event t="2011-01-01T12:15:06" type="info.observer">
      Observer activated
      </Event>
    </StationEvents>
  </StationData>
</ StationDataList>
```

When an observer enters a textual comment, a station event of type “info.observer.text” will be created as shown below:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<StationDataList>
  <StationData stationId="0123456789" name="TestStation" timezone="+01:00">
    <StationEvents>
      <Event t="2011-01-01T12:15:06" type="info.observer.text"
                 textCode="0001">
      Surface covered with ice
      </Event>
    </StationEvents>
  </StationData>
</ StationDataList>
```

If the text is taken from a list of predefined texts with associated codes, the optional attribute *textCode* contains this code.

When the observer enters a value without adjusting the scaling, in this specific channel an event of type “info.observer.valueEntry” will be created as illustrated below:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<StationDataList>
  <StationData stationId="0123456789" name="TestStation" timezone="+01:00">
    <ChannelData channelId="0010" name="Water level" unit="m">
      <Values>
        <VT t="2011-01-01T19:15:00">3.11</VT>
        <VT t="2011-01-01T19:30:00">3.12</VT>
        <VT t="2011-01-01T19:45:00">3.12</VT>
      </Values>
      <Events>
        <Event t="2011-01-01T19:36:17" type="info.observer.valueEntry"
                  value="3.14" >
        Value 3.14 entered
        </Event>
      </Events>
```

```
    </ChannelData>
  </StationData>
</StationDataList>
```

For events of type “info.observer.valueEntry” the attribute *value* contains the value, the observer has entered.

When the observer enters a value with adjusting the scaling, in this specific channel an event of type “info.observer.valueAdjust” will be created as illustrated below:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<StationDataList>
  <StationData stationId="0123456789" name="TestStation" timezone="+01:00">
    <ChannelData channelId="0010" name="Water level" unit="m">
      <Values>
        <VT t="2011-01-01T19:15:00">3.11</VT>
        <VT t="2011-01-01T19:30:00">3.12</VT>
        <VT t="2011-01-01T19:45:00">3.14</VT>
      </Values>
      <Events>
        <Event t="2009-01-01T19:42:00" type="info.observer.valueAdjust"
                value="3.14" value2="3.12" >
        Value changed from 3.12 to 3.14
        </Event>
      </Events>
    </ChannelData>
  </StationData>
</StationDataList>
```

For events of type “info.observer.valueAdjust” the attribute *value* contains the value, the observer has entered and the attribute *value2* contains the last measured value at this time.

3.4. Data messages including additional station information

For self-timed data transmission, the optional inclusion of additional station information can be activated in the logger configuration. For queries, the attribute `includeStationInfo` is used to enforce inclusion of this information in the response.

The element `<StationInfo>` element contains the additional information as different attributes, as illustrated in the example below. Note that in this example the actual values and events are omitted.

The optional `<StationInfo>` element is the first child element of a `<StationData>` element.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<StationDataList>
  <StationData stationId="0123456789" name="TestStation" timezone="+01:00">
    <StationInfo time="2009-02-20T14:15:11"
                 firmware="1.12"
                 configtime="2008-12-13T10:25:09"
                 paramtime="2008-12-15T16:11:20"
```

```
                batteryVoltage="12.4"
                gsmSignal="20"
                ipAddress="250.145.64.10"
                temperature="1"
                transmissionCycle="600" />
    ... additional Events or Values here
  </StationData>
</ StationDataList>
```

Below is a table with the different attributes and their meaning:

Attribute	Meaning
time	Current logger time when message was sent
firmware	Current firmware version on logger
configtime	Time when new configuration was written, that made structural changes (e.g. new channels)
paramtime	Time when existing configuration was modified (e.g. transmission interval), but no structural change was made
batteryVoltage	Voltage of power supply
gsmSignal	Current signal quality of GSM/GPRS module
ipAddress	Current IP address of logger
temperature	Current temperature on logger mainboard
transmissionCycle	Currently used transmission cycle in seconds for self-timed data transmission

Note that most of these attributes are optional and the inclusion of an specific attribute depends on the individual logger configuration.

3.5. Data messages for instantaneous values

A special kind of data are instantaneous values, which provide a snapshot of a sensor value at a specific timestamp. Typically multiple instantaneous values are taken by the logger and averaged to create storage values, which are representative for a certain interval.

Below is an example describing the instantaneous values of 2 channels:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<StationDataList>
  <StationData stationId="0123456789" name="TestStation" timezone="+01:00">
    <ChannelData channelId="0010" name="Water level" unit="m">
      <VInst t="2009-01-01T16:00:00">3.62</VInst>
    </ChannelData>
    <ChannelData channelId="0020" name="Water temperature" unit="Celsius">
      <VInst t="2009-01-01T16:00:00">6.4</VInst>
    </ChannelData>
  </StationData>
</StationDataList>
```

In the chapter “Querying instantaneous values” the format of queries is described, which returns messages as the one in the example above.

4. Command messages

With so called command messages, the logger can be instructed to perform certain operations. The logger can be configured to check for a new command message, whenever it transmits data to a web server.

In the logger configuration or by using the attribute `commandScript` of the SetHttpServer command, the URL of a server side script can be specified, that the logger uses to check for command messages. Whenever the logger wants to check, if a command message is waiting for execution, it sends an HTTP GET request to this script.

Example:

The URL of the command script is configured as http://test.ott.com/command.php

To check for a command message, the logger will send a GET request with this URL:
http://test.ott.com/command.php?stationid=0123456789&action=requestcommand

So the logger appends the URL parameters stationid=..... to identify itself and "action=requestcommand" to indicate that a request for a command message shall be performed.

As with all HTTP requests, this request uses HTTP basic authentication, if configured in the logger settings.

If a command message shall be executed by the logger, then the result of this GET request is the command message in XML format, as described below. Otherwise the result of the request is empty.

When the logger has retrieved a command message, the commands contained in the message are executed and after execution, an acknowledge message is sent back to the server to indicate success or failure of the execution, as described further below.

See also the sequence diagram "Diagram: Processing general command messages" in Appendix A, which illustrates the message flow between logger and server.

4.1. Structure of command messages

A command message contains one or more commands to be executed by the data logger. In this chapter, the basic structure of a command message is explained. The next chapters explain every available command in detail.

The file OTT_Command.xsd contains the XML Schema of a command message.

Below is a first example of a command message to illustrate the basic structure.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <Logging ackId="1" storageInterval="3600" />
    <ChannelCommands channelId="0010">
      <Limit ackId="2" lower="10" />
    </ChannelCommands>
  </StationCommands>
</CommandList>
```

The root element of a command message is the <CommandList> element, which contains as child elements a list of <StationCommands> elements. A <StationCommands> element contains as direct child elements commands, that are station specific. A <StationCommands> element has an optional attribute stationID to identify the station, which is necessary, if the recipient of the command message can handle commands for multiple stations.

In the example, there is one station specific command <Logging>, which can be used to modify the sampling and storage intervals of the logger. This command is explained in more detail further below.

The optional attribute ackId (>0) of every command element can be used to identify this command unambiguously. When this attribute is present, the acknowledge message will refer to this ackId. For details on the acknowledgement of command messages see chapter “Acknowledgement of command messages”.

In addition to station specific commands, there are channel specific commands, which affect only one channel. These channel specific commands are embedded as child elements of <ChannelCommands> elements, which are themselves optional child elements of a <StationCommands> element. A <ChannelCommands> element has a mandatory attribute channelId to identify the channel.

In the example above, there is one channel specific command for channel “0010” to set the lower limit of the channel to a value of 10. This command is explained in more detail further below.

For every channel a <ChannelCommands> element could be embedded in the parent <StationCommands> element.

4.1.1. Station specific commands

The station specific commmands affect the whole configuration of the logger or all channels of the logger. A station specific command is always embedded as a direct child element of a <StationCommands> element.

4.1.1.1. Logging command

The <Logging> command element as a direct child element of the <StationCommands> element is used to modify the data acquisition intervals of **ALL** channels of the logger. The same command can be applied as a channel specific command to modify the settings of specific channels only.

The <Logging> element has 3 attributes to modify the data acquisition interval.
The samplingInterval attribute defines the interval (in seconds) at which sample (=instantaneous) values are acquired from a sensor.
The storageInterval attribute defines at which interval (in seconds) storage values are stored in the database for this channel. A storage value is either the mean or total value of all instanteneous values that were acquired from a sensor within the storage interval.
The storageOffset attribute defines the offset (in seconds) relative to 00:00:00, where the storage interval starts.

Below is an example command message, where the data acquisition intervals are set the same for all channels.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <Logging ackId="1" samplingInterval="300"
                       storageInterval="3600"
                       storageOffset="0" />
  </StationCommands>
</CommandList>
```

The samplingInterval is set to 5 minutes, so every 5 minutes (300 seconds) a sample value is acquired.
The storageInterval is set to 1 hour (3600 seconds), so every hour the mean value (or total, depending on the configuration of the channel) of the 12 acquired instantaneous values since the last storage is calculated and stored in the database of this channel.
As the storageOffset attribute is set to 0 seconds, in this case the logger stores values at 00:00:00, 01:00:00, 02:00:00, 03:00:00 and so on.

If the storageOffset attribute is set to an offset of a half hour (1800 seconds), the logger would store values at 00:30:00, 01:30:00, 02:30:00, 03:30:00, and so on.

4.1.1.2. SetInternetConnection command

The OTT netDL1000 supports currently three different ways to provide internet access. The first is via a GPRS module, the second via Hayes compatible modem using PPP and the third via a LAN connection to a DSL router. Configuration of internet access is done in the operating software of the logger or via the SetInternetConnection command, described in this chapter.

Below is an example illustrating the use of this command:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
```

```
    <SetInternetConnection ackId="123" index="1">
      <LANConnection />
      <ProxyList>
        <HttpProxy address="220.150.22.0" port="8080" />
      </ProxyList>
    </SetInternetConnection>
    <SetInternetConnection ackId="124" index="2" name="GPRS Connection">
      <GPRSConnection apn="m2m.vodafone.de"
                      dialNo="*99#" username="dummy" password="1xRt*!" />
    </SetInternetConnection>
  </StationCommands>
</CommandList>
```

The <SetInternetConnection> element is a direct child of the <StationCommands> element. In OTT netDL1000 up to 2 different internet connections can be defined to support redundant communication paths for IP transmission (see also chapter “Redundant communication paths”).

Every internet connection is assigned a unique index (1 or 2), which is set using the index attribute of the <SetInternetConnection> element. This index is then referenced, when setting up an IP transmission, either in the operating software of the logger or by using the SetIPDataTransmission command (for details see the next subchapters).

In the example above, 2 different internet connections are defined: the first via a DSL router connected to the Ethernet interface of the logger and the second via a GPRS module connected to COM1 (=first RS-232 port of the logger).

If the logger is connected via LAN/Ethernet to a DSL router, no additional information is provided here, like in the example above with an empty <LANConnection> element.

For a GPRS connection, the attribute apn must be defined.
The attribute dialNo is typical "*99#" to dial in to the GPRS service. This is common in most countries, however it may be necessary to use a provider specific dialNo here.
The attributes username and password are optional, depending on the required authentication of the GPRS provider.

A PPP connection via a Hayes compatible modem can also be reconfigured using the GPRSConnection element. In this case the attribute apn is not specified. The other attributes are the same for a GPRS and a PPP connection.

Optionally a proxy can be used, when connection to a HTTP server is established.
Proxy Settings are defined within an optional <ProxyList> element. To define the HTTP Proxy settings, a <HttpProxy> element must be placed within the <ProxyList> element.
There are mandatory attributes address and port, as well as optional attributes username and password.

Please note that for an internet connection via the LAN/Ethernet interface additional settings are necessary, which are not part of the SetInternetConnection command. This is the use of a DHCP client for the automatic assignment of an IP address, as well as the subnetmask and the standard gateway settings. These settings are configured using the logger operating software.

4.1.1.3. SetHttpServer command

In a typical IP based measuring network the OTT netDL1000 datalogger communicates with a HTTP server for data transmission and remote control using command messages. To increase data availability on the server side, OTT netDL1000 supports up to two redundant HTTP servers (see also chapter “Redundant servers”) in one transmission. In total up to 8 different servers can be defined.

Using the SetHttpServer command different settings of these HTTP servers can be defined.

Below is a list of the different attributes of the SetHttpServer command.

Attribute	Meaning
index	Index of the HTTP server (1 to 8)
name	Plain text name of the server.
address	IP address of the server (without leading http://)
port	Port on which HTTP server is listening (default 80).
username	Optional username for HTTP basic authentication.
password	Optional password for HTTP basic authentication.
dataScript	Relative URL of server side data script to which data are sent in self-timed mode.
commandScript	Relative URL of server side command script which logger invokes to check for command messages to execute.
ackScript	Relative URL of server side acknowledge script which logger invokes to acknowledge executed commands.
alarmScript	Relative URL of server side alarm script which logger invokes to transmit alarm messages.
configurationScript	Relative URL of server side configuration script to which the proprietary configuration file is sent after the configuration has changed.
firmwareScript	Relative URL of server side firmware script to which the binary firmware file is sent after the firmware has changed.

Below is an example for the usage of the SetHttpServer command:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <SetHttpServer ackId="123"
                   index="1"
                   name="Primary HTTP Server"
                   address="testserver.ott.com"
                   username="dummy"
                   password="7sT%_#2Rst"
                   dataScript="data.php"
                   commandScript="command.php"
                   ackScript="ack.php"
                   alarmScript="alarm.php"
```

```
                    configurationScript="config.php"
                    firmwareScript="firmware.php" />
  </StationCommands>
</CommandList>
```

There are different server side scripts defined, which are invoked when the logger wants to perform a certain action.

When the logger invokes the individual scripts it passes additional information as URL parameters to the script. For every GET or POST request the URL parameter “stationid” and “action” are passed. The value of the “action” URL parameters depends on which script is invoked. Below is a list of the different values for “action”.

Server Side Script	Value of action
dataScript	senddata
commandScript	requestcommand
ackScript	acknowledge
alarmScript	sendalarm
configurationScript	sendconfiguration
firmwareScript	sendfirmware

As the action is explicitly passed as a URL parameter to the script, in principle one server side script e.g. doAll.php could handle all the different requests from the logger, e.g. receiving data and alarm messages, providing command messages, and so on.

For details on the usage of the `dataScript` attribute see chapter “Self-timed data transmission”.

For details on the usage of the `commandScript` and `ackScript` attributes see the chapters “Command messages” and “Acknowledgement of command messages”.

For details on the usage of the `configurationScript` attribute see the chapters “Configuration Update” and “SaveConfiguration command”.

For details on the usage of the `firmwareScript` attribute see the chapter “Firmware Update”.

For details on the usage of the `alarmScript` attribute see the chapter “Alarm messages”.

4.1.1.4. SetIPDataTransmission command

The OTT netDL1000 logger can be set up to transmit the measured data (+ metadata) in regular intervals (self-timed) to a central server. The setup of this self-timed data transmission can be done in the operating software of the logger or using the `<SetIPDataTransmission>` command element in a command file.

In principle, the setup of such a self-timed transmission consists of different logical parts:

- Timing: Definition of the transmission interval (+ optional time offset)

- Channels: Which channels shall be transmitted
- DataFormat: In which format are the data transmitted
- InternetConnection: How is the connection to the internet established
- Server: To which server (and with which protocol) shall the data be transmitted

Below is an example to set up an hourly self-timed transmission to a central HTTP server.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <SetIPDataTransmission ackId="123" index="1" name="Transmission1">
      <Timing interval="3600" offset="1800" />
      <Channels>
        <Channel channelId="0010" />
        <Channel channelId="0020" />
      </Channels>
      <DataFormat format="OTT-ML" />
      <InternetConnection index="1" />
      <Server index="1" redundantIndex="2" retries="1"  />
    </SetIPDataTransmission>
  </StationCommands>
</CommandList>
```

The `<SetIPDataTransmission>` element is a direct child element of the `<StationCommands>` element and has a mandatory `index` attribute, which is used to identify different transmission configurations (up to 8).

The `<SetIPDataTransmission>` element has itself different child elements to define the different logical parts of such a configuration. The mandatory `<Timing>` element defines at which interval the messages are sent. In the example, messages are sent every hour (3600 seconds) with an ½ hour offset (1800 seconds) due to the attributes `interval="3600"` and `offset="1800"`. So data messages will be sent at 00:30:00, 01:30:00, 02:30:00, ...

The optional `<Channels>` element is used to define, which channels shall be included in the data message. For every channel to be included, a `<Channel>` element must be added with the required attribute `channelId` identifying the channel. If the `<Channels>` element is missing, then all channels will be included in the data message.

The optional `<DataFormat>` element is used to define the data format of the messages, that are transmitted to the server. Currently the only format that is supported is the `format="OTT-ML"`, which is the XML format described in detail in the chapter “Format of data time series messages”.

The optional `<InternetConnection>` element is used to define which internet connection shall be used for establishing a connection to the server. The connection itself is configured using the operating program of the logger or the command SetInternetConnection. Different internet connections can be configured, e.g. one which uses a GPRS module on COM1 and another which uses a DSL router on the Ethernet interface.

Every configured internet connection is assigned a unique index (see also the chapter “SetInternetConnection command”). This index is then referenced in the `index` attribute of

the <InternetConnection> element. If the <InternetConnection> element is missing, then index 1 is assumed.

An optional redundant internet connection for this transmission configuration could be defined using the redundantIndex attribute (see also the chapter “Redundant communication paths”).

The optional <Server> element is used to define to which server the data messages shall be transmitted. Using the logger operating program or the SetHttpServer command, different servers can be configured, where each server gets a unique index number. This index is then referenced in the index attribute of the <Server> element.

An optional redundant server for this transmission configuration could be defined using the redundantIndex attribute (see also the chapter “Redundant servers”). If the <Server> element is not present, the server with index 1 is implicitly used for this configuration.

If a redundant server is specified, an optional attribute retries can be used to define, how many retries are made on the first server, before the second server is used.

Typically a self-timed transmission is transmitting at fixed intervals. By using the limit module connected with an alarm action however it is possible to change the transmission interval automatically depending on the values of specific channels (see also chapter “Alarm actions”). For a self-timed transmission up to 5 alarm intervals can be defined. For every limit module the corresponding alarm interval can be defined in the logger operating software.

If the actual measured value of a station then crosses this limit, the alarm action can be used to switch the transmission interval of a self-timed transmission configuration from the standard interval to one of the 5 alarm intervals.

Below is an example, that illustrates how to define an alarm interval for a self-timed transmission.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <SetIPDataTransmission index="1">
      <Timing interval="3600" alarmInterval1="900"  />
    </SetIPDataTransmission>
  </StationCommands>
</CommandList>
```

The alarmInterval1 attribute of the <Timing> element is used to set the first alarm interval to 15 minutes (=900 secondes). The connection logic, that this alarm interval is used, when a certain limit is crossed, must be explicitly configured in the limit module and the alarm management in the logger operation software. The <SetIPDataTransmission> element can then be used to modify standard and/or alarm intervals.

4.1.1.5. SetMaintenanceWindow command

The SetMaintenanceWindow command can be used to define a maintenance window, in which the communication devices of the logger are powered and the logger waits for connection requests. For details on maintenance windows see chapter the “Maintenance windows”.

Maintenance windows can either be defined as part of the configuration, i.e. the maintenance window is opened every day at configurable times. Maintenance windows can also be opened immediately for a certain duration. In this case the logger does not store the maintenance window settings.

For configuring maintenance windows, the logger operating software or the `<SetMaintenanceWindow>` command element is used.

Below is a first example of its usage:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
   <StationCommands stationID="0123456789">
    <SetMaintenanceWindow index="1" deviceIndex="1" callerId1="745324" >
      <Timing start="12:30:00" duration="1800" />
      <ConnectMode>
        <AcceptCall />
      </ConnectMode>
    </SetMaintenanceWindow>
  </StationCommands>
</CommandList>
```

The `<SetMaintenanceWindow>` element is a direct child element of the `<StationCommands>` element. The mandatory `index` attribute is used to identify which maintenance window configuration shall be created of modified. Up to 8 different configurations can be defined (`index="1-8"`).

If the `index` is 0, no configuration is created or modified, but a maintenance window is opened once for the specified duration. The `start` attribute is in this case optional. If it is missing, the maintenace window starts immediately.

Using the `deviceIndex` attribute, the device can be selected, which is used for the maintenance window. For a maintenance window with `index`=0 (see below), the `deviceIndex` attribute is mandatory. For maintenance windows of type “Accept IP Connection, the `deviceIndex` attribute is ignored.

The attributes `callerId1` to `callerId4` can be used to restrict the access to the logger to certain phone numbers or IP addresses.

The <Timing> element as a child element of the `<SetMaintenanceWindow>` element is used to define the start time and the duration (in seconds) of a maintenance window. A maintenance window configuration can have up to 4 <Timing> elements, to define up to 4 time ranges within a day, where a maintenance window is active.

If a maintenance window shall be opened immediately after executing the command, the `index` attribute of the `<SetMaintenanceWindow>` element must be set to 0 and the `start` attribute of the element <Timing> is omitted, so only the `duration` attribute is then specified.

Finally for a maintenance window, a `<ConnectMode>` element is necessary, to define how an incoming call/connection shall be handled.

There are 3 different modes available, which are explained in detail in the chapter "Maintenance windows".

In the example above, the simplest mode is activated, by embedding an empty `<AcceptCall>` element in the `<ConnectMode>` element. In this mode, incoming calls are answered making use of the OTT-protocol (for configuration and data polling).

Below is an example of creating a maintenance window with a more complex behaviour:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationID="0123456789">
    <SetMaintenanceWindow index="1" deviceIndex="1" callerId1="745324" >
      <Timing start="12:30:00" duration="1800" />
      <ConnectMode>
        <ConnectToServer ipTransmissionIndex="1" listenMode ="RING" />
      </ConnectMode>
    </SetMaintenanceWindow>
  </StationCommands>
</CommandList>
```

The settings are the same as in the first example, except for the connect mode, which is now set to `<ConnectToServer>`. In this case the logger does not accept incoming calls, but connects in response to a call or SMS to an internet server, transmits data and checks if a command message is available for execution. The attribute `listenMode` defines, whether the logger reacts to incoming calls (`listenMode ="RING"`) or incoming calls and SMS (`listenMode ="RING+SMS"`). If no `listenMode` attribute is specified, the logger uses implicitly the default value `RING`.

The information which internet connection shall be used for connecting and to which server shall be transmitted is provided by the optional attribute `ipTransmissionIndex`. This index refers to the index of an IP data transmission configuration, which can be defined used the operating software or the SetIPDataTransmission command. If this attribute is missing, a default index of 1 is assumed.

The last example illustrates the configuration of an immediate maintenance window with the third mode.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationID="0123456789">
    <SetMaintenanceWindow index="0" deviceIndex="1" >
      <Timing duration="1800" />
```

```
      <ConnectMode>
        <AcceptIP
            connectionIndex="1"
            port="85"
            publishURL="http://user:pass@members.dyndns.org/nic/update?host=abc"/>
      </ConnectMode>
    </SetMaintenanceWindow>
  </StationCommands>
</CommandList>
```

As the `index` attribute of the `<SetMaintenanceWindow>` element is set to 0 and the `start` attribute of the `<Timing>` element is not specified, the maintenance window will be activated immediately after execution of this command.

The connect mode `<AcceptIP>` instructs the logger to listen on port 80 (HTTP) for incoming connection requests. This mode is intended to connect to the logger via IP communication with the operating software using a proprietary format. The attribute `connectionIndex` defines, which internet connection is used. If this attribute is missing, the first connection is used. Optionally a different listening port can be defined using the `port` attribute.

If the logger cannot be connected on a fixed IP address, the `publishURL` attribute can be used to instruct the logger to publish its IP address, when it establishes a connection with the internet. A service like DynDNS can then be used to access the logger using a fixed symbolic name, which is then resolved dynamically to the actual IP address. The IP address can also be published to the central server, so the address is known to the central server only. For details on this publishing mechanism see the chapter “Accepting IP connections”. The attribute `publishURL` must contain the complete URL, that is sent to the address provider (without the current IP address, which is added by the logger).

The configuration of the publishing mechanism for dynamic IP addresses is done via the logger operating software.

4.1.1.6. UpdateConfiguration command

The `<UpdateConfiguration>` command element is used to instruct the logger, that it shall download a new configuration file from the URL specified by the mandatory `serverIndex` and `configurationFile` attributes.

The configuration file must be a file on the HTTP server referenced by `serverIndex`, which is then downloaded by the logger via an HTTP GET request. The `configurationFile` attribute includes the full path on the server.

The returned configuration file must be a binary file in the OTT specific configuration file format.

Below is an example of this command:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <UpdateConfiguration serverIndex="1"
                         configurationFile="/configs/config0123456789.bin"
                         ackId="123" />
  </StationCommands>
</CommandList>
```

The `configurationFile` attribute can also reference a dynamic script, which returns the binary configuration file. The HTTP GET request is always performed with the URL parameters “stationId=...” and “action=getconfiguration”.

For the example above, the request URL for the configuration file would be:

http://server1/configs/config0123456789.bin?stationid=0123456789&action=getconfiguration

where server1 is the address of the server corresponding to serverIndex 1.

After the logger has downloaded the configuration file, the command message is acknowledged, the new configuration is installed, the logger is restarted and the database is erased.

For details on this sequence, see the chapter “Diagram: Process UpdateConfiguration command” in Appendix A.

4.1.1.7. SaveConfiguration command

The `<SaveConfiguration>` command element is used to instruct the logger, that it shall upload its current configuration in proprietary binary format to a HTTP server.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <SaveConfiguration serverIndex="1" ackId="1" />
  </StationCommands>
</CommandList>
```

The HTTP server, to which the upload shall be performed is defined by the `serverIndex` attribute. Different server configuration can be defined in the logger operating software or by using the command SetHttpServer. Every server configuration has a unqiue index. If the attribute `serverIndex` is missing, index 1 is implicitly used.

The configuration file is uploaded to the server side script of the HTTP server defined by the `configurationScript` attribute of the SetHttpServer command (see chapter “SetHttpServer command”) or in the operating program of the logger.

The configuration file is uploaded in an HTTP POST request to this script using the content type “application/octet-stream”. The id of the station is passed to the server side script using an URL-parameter `stationid`. The URL parameter “action=sendconfiguration” indicates the configuration upload.

In response to the POST request with the file upload, the server returns an acknowledge message as defined by XML schema file OTT_Response.xsd. For details on this response format see also chapter “Acknowledgement of command messages”.

Note that the logger does not send an explicit acknowledgement of the command message to the server. Instead the configuration file is uploaded, which means an implicit acknowledgement.

For details on this sequence see chapter “Diagram: Process SaveConfiguration command”.

4.1.1.8. UpdateFirmware command

The `<UpdateFirmware>` command element is used to instruct the logger, that it shall download a new firmware file from the URL specified by the mandatory `serverIndex` and `firmwareFile` attributes.

The firmware file must be a file on the HTTP server referenced by `serverIndex`, which is then downloaded by the logger via an HTTP GET request. The `firmwareFile` attribute includes the full path on the server.

The returned firmware file must be a binary file in the OTT specific firmware file format.

Below is an example of this command:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <UpdateFirmware serverIndex="1"
                    firmwareFile="/firmware/sli01mv1.10.0.bin"
                    ackId="123" />
  </StationCommands>
</CommandList>
```

The `firmwareFile` attribute can also reference a dynamic script, which returns the binary firmware file. The HTTP GET request is always performed with the URL parameters “stationId=...” and “action=getfirmware”.

For the example above, the request URL for the firmware file would be:

http://server1/firmware/sli01mv1.10.0.bin?stationid=0123456789&action=getfirmware

where server1 is the address of the server corresponding to serverIndex 1.

After the logger has downloaded the firmware file, the command message is acknowledged, the new firmware is installed and the logger is restarted.

For details on this sequence see chapter “Diagram: Process UpdateFirmware command” in Appendix A.

4.1.1.9. SetStatusOutput

The <SetStatusOutput> command element is used to set an status output to either ON or OFF.

In the example below the status output corresponding to (actuator) channel “0100”, which can be used to switch ON or OFF a device is set to “ON”.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <SetStatusOutput channelId="0100" value="ON" />
  </StationCommands>
</CommandList>
```

4.1.2. Channel specific commands

The channel specific commmands affect the configuration of a specific channel of the logger. A channel specific command is always embedded as a child element of a `<ChannelCommands>` element. Every `<ChannelCommands>` element has a mandatory attribute `channelID` to identify the channel.

4.1.2.1. Logging command

The `<Logging>` command element as a direct child element of the `<ChannelCommands>` element is used to modify the data acquisition intervals of one specific channel of the logger. The same command can be applied as a station specific command to modify the settings of **all** channels.

The details of this command are described in the corresponding chapter on station specific commands.

Below is an example illustrating the structure of the logging command for a specific channel.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <ChannelCommands channelId="0010">
      <Logging ackId="1" samplingInterval="300"
                         storageInterval="3600"
                         storageOffset="0" />
    </ChannelCommands>
  </StationCommands>
</CommandList>
```

4.1.2.2. Scaling command

The `<Scaling>` command element is used to modify the linear transformation settings for a specific channel.

Below is a simple example to illustrate this.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <ChannelCommands channelId="0010">
      <Scaling ackId="1" factor="0.1" offset="2" />
    </ChannelCommands>
  </StationCommands>
</CommandList>
```

After the execution of this command, all values of this channel will be transformed according to this formula:

Y = factor * X + Offset
where X is the original value and Y the transformed value.

Note that this command only has an effect, when the specified channel contains a module for a linear transformation.

4.1.2.3. Limit command

The `<Limit>` command element is used to define lower and upper limits for a specific channel. Depending on the configuration of the logger, alarm messages can be sent, when such a limit is crossed, or e.g. a relais can be switched.

Below is a simple example to illustrate the use of this command.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <ChannelCommands channelId="0010">
      <Limit ackId="1" upper="100" lower="20" />
    </ChannelCommands>
  </StationCommands>
</CommandList>
```

In the above example, the upper limit of channel “0010” is set to a value of 100 and the lower limit is set to a value of 20. Note that every limit module in the logger configuration defines either a lower or an upper limit. So for the above example, the channel needs to have 2 limit modules, one defining an upper limit and another one for a lower limit.

As a channel can have multiple limit modules, there is a mechanism necessary to identify which limit shall be modified. This is accomplished using the optional `index` attribute. If this attribute is not set, the first upper or lower limit is modified. If it is set, the n-th limit is modified, where n = the value of the `index` attribute (Indexing starts with 1).

So to set the 2nd upper limit to 200, the command message below is used:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
```

```
  <StationCommands stationId="0123456789">
    <ChannelCommands channelId="0010">
      <Limit ackId="1" upper="200" index="2" />
    </ChannelCommands>
  </StationCommands>
</CommandList>
```

Using the limit command, only the values of the upper and lower limits can be adjusted. To modify the details of the limit module, like hysteresis or the action that is taken, when the limit is crossed, the logger operating program has to be used.

4.1.2.4. Gradient command

The `<Gradient>` command element is used to define gradient checks for rising or falling values or for both directions for a specific channel.

Depending on the configuration of the logger, alarm messages can be sent, when such a gradient is exceeded.

Below is a simple example to illustrate the use of this command.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <ChannelCommands channelId="0010">
      <Gradient ackId="1" rising="100" />
    </ChannelCommands>
  </StationCommands>
</CommandList>
```

The value for the check for rising gradients is set to 100 for channel “0010”.

Like in the limit command, the optional attribute `index` can be used to select for which gradient module, the values shall be adjusted.

Note that gradient is only available as a sub option of the limit module, so to define gradients, limit modules must be configured in the channel. For details see the operating manual for the logger.

To set the value for the gradient check on falling values of the 2nd module, the command message below is used:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <ChannelCommands channelId="0010">
      <Gradient ackId="1" falling="150" index="2" />
    </ChannelCommands>
  </StationCommands>
</CommandList>
```

Gradient values for falling and rising values can be defined separately. In this case 2 limit modules with enabled gradient check must be used.

To use one gradient value for rising and falling values, the `both` attribute is used:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CommandList>
  <StationCommands stationId="0123456789">
    <ChannelCommands channelId="0010">
      <Gradient ackId="1" both="100" />
    </ChannelCommands>
  </StationCommands>
</CommandList>
```

4.2. Acknowledgement of command messages

When the logger has received and executed a command message, an acknowledge message is sent to the server to inform about the successful execution of the commands contained in the message or in case of an error to return an error message.

The file OTT_Response.xsd contains the XML Schema of an acknowledge message. This chapter describes the format of these messages including examples.

The acknowledge message is sent in the body of a HTTP POST message to an acknowledge script on the web server, from where the command file was fetched. The URL of the acknowledge script is defined by the attribute `ackScript` of the SetHttpServer command.

When the acknowledge script is invoked, the ID of the station is passed as a URL parameter “stationid” and the additional URL parameter “action=acknowledge” indicates the acknowledgement of a command message.
Below is an example of a simple acknowledge message acknowledging a single command.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Response stationId="0123456789" commandDateTime="2009-01-15T17:10:20"
responseDateTime="2009-01-15T17:10:25">
  <Ack ackId="123" status="OK" />
</Response>
```

The root element of an acknowledge message is the `<Response>` element, which has the `stationId` as an optional attribute. Additional optional attributes are the timestamp of the command message, which is acknowledged and the timestamp when this message was sent.

For every message, that needs to be acknowledged, an `<Ack>` element is included as a child element to the `<Response>` element. The `<Ack>` element has a manadatory attribute `status` to indicate if the command was executed successfully (`status="OK"`) or not (`status="FAIL"`). In case of a failure, the element contains a plain text error message. If the command was executed successfully, the element is empty.

If the command that is acknowledged, had an `ackID` attribute in the command message, then this ackID is repeated in the `ackID` attribute to provide a clear reference to the acknowledged

command. Multiple commands can be acknowledged in one acknowledge message, as illustrated below:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Response stationId="0123456789" commandDateTime="2009-01-15T17:10:20"
responseDateTime="2009-01-15T17:10:25">
  <Ack ackId="123" status="OK" />
  <Ack ackId="124" status="FAIL">Invalid time period</Ack>
  <Ack ackId="125" status="OK" />
</Response>
```

In the example above, three commands are acknowledged, where the second command was not successful.
The acknowledge message itself must be acknowledged with an acknowledge message. The sequence diagram “Diagram: Processing general command messages” illustrates this.

5. Querying data from the logger

A special case of a command message is a query message, which is used to query the data logger for specific data. As already explained in the chapter “Data Types”, the logger manages these different kinds of data, which can be queried for:

- storage values of the different channels
- instantaneous values of the diffferent channels
- different types of channel specific events, that are generated by the data logger
- different types of station specific events, that are generated by the data logger

To query the logger in offline mode, a query message instead of a command message must be returned from the web server to the logger, when the data logger checks for a command message.

See chapter “Format of data time series messages” for a documentation about the structure of the messages that are sent in response to query messages.

5.1. Basic structure of a query message

The file OTT_Query.xsd contains the XML Schema of a query message. This chapter describes the format of query messages including examples.

Below is a minimalistic example to get started with the structure of a query:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<QueryList>
  <Query start="2009-01-01" end="2009-02-01" ackId="123">
    <QueryStationEvents />
    <QueryValues />
    <QueryEvents />
  </Query>
</QueryList>
```

The root element of a query message is a `<QueryList>` element instead of the `<CommandList>` root element of a command message. A `<QueryList>` element contains a list of `<Query>` elements, where each `<Query>` element contains a query for data of one measuring station. OTT netDL1000 only evaluates the first `<Query>` element within the `<QueryList>` element. For data servers however, that manage data of multiple stations, different queries can be combined in one query message.

Using the `start` and `end` attributes of the `<Query>` element, the time range can be specified, for which data shall be returned. If the `start` attribute is missing, the returned data start with the oldest data available. If the `end` attribute is missing, the returned data end with the latest data available. In consequence if both elements are missing, all available data are returned. Please keep in mind, that this might result in huge return messages. An optional `ackId` attribute can be specified for the query. When this attribute is set, the corresponding response will contain the same `ackId` as an attribute of the `<StationData>` element.

In the example above, data would be returned from January 1st, 2009 until February 1st, 2009. In the example the timerange was specified in full days. For the `start` and `end` attributes however date and time can be specified, to allow for precise queries down to a temporal resolution of seconds. The start of the specified interval is exclusive of the timerange, the end is inclusive.

Using the `<QueryStationEvents>`,`<QueryValues>`,and `<QueryEvents>` elements, it can be specified, which kind of data shall be returned. In the table below all elements to define the returned data are listed:

Element	Data Type
`<QueryValues>`	storage values of the channels
`<QueryStationEvents>`	station specific events
`<QueryEvents>`	channel specific events
`<QueryInstValues>`	instantaneous values of the channels

Direct child elements of the `<Query>` element are affecting all channels.
In the above example thus the station specific events, the channel specific events for all channels and the storage values of all channels are requested.

The query format allows to define generic settings for the query at a global level (as direct child elements of the `<Query>` element), which affect all channels.

5.2. Refining queries for individual channels

Below is a more complex example illustrating how to define which channels shall be queried and the meaning of some additional elements:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<QueryList>
  <Query start="2009-01-01" end="2009-02-01">
    <QueryStationEvents />
    <QueryValues />
    <QueryChannel channelId="0010" />
    <QueryChannel channelId="0020" />
  </Query>
</QueryList>
```

In the example above, the time range is defined from January 1st, 2009 until February 1st, 2009, which applies to all channels.

The `<QueryStationEvents>` element specifies that all station specific events in the specified time range shall be included in the response.

The `<QueryValues>` element specifies that storage values for all channels shall be included in the response.

The <QueryChannel> elements are then used to define which channels shall be included in the response. In the example above, data for 2 channels identified by the channelId attributes shall be returned (“0010” and “0020”).

5.3. Querying for instantaneous values

To query the instantaneous values of the channels of a data logger, the <QueryInstValues> element is used within the <Query> element, as illustrated in the example below.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<QueryList>
  <Query>
    <QueryInstValues />
  </Query>
</QueryList>
```

In the above example the instantaneous values of all channels of a logger are queried for. As can be seen in the example, no timerange has to be specified for querying instantaneous values.

To query the instantaneous values of specific channels, the channels must be explicitly defined using the <QueryChannel> elements, as illustrated in the example below.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<QueryList>
  <Query>
    <QueryInstValues />
    <QueryChannel channelId="0010" />
    <QueryChannel channelId="0020" />
  </Query>
</QueryList>
```

The format of the response message for instantaneous values is described in the chapter “Data messages for instantaneous values”.

5.4. Addressing a specific station on a data server

When a query message is directly addressed to the data logger, the logger will respond with his data, so it is not necessary to explicitly identify the station, for which data are requested.

However the same message format could be used to query a (web) server, that has access to a database of different loggers. In this case a unique identification of the station is necessary.

This is shown in the example below:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<QueryList>
  <Query workspaceId="1" stationId="0123456789"
```

```
        start="2009-01-01" end="2009-02-01">
    <QueryValues />
  </Query>
</QueryList>
```

The attributes `workspaceId` and `stationId` are used to define the station, from which data shall be queried. These 2 attributes are optional and only necessary, when the recipient of the query message can provide data from different stations.

In this case at least the `stationId` is necessary. In bigger networks, there might be multiple stations with the same station ID. For such cases, the additional `workspaceId` is used to logically structure different loggers in so called workspaces. In this case the combination of `workspaceId` and `stationId` must be unique for every station to unambiguously identify a station.

5.5. Refining queries for individual events

Using the <QueryStationEvents> and <QueryEvents> elements, station and channel specific events can be explicitly included in the query.

If such an element is included as an empty tag, then **all** station or channel events shall be included in the response:

To define a filter, that only certain events are delivered, the <QueryStationEvents> and <QueryEvents> elements can be refined by embedding <Include> and <Exclude> elements, as shown in the example below:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<QueryList>
  <Query>
    <QueryStationEvents>
      <Include>error.communcation</Include>
    </QueryStationEvents>
    <QueryEvents>
      <Include>limit</Include>
      <Include>gradient</Include>
    </QueryEvents>
    <QueryChannel channelId="0010" />
    <QueryChannel channelId="0020" />
  </Query>
</QueryList>
```

In the example the <QueryStationEvents> element contains one <Include> element. If such elements are present within a <QueryStationEvents> or <QueryEvents> element, it means that only events of types, that are explicitly included in an <Include> element will be included in the response.

In the above example only events of type “error.communication” are included, which means that only error events will be included in the response, that occured during a communication operation. Note that event types can be defined hierarchically. The type “error.communication” is a subtype of the generic type “error”. Further below is a list of the different event types supported in the OTT netDL1000.

In the above example the events for channels “0010” and “0020” are restricted to “limit” and “gradient” events, which occur when an upper or lower limit is crossed, or when a gradient is exceeded.

5.5.1. Event Types

Below is a list of station specific events, which are supported in the OTT netDL1000:

Eventtype	Meaning
error.communication	any kind of error that occured during a communication

info.settime	the internal time of the logger was changed
info.observer	an observer has operated the logger manually in the field

Below is a list of channel specific events, which are supported in the OTT netDL1000:

Eventtype	Meaning
info.limit	a limit was crossed
info.gradient	a gradient was exceeded
info.statusbit	a statusbit of a status sensor has changed its state
info.observer	an observer has manually edited a value for a channel

5.6. Including information about the station

When querying data from a station, certain information about the station can be included in the response (e.g. firmware version, battery level, ...).

To activate the inclusion of this information in the response, the `includeStationInfo` attribute of the `<Query>` element has to be set to `"1"`, as in the example below:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<QueryList>
  <Query start="2009-01-01" end="2009-02-01" includeStationInfo="1">
    <QueryValues />
  </Query>
</QueryList>
```

The actual station information is then included in the response in the `<StationInfo>` element. See also the chapter “Data messages including additional station information” for details.

5.7. Requesting data in compressed format

As with all XML based formats, the response to a query in XML format tends to be rather verbose. To reduce the message size, an optional compression on the XML format can be requested. To activate compression of the response message, the `compression` attribute is used. Currently the compression method “deflate” is supported.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<QueryList>
  <Query compression="deflate">
    <QueryValues />
  </Query>
</QueryList>
```

For deflate compressed content the “Content-Encoding” attribute in the HTTP header is set to “deflate”.

6. Logger operation use cases

In this chapter different use cases for the operation of the logger in an IP network are explained.

6.1. Self-timed data transmission

In modern IP measurement networks, the data logger typically transmits the measured data (+metadata) in regular intervals (=self-timed) to a central server.

The data are transmitted as XML data messages using the HTTP POST method. The schema file for the XML data messages is the file “OTT_Data.xsd”. For a detailed description of the XML data message format see chapter “Format of data time series messages”.

Setting up a self-timed transmission can be done in the operating software of the logger or by using the SetIPDataTransmission command (see chapter “SetIPDataTransmission command”). This command can be used for example to modify the transmission intervals explicitly.

A configuration of a self-timed transmission consists of different parts, including how to establish the internet connection, what content to transmit and to what destination server and with which protocol the data shall be sent.

Each time data shall be transferred to the server, the logger invokes the server side script of the HTTP server defined by the `dataScript` attribute of the SetHttpServer command (see chapter “SetHttpServer command”) or in the operating program of the logger. The URL parameter “stationid” identifies the transmitting station and the URL parameter “action=senddata” indicates to the script that a data transmission is performed.

In response to the HTTP POST request containing the data, the server returns an acknowledge message as defined by the XML schema file OTT_Response.xsd. (see also the sequence diagram “Diagram: Self-timed data transmission” in Appendix A).

6.1.1. Automatic adjustment of transmission interval

By defining different limits for a channel and assigning them so called limit cycles (or intervals), the transmission interval can be automatically adjusted from the logger, so that e.g. in case of critical water levels the automatic data transmission is performed more frequent than under normal conditions (see also next chapter “Alarm actions”).
Using the SetIPDataTransmission command, the standard and alarm intervals can be adjusted explicitly.

6.1.2. Redundant communication paths

OTT netDL1000 supports 2 independent communication paths for IP based self-timed transmission of data. Data could be transmitted for example on the one hand via a GPRS connection and on the other hand via a DSL connection. This allows for high data availability even in the case of an interruption in one of the networks. E.g. when the cable of a ADSL connection was cut for some reason, the GPRS connection will typically not be affected by this problem and still provide internet connectivity.

A redundant communication path is either defined in the logger configuration software or via the `redundantIndex` attribute of the `<InternetConnection>` child element of an SetIPDataTransmission command.

For details see also the chapters “SetInternetConnection command” and “SetIPDataTransmission command”.

6.1.3. Redundant servers

However even with a high availability of the internet connection due to redundant communication paths, there is still a single point of failure if only one central server is used to receive and process the data.

To circumvent this potential single point of failure, OTT netDL1000 supports the use of redundant servers. So if one server is no longer available, data can still be transmitted to a second server to optimize data availability.

A redundant server is either defined in the logger configuration software or via the `redundantIndex` attribute of the `<Server>` child element of an SetIPDataTransmission command.

For details see the chapters “SetHttpServer command” and “SetIPDataTransmission command”.

6.2. Alarm actions

OTT netDL1000 can be configured to perform certain alarm actions in the case that certain events occured. These events include crossing configurable limits or gradients of a channel and status alarms, e.g. to detect when the door to the station is openend.

There are different actions that can be taken, e.g. switching a device (e.g. signal horn), sending an alarm message (via SMS, Mail, HTTP, ...) or adjusting the transmission interval of a self-timed data transmission. For a self-timed transmission in addition to the standard transmission interval, up to 5 additional alarm intervals can be defined using the operation software or the SetIPDataTransmission command. In the alarm management module of the logger operation software, the alarm actions can be defined and connected to the individual modules generating the events.

Using the SetIPDataTransmission command, the different transmission intervals can be adjusted during operation of the logger (see also chapter “Automatic adjustment of transmission interval”).

6.2.1. Alarm messages

Probably the most common alarm action is to send an alarm message, when a certain event occurs. OTT netDL1000 can send alarm messages as a SMS, e-Mail or can send it to an HTTP server. In this chapter only the transmission to an HTTP server is explained. For details on the other messaging methods see the logger manual.

Each time an alarm message shall be sent to the HTTP server, the logger invokes the server side script of the HTTP server defined by the `alarmScript` attribute of the SetHttpServer command (see chapter “SetHttpServer command”) or in the operating program of the logger. The URL parameter “stationid” identifies the transmitting station and the URL parameter “action=sendalarm” indicates to the script that an alarm message is transmitted.

The actual alarm message is contained in the body of the POST request as a simple XML message described below. The file OTT_Alarm.xsd contains the XML Schema of an alarm message.

Below is an example of such a message:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<AlarmMessage stationId="0123456789" channelId="0010"
              t="2009-02-01T17:30:00" type="limit">
Value 10.1 has exceeded limit (10)
</AlarmMessage>
```

There is only one element `<AlarmMessage>` in an XML alarm message from the logger.
The mandatory attribute `stationId` identifies the station that has sent the alarm message. If the alarm message is channel specific, the attribute `channelId` identifies the channel which has caused the event that triggered transmission of the alarm message.
The mandatory attribute `t` contains the timestamp, when the alarm condition occurred.
The mandatory `type` attribute identifies which kind of event has triggered the alarm message. For a list of these event types see chapter “Event Types”.

Note that the event, that has triggered the transmission of an alarm message is also logged in the event log of the data logger. If events are included in the data that are transmitted in a self-timed transmission to a server, this event will also be transmitted in the next regular self-timed transmission as part of the transmitted time-series data. See chapter “Data messages including events” for a description of this format. If events are not included in the self-timed messages, they could also be explicitly queried for by using a query command. For details see chapter “

Querying data from the logger".

See also the sequence diagram "Diagram: Sending alarm messages" in Appendix A, that illustrates the alarm message transmission to an HTTP server.

6.3. Modifying channel specific settings

The configuration of individual channels can be setup and modified using the logger operating software. Using the channel specific commands, described in detail in chapter "Channel specific commands", settings of individual channels, like the sampling and storage interval can be adjusted specifically for every channel or with one command for the whole station.

Additionally the linear transformation of the measured values can be defined, as well as limits and gradients, that are used for checking for alarm conditions.

6.4. Firmware Update

There are different ways how to update the logger firmware. The first is a manual installation in the field via RS-232 interface. There is also the option to perform a remote update using the operating software via a phone connection to the logger. Finally the logger can be instructed to download and install a new firmware using the UpdateFirmware command.

The logger can be configured to upload a new firmware automatically to a server, when the firmware was updated locally or remotely using the logger operating software. In this case, the logger will contact a configured server and transmit the firmware to a dynamic script, defined by the `firmwareScript` attribute of the SetHttpServer command or in the logger operating software.

The binary firmware file is uploaded in an HTTP POST request to this script using the content type "application/octet-stream". The ID of the station is passed to the server side script using an URL-parameter `stationid`.

The server side script must return as a result of the HTTP POST request an acknowledgement message according to XML schema "OTT_Response.xsd".

6.5. Configuration Update

There are different ways how to modify the logger configuration. The first is a manual modification using the logger operating software in the field via RS-232 interface. There is also the option to perform a remote configuration using the operating software via a phone connection to the logger. Finally the logger can be instructed to download and apply a new configuration using the UpdateConfiguration command.

The logger can be configured to upload a new configuration automatically to a server, when the configuration was updated locally or remotely using the logger operating software. In this case, the logger will contact a configured server and transmit the configuration file to a

dynamic script, defined by the `configurationScript` attribute of the SetHttpServer command or in the logger operating software.

The binary configuration file is uploaded in a HTTP POST request to this script using the content type “application/octet-stream”. The id of the station is passed to the server side script using an URL-parameter `stationid`.

The server side script must return as a result of the HTTP POST request an acknowledge message according to XML schema “OTT_Response.xsd”.

6.6. Time synchronisation

The data logger can be configured using the operating software, to synchronize its internal clock automatically using the Network Time Protocol (NTP). For this purpose up to 3 NTP servers can be defined, which are contacted in regular intervals to avoid a drift of the internal clock.

7. Logger communication wakeup

Normally the data logger is configured in such a way, that connected communication devices are not switched on all day long but are only powered on demand, i.e. when a communication operation shall be performed. By this kind of power management, the average power consumption of the logger and communication system can be reduced effectively.

For a correct configuration of the logger, it is necessary to understand the different cases in which a station wakes up, powers its communication devices and establishes a communication or listens for incoming communication requests.

Below is a list of different wakeup cases:

- Self-timed data transmission to server
- An alarm condition occured (limit crossed, gradient exceeded or status alarm)
- Maintenance window

For details to the self-timed data transmission, see chapter “Self-timed data transmission” and chapter “SetIPDataTransmission command”.

For details to the handling of alarm conditions see the logger manual and the chapters “Limit command” and “Gradient command”.

For details on the different cases of maintenance windows, see the next chapter.

7.1. Maintenance windows

As the communication devices of the data logger are typically not powered all the time, it is normally not possible to establish an ad-hoc connection to the logger in order to query for data or issue commands, e.g. to change the configuration.

Often in IP communication environments, there is no direct way to establish a connection to the logger, because communication devices might not be powered continuously and if they are powered, the logger (rsp. router, to which the logger is connected) might have a dynamically assigned IP address, which is initially not known.

For these kinds of communication environments, the data logger can be configured in such a way, that all kinds of communication are initiated by the logger (=client) and the communication only takes place between logger and one (or more) servers. In these scenarios, the logger will send its data by self-timed transmissions (e.g. hourly) and check after the transmission, if there is a command message available for it, which shall be executed.

Using these command messages, the most common logger options can be modified or firmware and configuration updates can be initiated. In this case, the command messages are not executed immediately, i.e. when they are created on the server either manually by a human operator, or a software application, that automatically generates command messages to control a logger network. However the generated command messages will be executed asynchronously, when the logger connects to the server and checks for a message. This scenario works completely without the need to establish a connection to the logger and is very well suited for IP communication with dynamic IP addresses.

However in some cases it might be required to be able to establish a direct connection to the logger from the office. For these cases, so called maintenance windows can be defined, in which a direct communication to the logger can be established in different ways. Configuration of maintenance windows can be done via the logger operating software or with the SetMaintenanceWindow command of XML command messages.

7.1.1. Modes for handling connection requests

Each maintenance window is defined by different parameters. These include start and end (or duration) of the maintenance window, the identification of the switching output, that is switched on during the window and a mode, which defines how connection requests are handled.

In the next chapters, the different modes are explained.

7.1.1.1. Accept incoming phone calls

This is the “classical” mode for a maintenance window, where the logger is connected to a public telephone network (either via analog/digital landline or GSM/GPRS). At the begin of the maintenance window, the communication device is powered on by the logger and remains powered for the duration of the window. When a RING indicates a connection request, the logger accepts this connection and is available for “classic” communication via OTT protocol

with either the logger operating software for modification of the configuration or a firmware update or for polling data e.g. by the application software Hydras 3.

7.1.1.2. Connect to Server on RING or SMS

If the logger is connected to a GSM/GPRS module, then this mode can be used to establish a connection to a web server, when a RING indicates a connection request via GSM. In this mode, the GSM/GPRS module is switched on at the begin of the maintenance window, however no attachment to the GPRS network is performed. When a RING indicates a GSM connection request, the connection is **NOT** accepted, instead the logger establishes a GPRS connection to the central server, transmits data and checks for an available command message.

In this mode the index of an IP transmission configuration can be defined (see chapter "SetIPDataTransmission command"). This configuration (=internet connection + server) is then used to establish the internet connection to the server and transmit data.

In principle this mode could be used with any kind of phone connection indicating a RING and any internet connection, e.g. analog modem on analog landline and DSL connection using the same analog landline.

This mode provides a simple way to trigger an IP communication with a logger that cannot be contacted directly due to dynamically assigned IP addresses.

Instead of reacting to a RING only, the logger can react to a RING or an incoming SMS. In this mode, the logger checks periodically, if a SMS was received and connects to a server in this case.

In both modes (RING or RING+SMS), a number of up to 4 phone numbers can be configured, to which the logger will react.

7.1.1.3. Accepting IP connections

This mode allows to directly connect to the logger via an IP connection. In this mode at the begin of the maintenance window, the internet connection device (GPRS module) is switched on and a connection to the internet is established (GPRS module attaches to network).

When the internet connection is available, the logger optionally publishes its current IP address using the update procedure by a HTTP GET request defined by DynDNS, as illustrated in the URL below:

http://username:password@members.dyndns.org/nic/update?hostname=abc&myip=1.2.3.4...

This URL can be defined by the user, so not only DynDNS can be used for publishing the IP address, but the current IP address could be also transmitted to the central server only, so that it is known to the server, but not to the public. In this case a server side script must be

provided, which receives the URL parameters “hostname” and “myip”, where hostname can be configured to be the station ID, and stores them on the server.

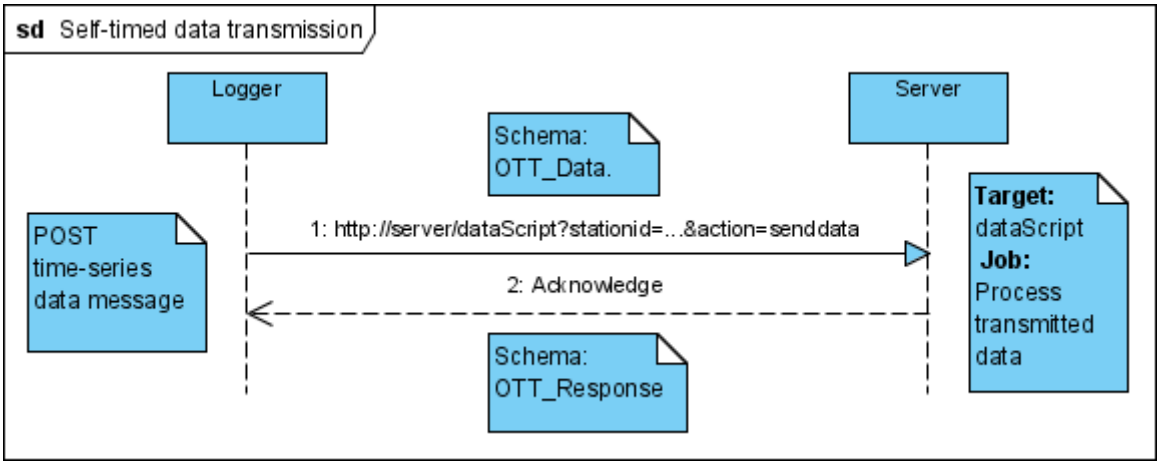
The options for publishing the current IP address are set using the logger operating software.

As soon as the IP address of the logger is known, the operating software of the logger can be used to establish an IP connection to a listening socket on the logger and perform operations on the logger (e.g. modification of the configuration). The logger listens in this case on server port 80 for incoming requests.

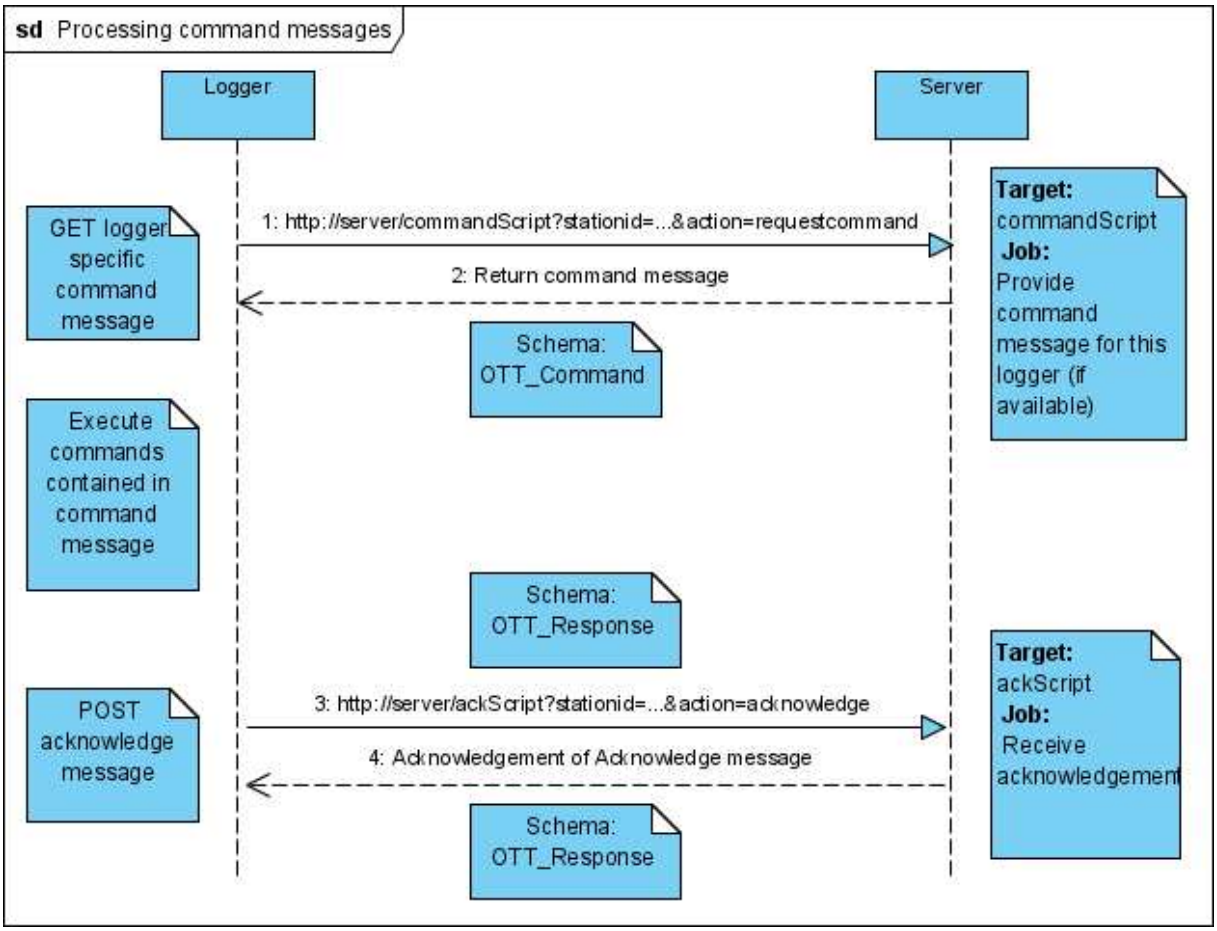
See also the sequence diagram “Diagram: Maintenance window (accept IP connection)” in the Appendix, that illustrates this mode.

8. Appendix A: Sequence diagrams

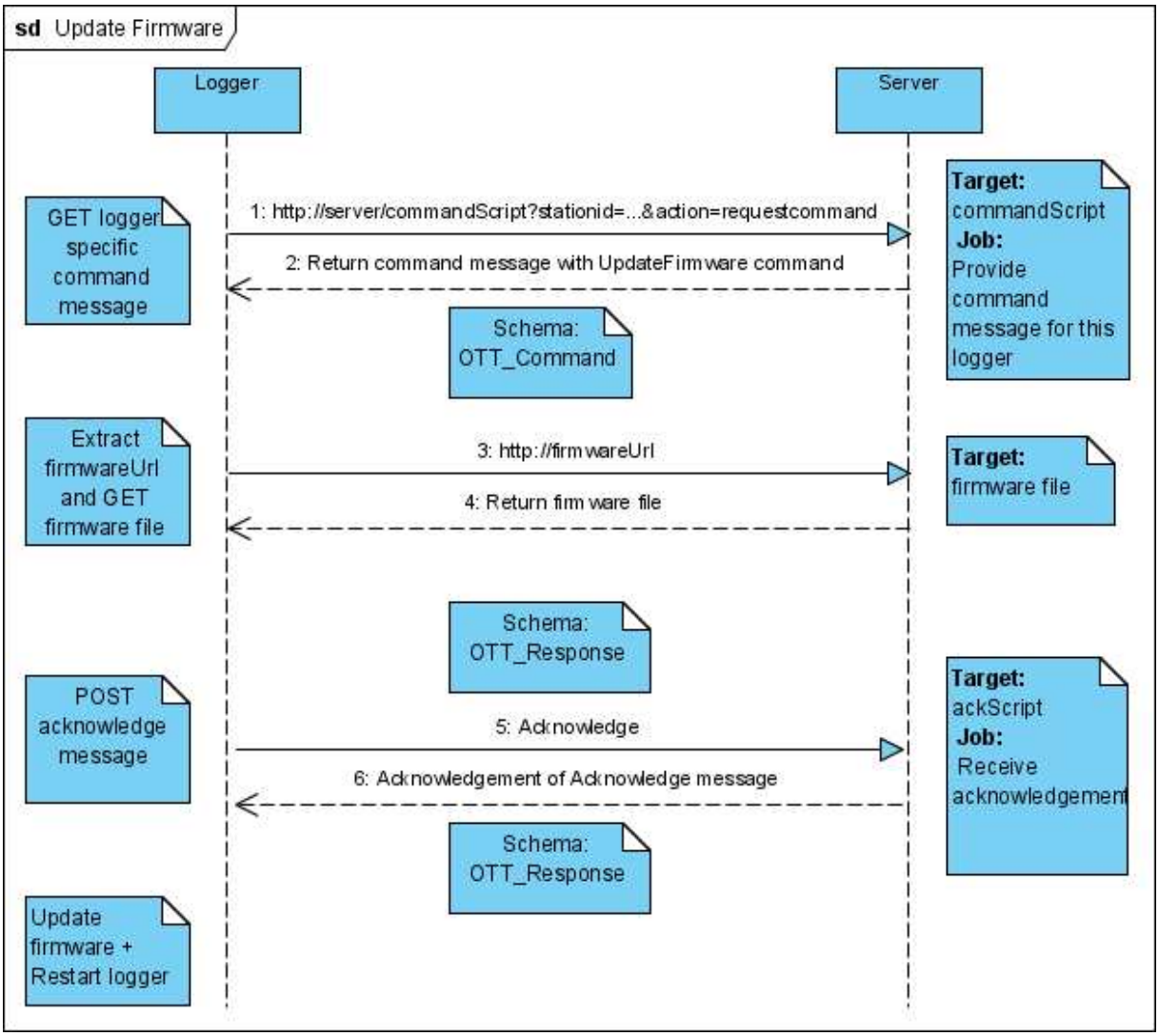
8.1. Diagram: Self-timed data transmission



8.2. Diagram: Processing general command messages



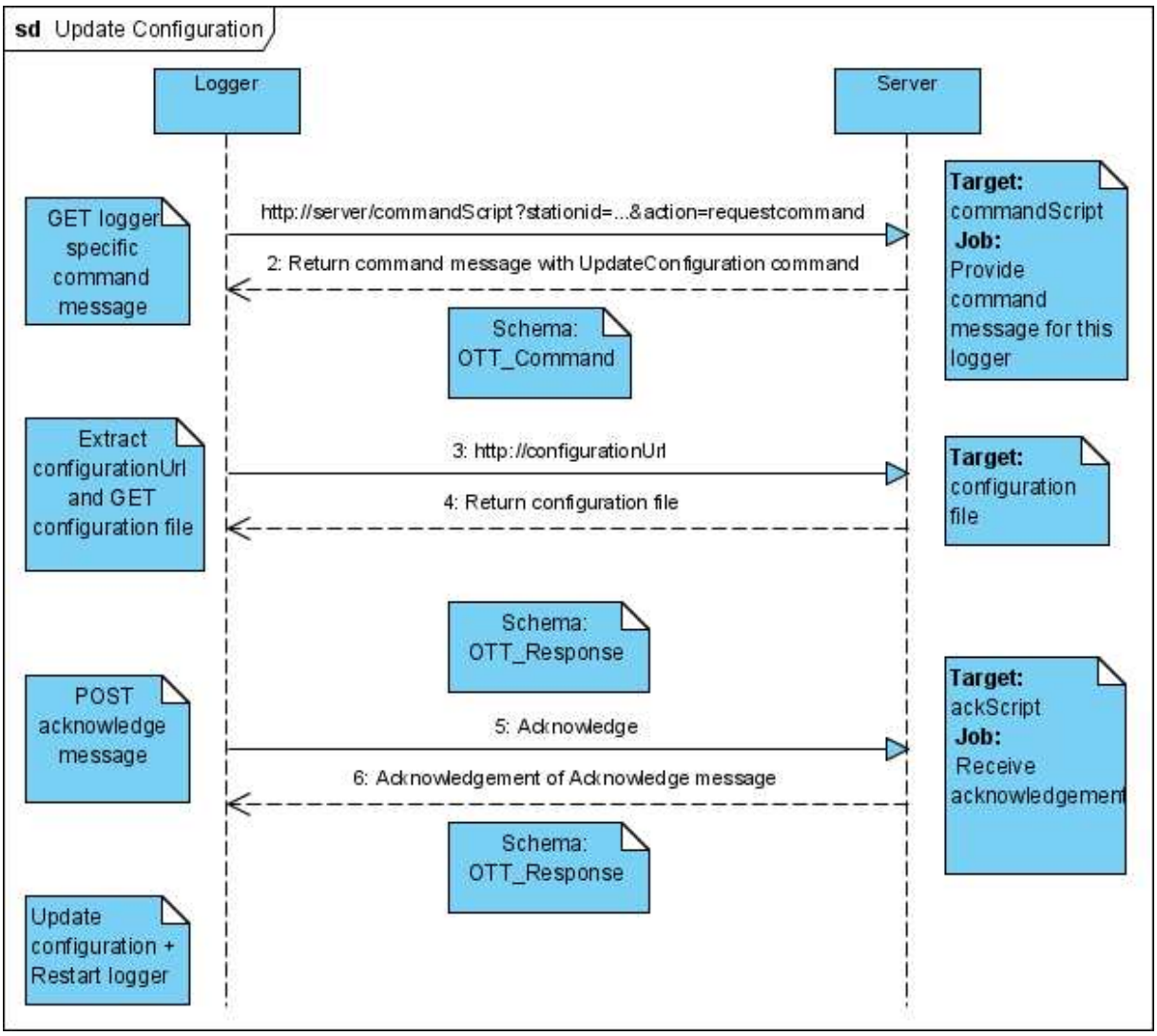
8.2.1. Diagram: Process UpdateFirmware command



Note that the acknowledgement of the command message (step 5 in diagram) is done after the firmware file was downloaded, but before the actual firmware update is performed. So if there is an error in the command message, or the file could not be downloaded from the URL, there will be a negative acknowledgement.

After the firmware was actually updated, no additional acknowledgement is sent. However the `firmware` attribute of the `<StationInfo>` element in the next transmission of a self-time data message will reflect the new firmware version.

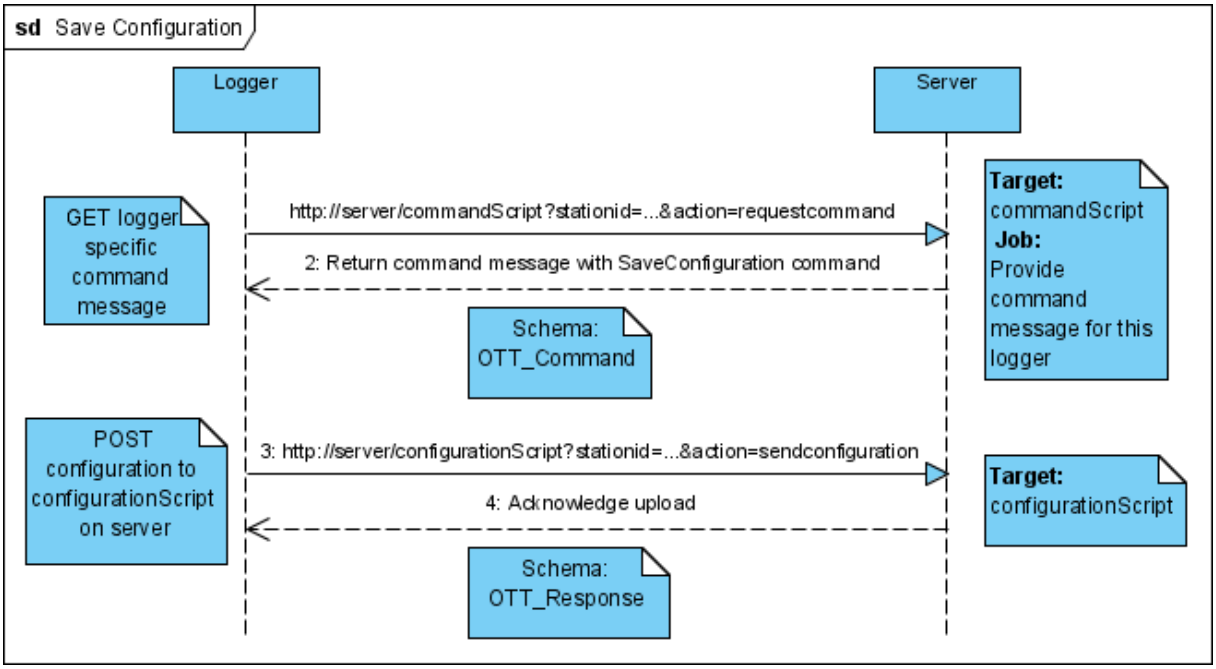
8.2.2. Diagram: Process UpdateConfiguration command



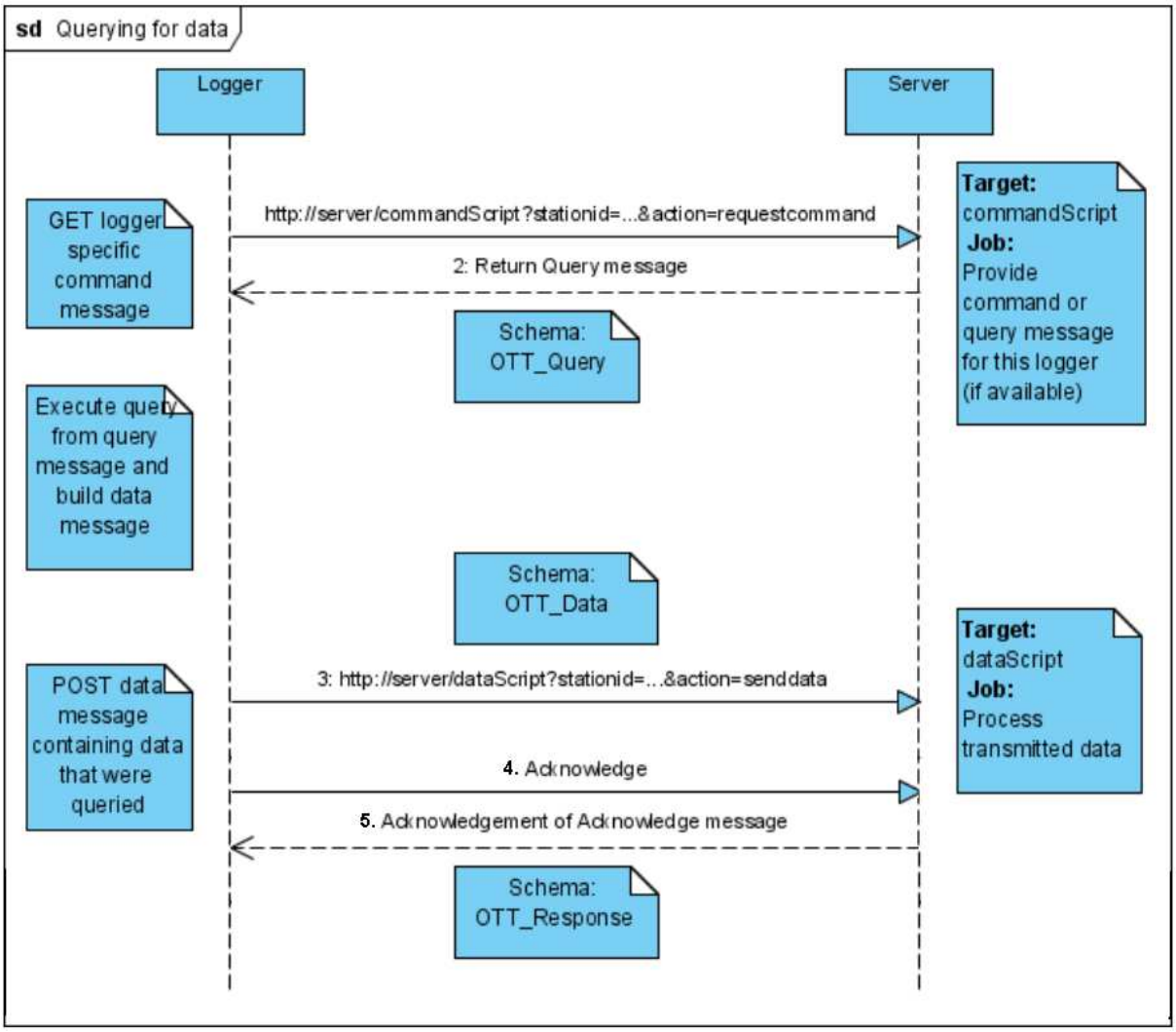
Note that the acknowledgement of the command message (step 5 in diagram) is done after the configuration file was downloaded, but before the actual configuration update is performed. So if there is an error in the command message, or the file could not be downloaded from the URL, there will be a negative acknowledgement.

After the configuration was actually updated, no additional acknowledgement is sent. However the `configtime` attribute of the `<StationInfo>` element in the next transmission of a self-time data message will reflect when the configuration was updated the last time.

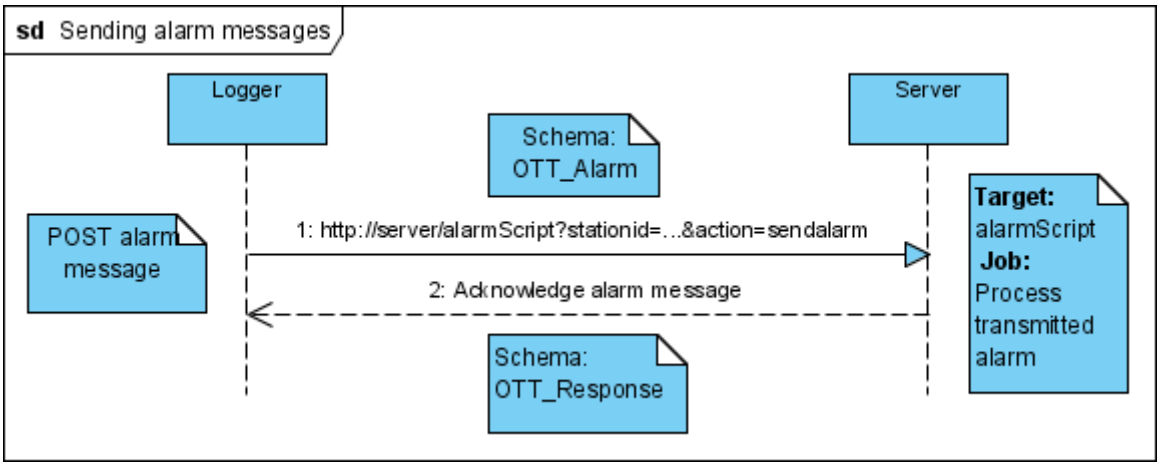
8.2.3. Diagram: Process SaveConfiguration command



8.3. Diagram: Processing a query



8.4. Diagram: Sending alarm messages

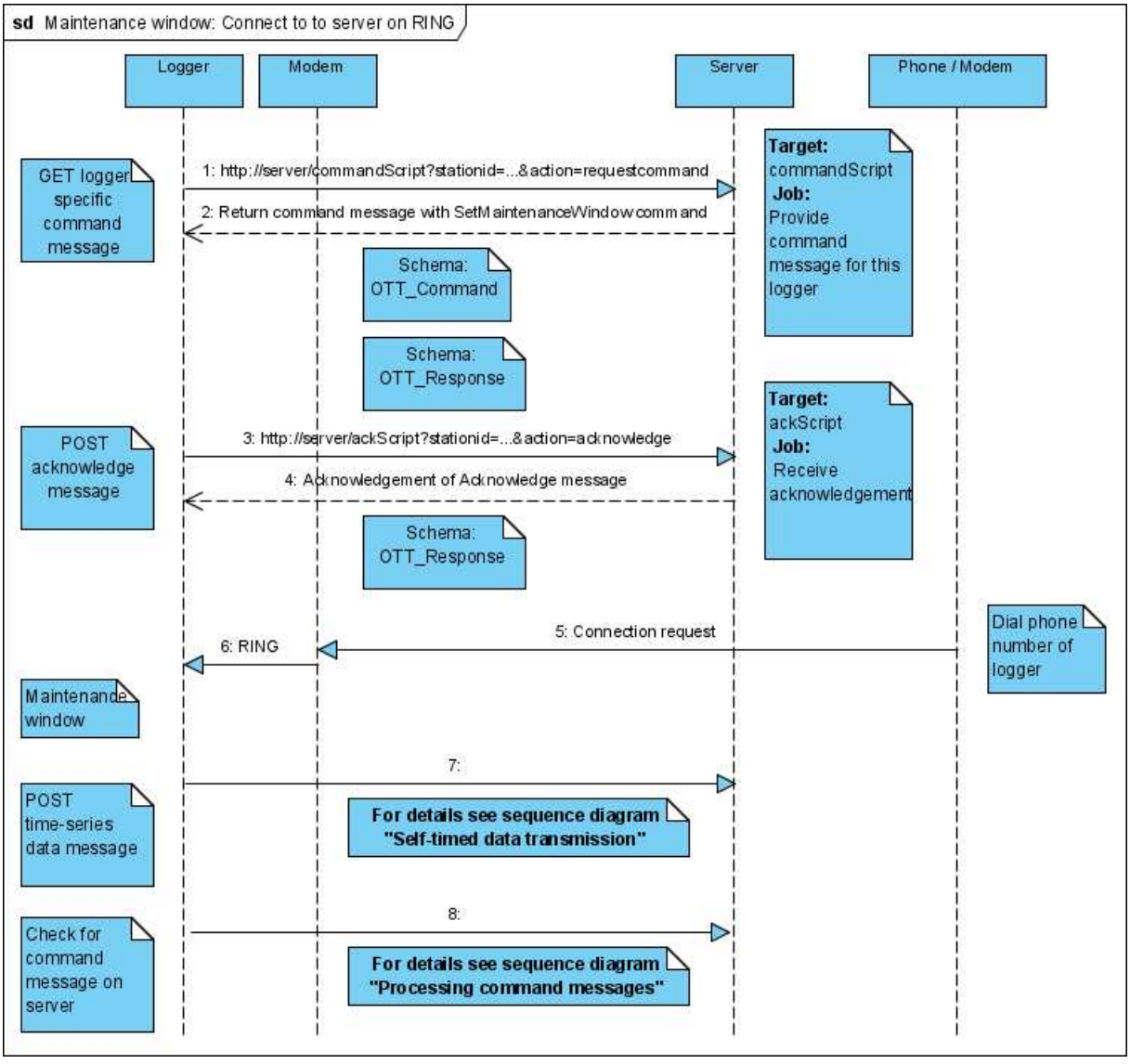

sd Sending alarm messages
Logger
Server
Schema:
OTT_Alarm
POST alarm
message
1: http://server/alarmScript?stationid=...&action=sendalarm
2: Acknowledge alarm message
Target:
alarmScript
Job:
Process
transmitted
alarm
Schema:
OTT_Response

8.5. Diagram: Maintenance window (connect to server on RING)

In the sequence diagram below, a maintenance window with mode “connect to server on RING” is opened using the SetMaintenanceWindow command. During the maintenance window a phone call to the station phone number triggers a data-transmission to the server with a subsequent check for a command message.

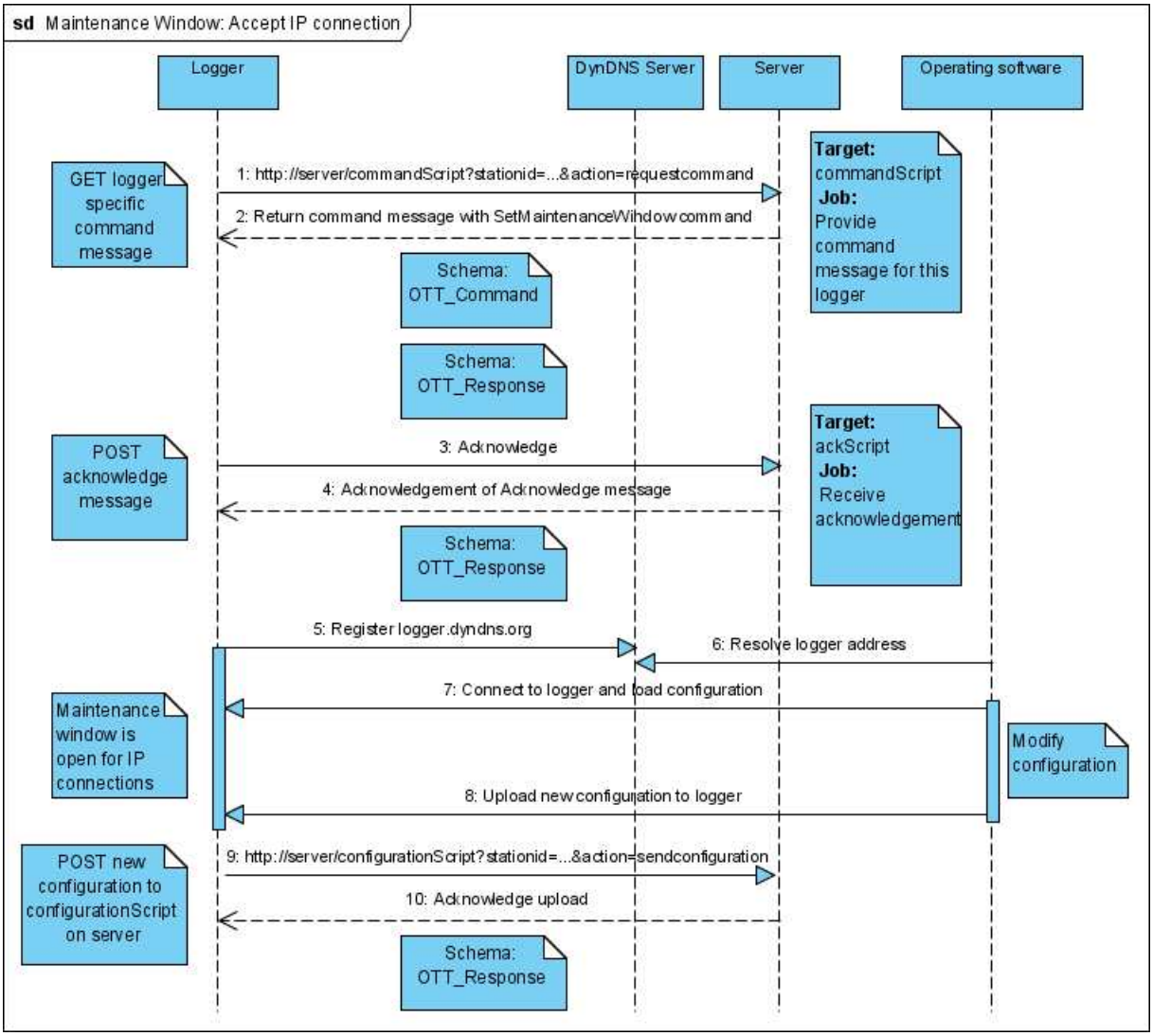
The details of the data transmission to the server are illustrated in chapter “Diagram: Self-timed data transmission”.

The details of the check for a command message and the optional command execution are illustrated in chapter “Diagram: Processing general command messages”.



8.6. Diagram: Maintenance window (accept IP connection)

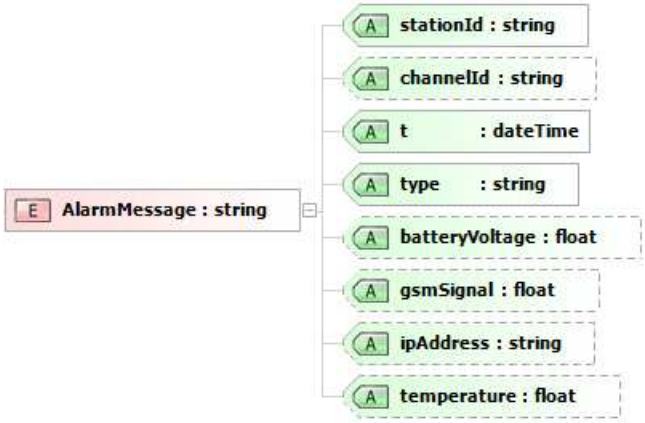
In the sequence diagram below, a maintenance window with mode “accept IP connection” is opened using the SetMaintenanceWindow command. During the maintenance window, the logger operating software connects to the logger via an IP connection and performs a configuration update. After the update is complete, the logger posts the new configuration to the server.



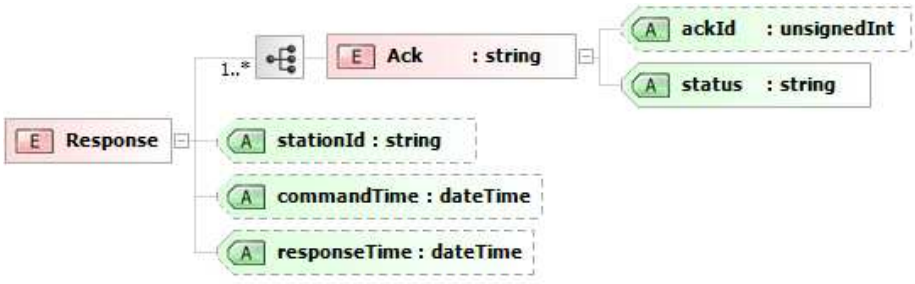
9. Appendix B: XML Schemas

9.1. OTT_Query

9.2. OTT_Alarm

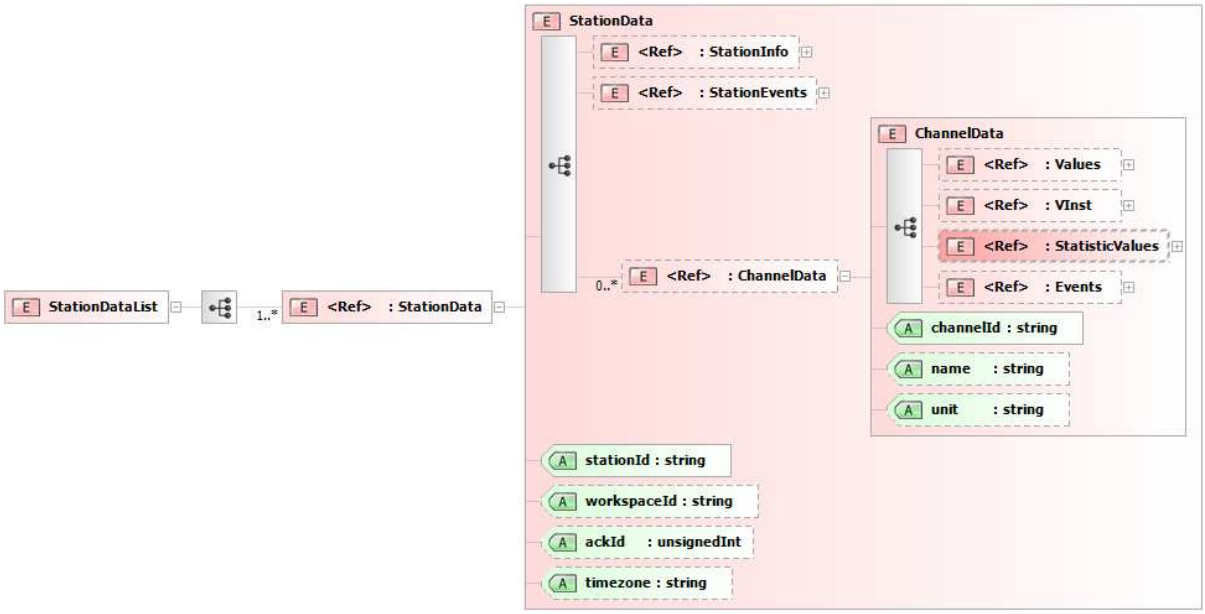


9.3. OTT_Response



9.4. OTT_Data

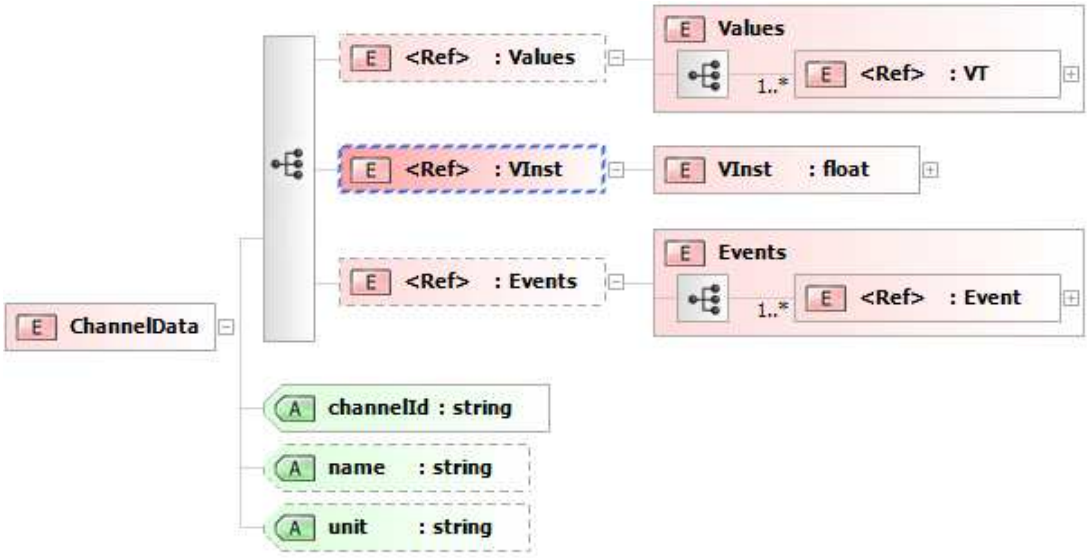
9.4.1. General Structure



The `<StationDataList>` root element contains one or more `<StationData>` elements. OTT netDL1000 always produces exactly one `<StationData>` element.

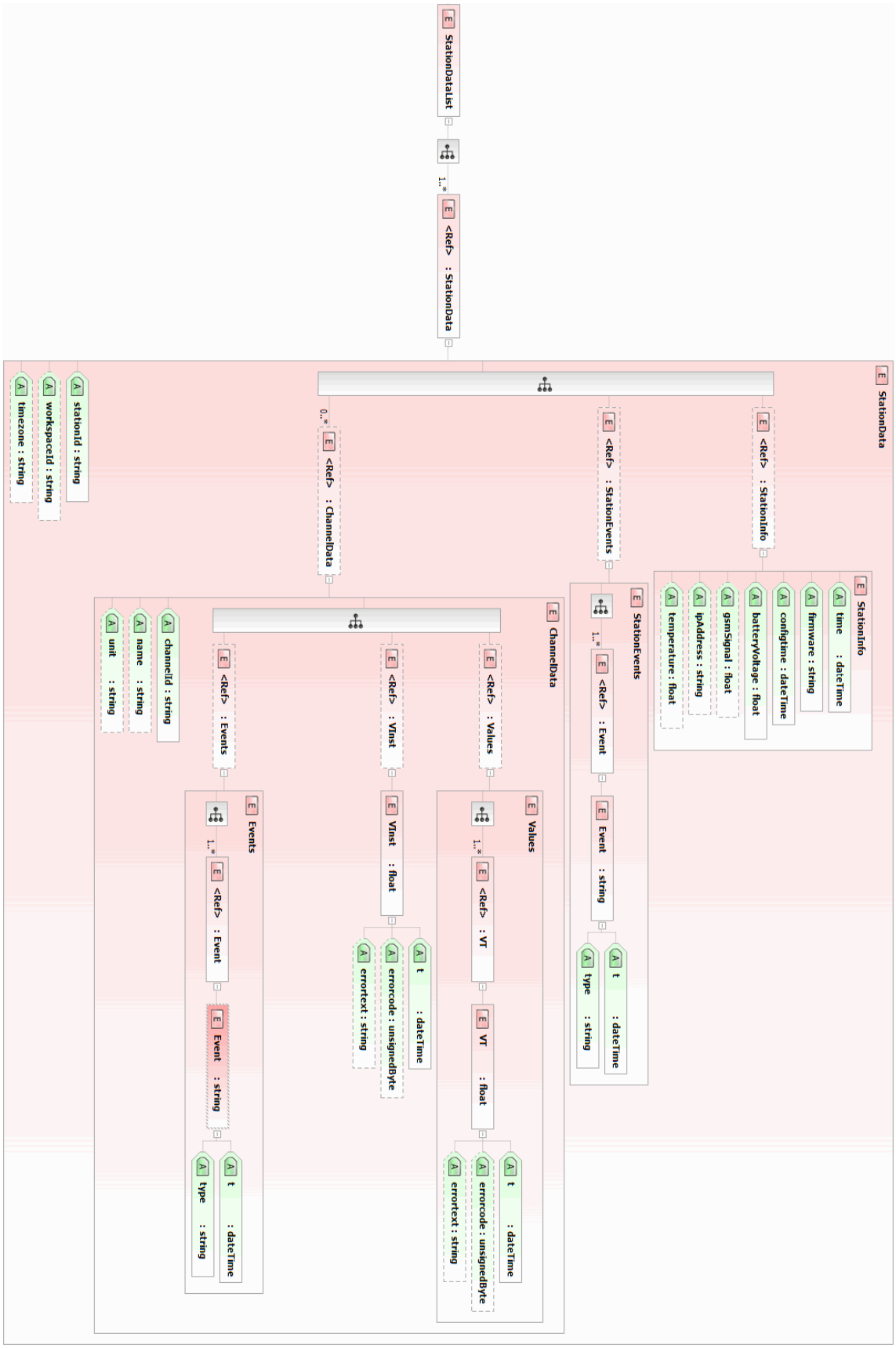
A `<StationData>` element consists of a sequence of an optional `<StationInfo>` element, an optional `<StationEvents>` element and optionally one or more `<ChannelData>` elements, that are containers for the actual data of the individual channels.

9.4.2. Channel Data



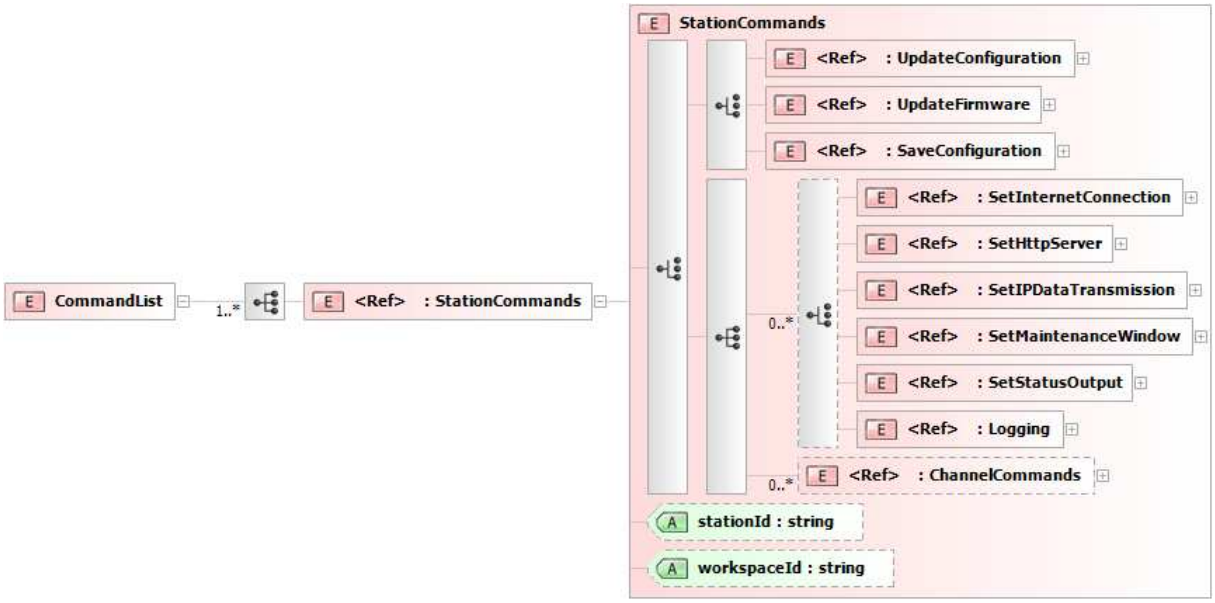
A <ChannelData> element consists of a sequence of optional <Values>, <VInst> and <Events> elements, containing the storage values, the instantaneous value and channel specific events.

9.4.3. Complete schema



9.5. OTT_Command

9.5.1. General Structure

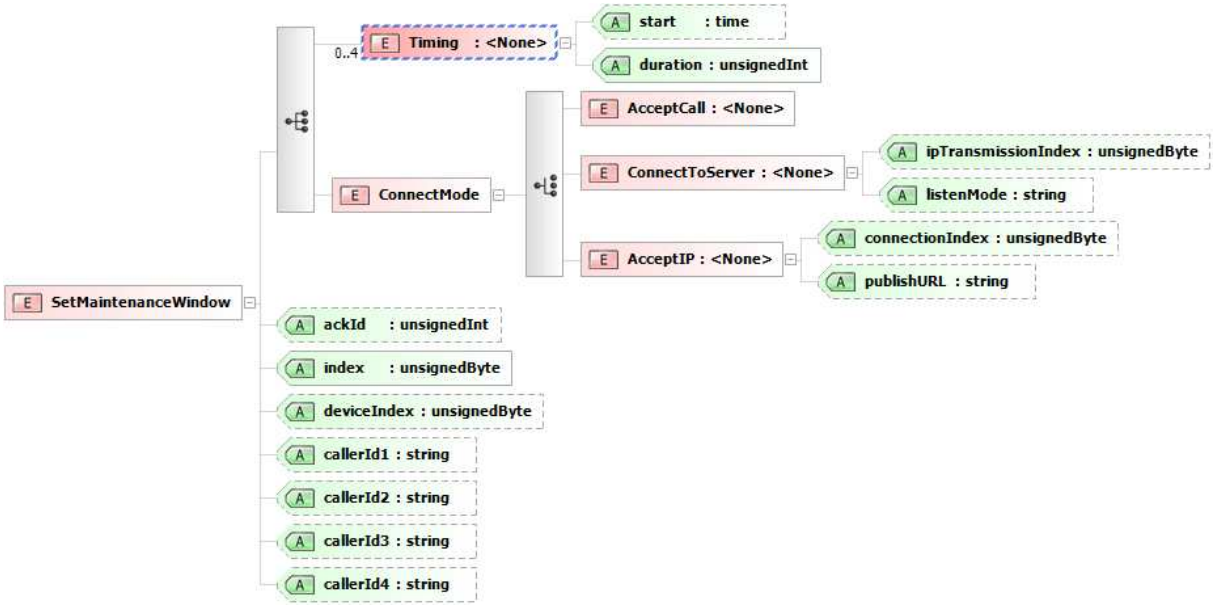


The <CommandList> root element contains one or more <StationCommands> elements, where OTT netDL1000 only processes the first <StationCommands> element.

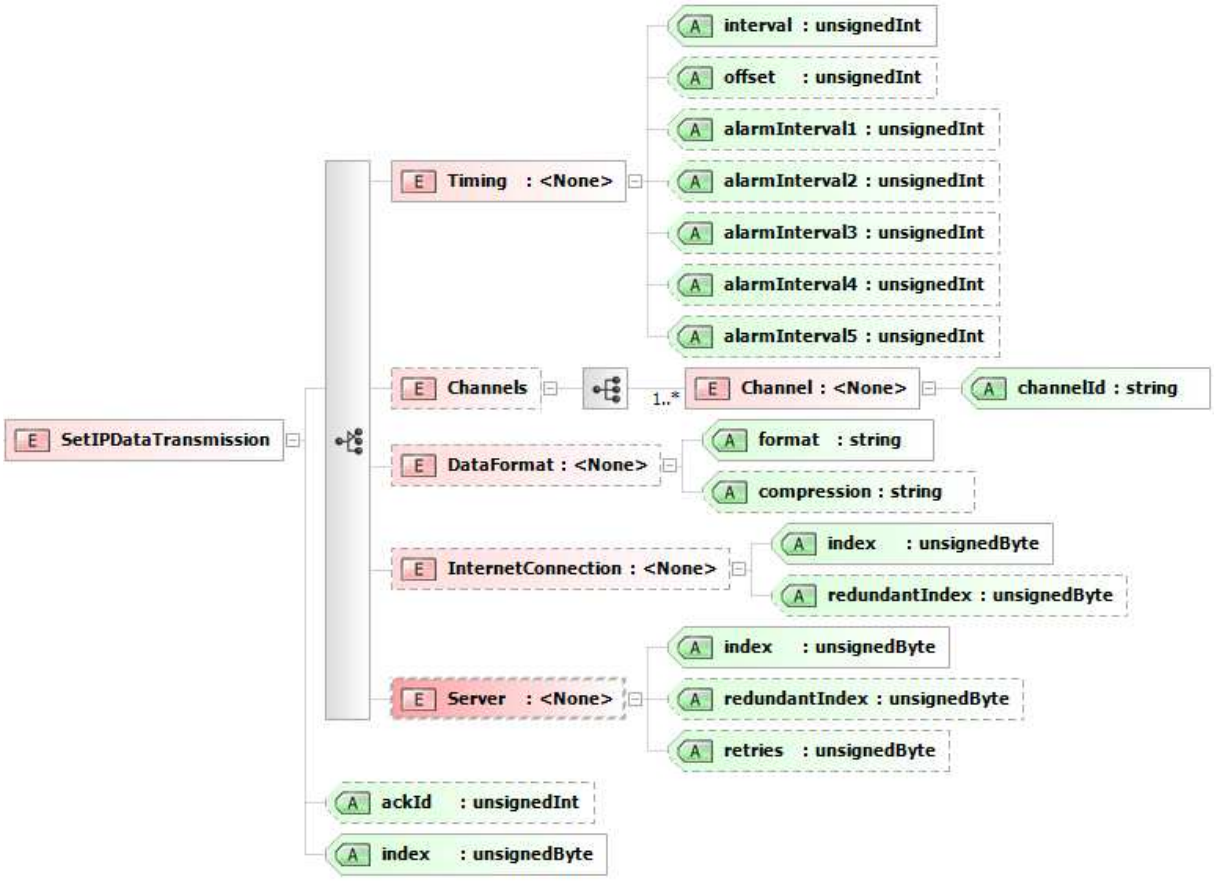
A <StationCommands> element can either contain one of the command elements <UpdateConfiguration>,<UpdateFirmware> or <SaveConfiguration> or a sequence of commands like <SetInternetConnection> or <SetHttpServer>, where each of these commands can occur multiple times. These commands are followed by optional <ChannelCommands> elements, which contain channel specific commands.

In the next subchapters only schemas for the more complex command elements are presented.

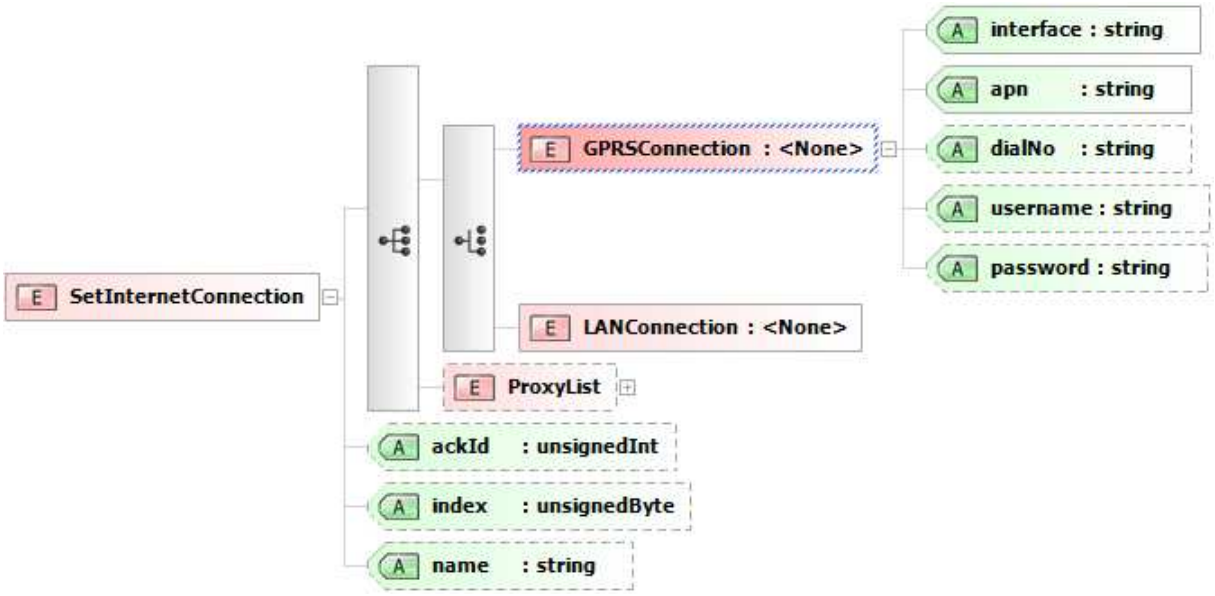
9.5.2. SetMaintenanceWindow



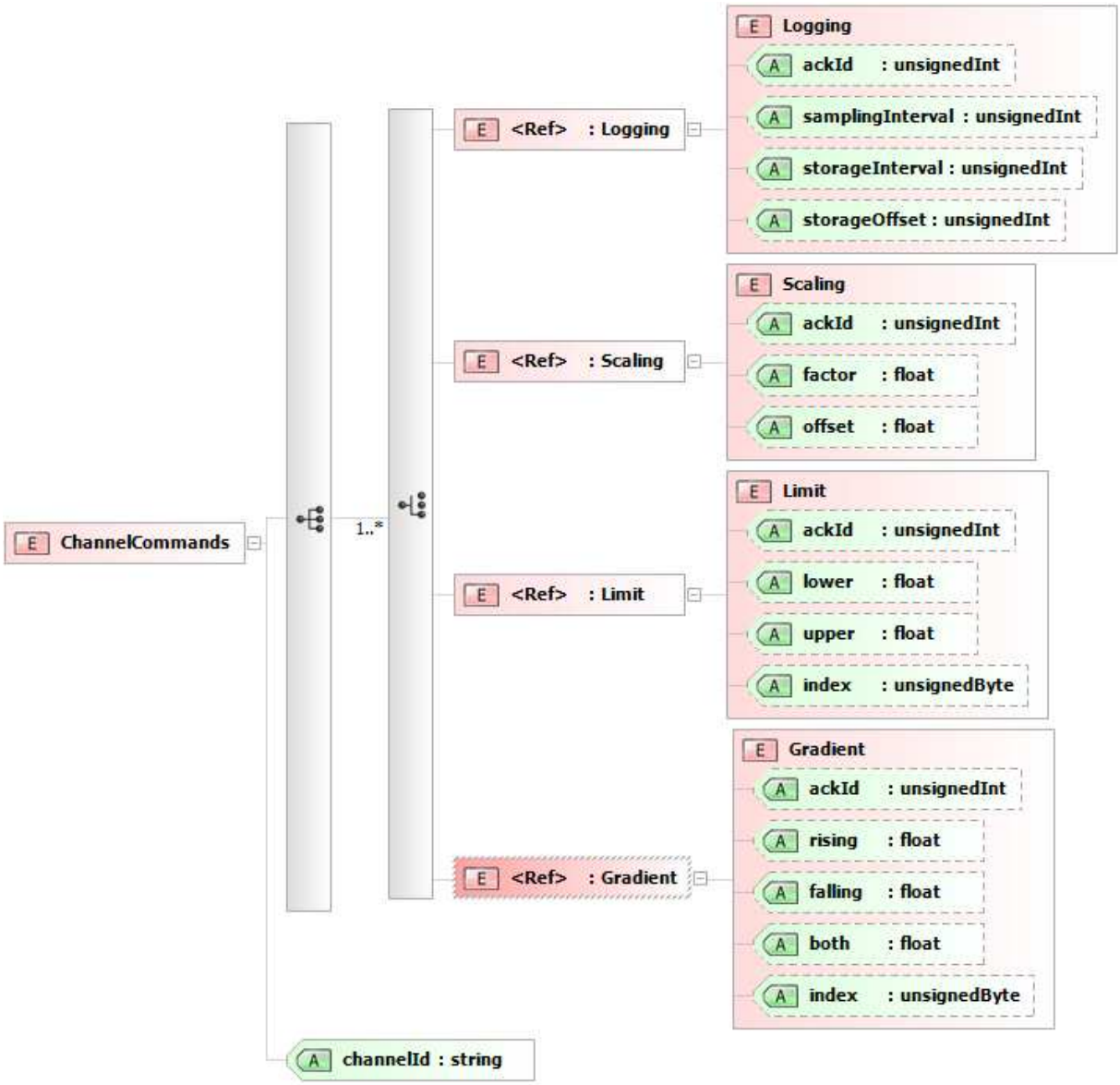
9.5.3. SetIPDataTransmission



9.5.4. SetInternetConnection



9.5.5. Channel commands



A `<ChannelCommands>` element contains a sequence of child elements `<Logging>`, `<Scaling>`, `<Limit>` and `<Gradient>` where the order of the elements does not matter and elements can occur multiple times, as e.g. multiple limit modules may exist in a channel.