TelevisGo

Data Download Procedure

User Protocol

EN

CONTENTS

1 OVERVIEW

Data transfer communication protocol is a feature that allows a client program (hereafter called the **consumer**) to query for data stored on the supervisor/monitoring server application (hereafter called the **service**).

Data exchange is performed through a series of TCP *messages* and responses over a connection established between the consumer and the service.

1.1.1 Consumer Initiative

Connection can be opened by the consumer (client-side initiative) who takes responsibility into calling the services IP and port in order to establishing the connection.

This modality is hereafter called **pull mode** and comes handy when the service's IP and TCP port are easily reachable and don't change over time, while the consumer's ones may vary a lot.

More than a consumer may contact the service in this way, minding that the more consumers simultaneously ask for data, the more work is loaded onto the service (and its performance will be affected).

1.1.2 Service Initiative

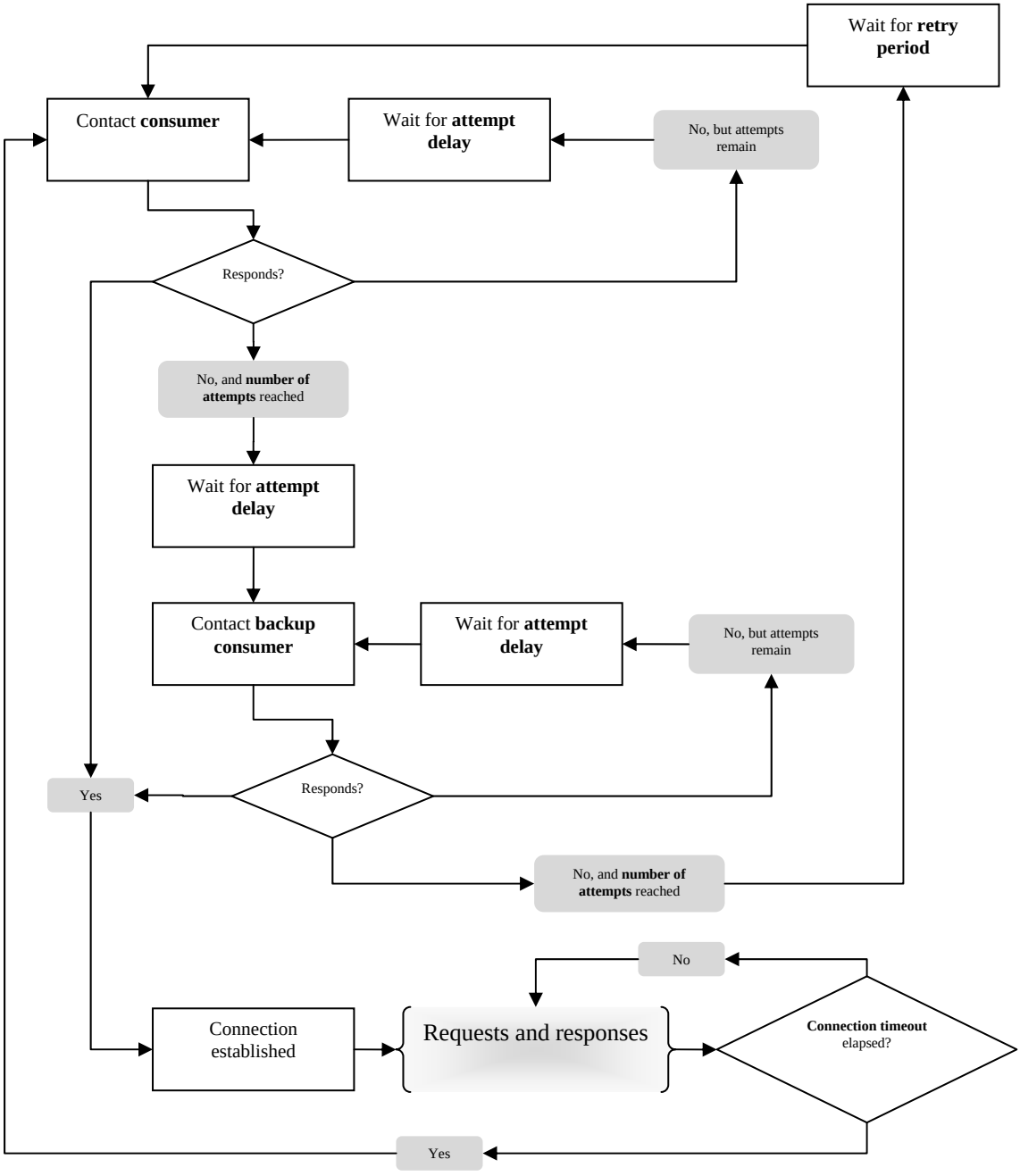
Otherwise, the service can take responsibility into calling the consumer's IP and port (server-side initiative) thus establishing the connection. Service exposes settings that the user may configure in order to define:

- the consumer's IP (or URL) and TCP port,
- a backup consumer's IP (or URL) and TCP port (in case the first one is down),
- the frequency with which the service calls the consumer
- the number of attempts (and their frequency) in establishing the connection
- a timeout after which an established connection is closed (unless a keep-alive is periodically being sent by the consumer)

This modality is hereafter called **push mode** and come handy when the service's IP and/or port are unreachable (e.g.: local IT regulations), or change frequently (dynamic IP).

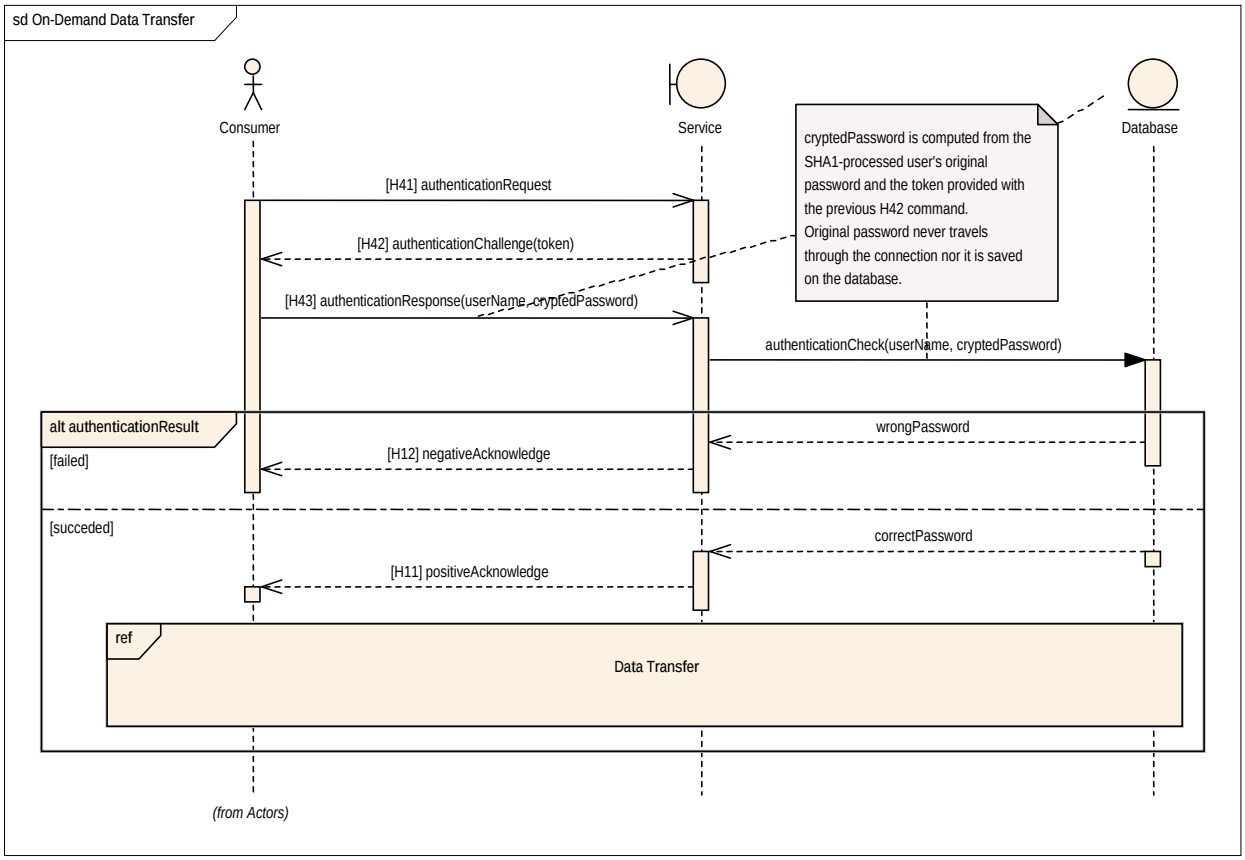
In push mode the service periodically attempts to establish a connection to the consumer. Connection is established when the consumer (which should be listening to the set port) accepts the incoming connection. Once the connection is established, the service will drop it if it doesn't receive *messages* from the consumer for a set period of time. In order to keep the connection alive, the consumer may send any request. If it does not need any data, the consumer may send a keep-alive message.

Connection attempts work as follows:

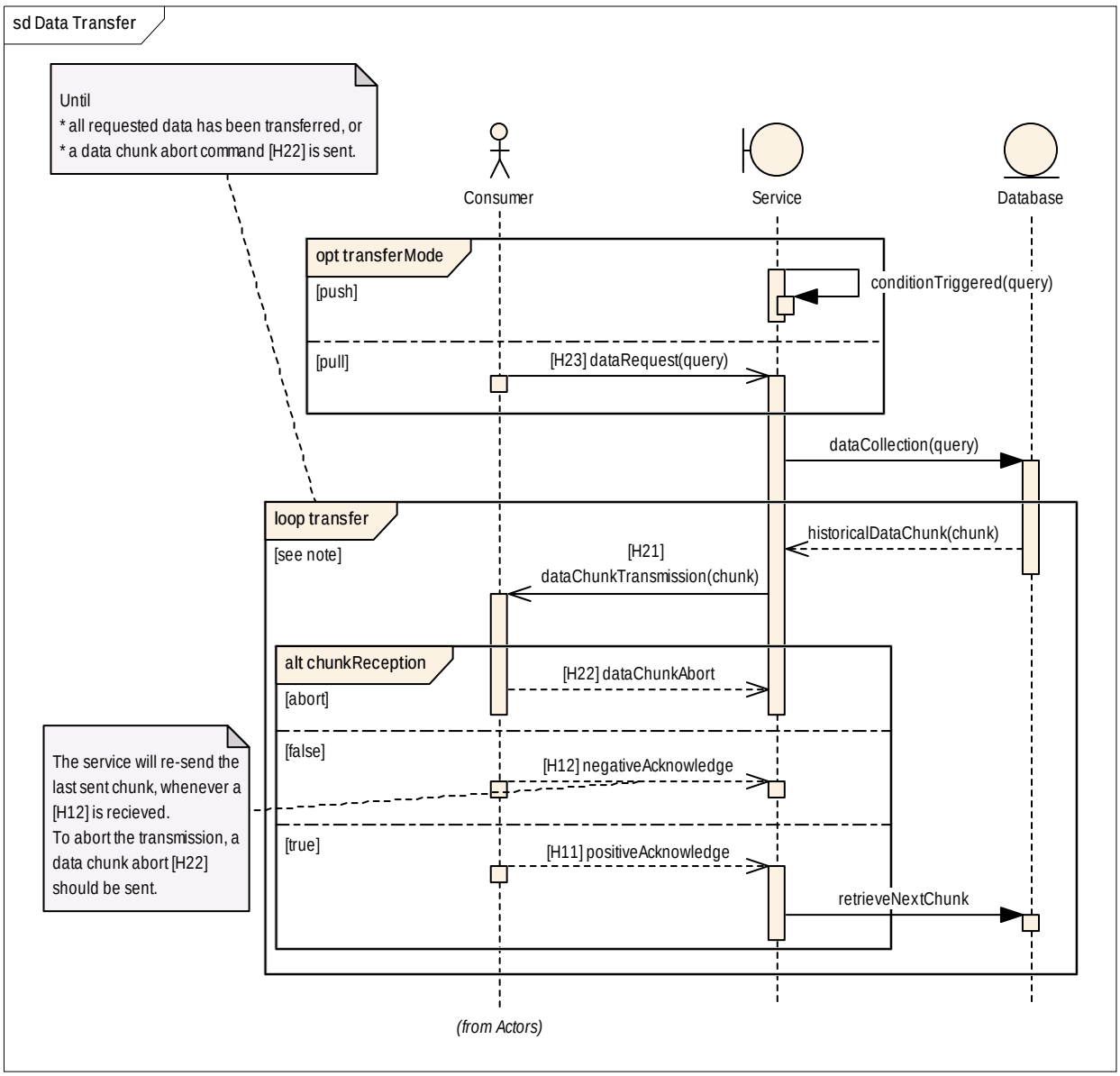

Wait for **retry period**
Contact **consumer**
Wait for **attempt delay**
No, but attempts remain
Responds?
No, and **number of attempts** reached
Wait for **attempt delay**
Contact **backup consumer**
Wait for **attempt delay**
No, but attempts remain
Responds?
Yes
No, and **number of attempts** reached
No
Connection established
Requests and responses
Connection timeout elapsed?
Yes

1.1.3 Mixed Mode

Push mode and pull mode may coexist as long as there is only one consumer reached through push mode whilst all the others use the pull mode.



An *overview* of the authentication challenge between Consumer and System in pull mode



Historical data transfer between System and Consumer

2 GENERAL FRAME FORMAT

The following table describes the *general frame format* for all the *messages* sent from e to the service. All the values are considered big endian.

Header				**Command**	**Data**	**CRC32**
ServiceType	Version	SendingTime	Length	Command	Data	CRC32
1 Byte	1 Byte	7 Bytes	4 bytes	1 Byte	n Bytes	4 Bytes

Where

Field Name	**Description**
ServiceType	The id of the specific service. For data transfer use H44
Version	Protocol version. This version is H01.
SendingTime	The sending time of the message on the sender's side. It must be formatted according to the *time format* (see **Real Time Data Request (H24)** (below).
Length	The length of the message, including the header.
Command	The command byte.
Data	The data field. It is optional and its content changes according to the command byte.
CRC32	The CRC32. The polynomial is the same used by CCITT-32 (HEDB88320).

3 MESSAGES

3.1 Authentication Request (H41)

The authentication request message must be sent by the consumer as first message after the establishment of the connection. No data field must be provided

ServiceType	Version	SendingTime	Length	Command	CRC32
H44	H01	time value	H00000012	H41	CCITT-32

The service may reply to this message with an authentication challenge or with a negative acknowledge. If a negative acknowledge is provided the service will disconnect the consumer.

3.2 Authentication Challenge (H42)

The authentication challenge provides a set of random data that must be used by the consumer to compute the expected cryptographic reply that is needed to authenticate the user.

ServiceType	Version	SendingTime	Length	Command	Data	CRC32
H44	H01	time value	Length	H42	Random bytes	CCITT-32

The consumer must reply with an authentication response message. Any other reply will be discarded and the connection terminated by the service.

3.3 Authentication Response (H43)

With the authentication response the consumer provides the cryptographic reply requested to authenticate it.

Header	Command	Data	User	CRC32
13 bytes	H43	20 bytes Cryptographic reply	username	CCITT-32

The username provided is the UTF8 representation of the username.

The 20 bytes cryptographic reply is computed with the following algorithm:

- Compute the output (PW-SHA1) of the SHA1 algorithm using the UTF8 representation of the user password as input.
- Append the output (PW-SHA1) of the previous step to the random bytes provided by the authentication challenge to form a single array (PRE-SHA1).
- Compute the output of the SHA1 algorithm using the output (PRE-SHA1) of the previous step as input. Use this output as the “20 bytes cryptographic reply”.
- If the authentication is successful the service will reply to this message with a positive acknowledge and after that the consumer may issue further commands.
- If the authentication is not successful, then the service will reply with a negative acknowledge and the connection will be terminated.

3.4 Positive Acknowledge or ACK (H11)

The positive acknowledge is used as general positive reply to a request. See the documentation of the single request to know if this is supported.

ServiceType	Version	SendingTime	Length	Command	CRC32
H44	H01	time value	H00000012	H11	CCITT-32

3.5 Negative Acknowledge or NACK (H12)

The negative acknowledge is used as general negative reply to a request. It may or may not contain an optional series of n status bytes. See the documentation of the single request to know if the H12 command is supported.

Header	Command	Statuses [optional]	CRC32
13 bytes	H12	[optional] n-bytes statuses	CCITT-32

Here is a table of the status bytes in use.

Status	Returned by	Meaning
1	H24	Real time request is too close to the previous one.
2	H20	The submitted filter does not match any resource.
17	Any	Internal error, report it to Eliwell.
Any other value	None	Reserved for future use.

3.6 Data Chunk Transmission (H21)

The data chunk transmission is used by the service to the consumer.

Header	Command	ChunkId	Flags	StartTime	EndTime	Data	CRC32
13 bytes	H21	2 bytes	1 byte	time value	time value	n-bytes data chunk	CCITT-32

Where

Field Name	Description
ChunkId	Is a progressive index of the chunk.
Flags	Is a set of flags used to inform about the compression of the data or the last data chunk (see next table).
StartTime	Starting time of the data (timestamp of the oldest datum present in the chunk).
EndTime	Ending time of the data (timestamp of the newest datum present in the chunk).
Data	Data block. Data within it may be compressed (zip algorithm) or plain, according to the compression flag. The plain data depends upon the request that generated the data chunk transmission. For detailed explanation of the various plain data see **Errore. L'origine riferimento non è stata trovata.** (**Errore. L'origine riferimento non è stata trovata.**).

The following table explains the meaning of the flags byte.

7 (MSb)	6	5	4	3	2	1	0 (LSb)
RFU	RFU	RFU	RFU	areAcquisitionsStopped	RFU	isCompressed	isLastChunk

Where

Bit Name	Position	Description
isLastChunk	0	It is set if this is the last chunk.
isCompressed	1	It is set if the data field is compressed
RFU	2	Reserved for future use.
areAcquisitionsStopped	3	It is used when the request is a Real Time Data Request Command (H25) only. It is set if the acquisitions are stopped.
RFU	4-7	Reserved for future use.

The consumer may reply to a Data Chunk Transmission in three ways:

- Positive Acknowledge or ACK (H11) (see above). The consumer states that the transmission is successful. If this is not the last chunk the next chunk will be transmitted, otherwise both the service and the consumer are allowed to disconnect.
- Negative Acknowledge or NACK (H12) (see above). The consumer states that an error occurred during the transmission. The service will retry the transmission of the same chunk. The consumer must not provide any status byte if NACK is used in this way.
- Data Chunk Abort (H22), (see below). The consumer states that the transmission will be aborted.

3.7 Data Chunk Abort (H22)

The data chunk abort message is sent by the consumer to abort a data chunk transmission.

ServiceType	Version	SendingTime	Length	Command	CRC32
H44	H01	time value	H00000012	H22	CCITT-32

3.8 Configuration Request (H20)

The configuration request is sent by the consumer to request data.

Header	Command	Flags	Request	CRC32
13 bytes	H20	1 byte	n-bytes request	CCITT-32

Where

Field Name	Description
Flags	Is a set of flags used to inform about the compression of the request.
Request	Request block. Request data within it may be compressed (zip algorithm) or plain, according to the isRequestCompressed flag. For detailed explanation of the plain data see **Current Configuration Request** (below).

The following table explains the meaning of the flags byte.

7 (MSb)	6	5	4	3	2	1	0 (LSb)
RFU	RFU	RFU	RFU	RFU	isRequestCompressed	isReplyCompressed	RFU

Where

Bit Name	Position	Description
RFU	0	Reserved for future use.
isReplyCompressed	1	The data contained in the reply will be compressed
isRequestCompressed	2	The request block is compressed.
RFU	3-7	Reserved for future use.

The service may reply in the following ways:

- Negative Acknowledge or NACK (H12) (see above). The service states that an error occurred while decoding the packet or parsing the XML request.
- Data Chunk Transmission (H21) (see above). The service sends data about the current configuration, according to **Current Configuration Response**. When the request is submitted without authentication, the server disconnects the consumer after the completion of the sending of data.

3.9 Historical Data Request (H23)

The historical data request is sent by the consumer to request historical data of specific resources over a span of time.

Header	Command	Flags	Request	CRC32
13 bytes	H23	1 byte	n-bytes request	CCITT-32

Where

Field Name	Description
Flags	Is a set of flags used to inform about the compression of the request.
Request	Request block. Request data within it may be compressed (zip algorithm) or plain, according to the isRequestCompressed flag. For detailed explanation of the plain data see **Errore. L'origine riferimento non è stata trovata. – Request Query Message** (**below**).

The following table explains the meaning of the flags byte.

7 (MSb)	6	5	4	3	2	1	0 (LSb)
RFU	RFU	RFU	RFU	RFU	isRequestCompressed	isReplyCompressed	RFU

Where

Bit Name	Position	Description
RFU	0	Reserved for future use.
isReplyCompressed	1	The data contained in the reply will be compressed
isRequestCompressed	2	The request block is compressed.
RFU	3-7	Reserved for future use.

3.10 Real Time Data Request (H24)

The configuration request is sent by the consumer to request current configuration's real time data.

Header	Command	Flags	CRC32
13 bytes	H24	1 byte	CCITT-32

Where

Field Name	Description
Flags	Is a set of flags used to inform about the compression of the request.

The following table explains the meaning of the flags byte.

7 (MSb)	6	5	4	3	2	1	0 (LSb)
RFU	RFU	RFU	RFU	RFU	RFU	isReplyCompressed	RFU

Where

Bit Name	Position	Description
RFU	0	Reserved for future use.
isReplyCompressed	1	The data contained in the reply will be compressed
RFU	2-7	Reserved for future use.

The service may reply in the following ways:

- No reply message and sudden disconnection. This happens when a NACK reply should be sent and the request was submitted without authentication.
- Negative Acknowledge or NACK (H12) (see above) with no status bytes. The service states that an error occurred while decoding the packet.
- Negative Acknowledge or NACK (H12) (see above) with status code H01. The service states that the current real time request is too close to the previous one.
- Data Chunk Transmission (H21) (see above). The service sends data about the real time data, according to
- **Real Time Response**. When the request is submitted without authentication, the server disconnects the consumer after the completion of the sending of data.

3.11 Real Time Configure Request (H25)

This is sent by the consumer to set the configuration of the data to be retrieved with a Real Time Data Request (H24).

Header	Command	Flags	Request	CRC32
13 bytes	H25	1 byte	n-bytes request	CCITT-32

Where

Field Name	Description
Flags	Is a set of flags used to inform about the compression of the request.
Request	Request block. Request data within it may be compressed (zip algorithm) or plain, according to the isRequestCompressed flag. For detailed explanation of the plain data see **Set Real Time Filter Request**.

The following table explains the meaning of the flags byte.

7 (MSb)	6	5	4	3	2	1	0 (LSb)
RFU	RFU	RFU	RFU	RFU	isRequestCompressed	RFU	RFU

Where

Bit Name	Position	Description
RFU	0-1	Reserved for future use.
isRequestCompressed	2	The request block is compressed.
RFU	3-7	Reserved for future use.

The server may reply in the following ways:

- Negative Acknowledge or NACK (H12) (see above). The server states that an error occurred while decoding the packet or parsing the XML request.
- Negative Acknowledge or NACK (H12) (see above) with status code H02. The server states that the filter provided with this request does not match any resource. The current real time configuration is not changed.
- Positive Acknowledge or ACK (H11) (see above). The server states that the change in configuration was successful.

3.12 Parameters Read Request (H26)

This is sent by the consumer to execute a parameter reading operation.

Header	Command	Flags	Request	CRC32
13 bytes	H26	1 byte	n-bytes request	CCITT-32

Where

Field Name	Description
Flags	Is a set of flags used to inform about the compression of the request.
Request	Request block. Request data within it may be compressed (zip algorithm) or plain, according to the isRequestCompressed flag. For detailed explanation of the plain data see **Parameter Read Request**.

The following table explains the meaning of the flags byte.

7 (MSb)	6	5	4	3	2	1	0 (LSb)
RFU	RFU	RFU	RFU	RFU	isRequestCompressed	*isReplyCompressed*	RFU

Where

Bit Name	Position	Description
RFU	0	Reserved for future use.
isReplyCompressed	1	The data contained in the reply will be compressed
isRequestCompressed	2	The request block is compressed.
RFU	3-7	Reserved for future use.

The server may reply in the following ways:

- Negative Acknowledge or NACK (H12) (see above) with no status bytes. The server states that an error occurred while decoding the packet.
- Data Chunk Transmission (H21) (see above). The server sends data about the outcome of the parameter reading operation, according to **Parameter Read Response**. When the request is submitted without authentication, the server disconnects the consumer after the completion of the sending of data.

3.13 Parameters Write Request (H27)

This is sent by the consumer to execute a parameter writing operation.

Header	Command	Flags	Request	CRC32
13 bytes	H27	1 byte	n-bytes request	CCITT-32

Where

Field Name	Description
Flags	Is a set of flags used to inform about the compression of the request.
Request	Request block. Request data within it may be compressed (zip algorithm) or plain, according to the isRequestCompressed flag. For detailed explanation of the plain data see **Parameter Write Request**.

The following table explains the meaning of the flags byte.

7 (MSb)	6	5	4	3	2	1	0 (LSb)
RFU	RFU	RFU	RFU	RFU	isRequestCompressed	*isReplyCompressed*	RFU

Where

Bit Name	Position	Description
RFU	0	Reserved for future use.
isReplyCompressed	1	The data contained in the reply will be compressed
isRequestCompressed	2	The request block is compressed.
RFU	3-7	Reserved for future use.

The server may reply in the following ways:

- Negative Acknowledge or NACK (H12) (see above) with no status bytes. The server states that an error occurred while decoding the packet.
- Data Chunk Transmission (H21) (see above). The server sends data about the outcome of the parameter writing operation, according to **Parameter Write Response**. When the request is submitted without authentication, the server disconnects the consumer after the completion of the sending of data.

3.14 Execute Device Command Request (H28)

This is sent by the consumer to execute a command on a device.

Header	Command	Flags	Request	CRC32
13 bytes	H28	1 byte	n-bytes request	CCITT-32

Where

Field Name	Description
Flags	Is a set of flags used to inform about the compression of the request.
Request	Request block. Request data within it may be compressed (zip algorithm) or plain, according to the isRequestCompressed flag. For detailed explanation of the plain data see **Device Command Request**.

The following table explains the meaning of the flags byte.

7 (MSb)	6	5	4	3	2	1	0 (LSb)
RFU	RFU	RFU	RFU	RFU	isRequestCompressed	*isReplyCompressed*	RFU

Where

Bit Name	Position	Description
RFU	0	Reserved for future use.
isReplyCompressed	1	The data contained in the reply will be compressed
isRequestCompressed	2	The request block is compressed.
RFU	3-7	Reserved for future use.

The server may reply in the following ways:

- Negative Acknowledge or NACK (H12) (see above) with no status bytes. The server states that an error occurred while decoding the packet.
- Data Chunk Transmission (H21) (see above). The server sends data about the outcome of the execution of the command(s), according to **Device Command Response**. When the request is submitted without authentication, the server disconnects the consumer after the completion of the sending of data.

3.15 Get System Info Request (H29)

This is sent by the consumer to get system information such as Plant Name, MAC Address, Plant Notes and versions of Application, OS and Bootloader.

Header	Command	Flags	Request	CRC32
13 bytes	H29	1 byte	n-bytes request	CCITT-32

Where

Field Name	Description
Flags	Is a set of flags used to inform about the compression of the request.
Request	Request block. Request data within it may be compressed (zip algorithm) or plain, according to the isRequestCompressed flag. For detailed explanation of the plain data see **System Information Request**.

The following table explains the meaning of the flags byte.

7 (MSb)	6	5	4	3	2	1	0 (LSb)
RFU	RFU	RFU	RFU	RFU	isRequestCompressed	isReplyCompressed	RFU

Where

Bit Name	Position	Description
RFU	0	Reserved for future use.
isReplyCompressed	1	The data contained in the reply will be compressed
isRequestCompressed	2	The request block is compressed.
RFU	3-7	Reserved for future use.

The server may reply in the following ways:

- Negative Acknowledge or NACK (H12) (see above) with no status bytes. The server states that an error occurred while decoding the packet.
- Data Chunk Transmission (H21) (see above). The server sends data about the system information according to **System Information Response**. When the request is submitted without authentication, the server disconnects the consumer after the completion of the sending of data.

3.16 Update Clock Request (H30)

After authentication it's possible to update the system time with the following command

Header	Command	Updated time	CRC32
13 bytes	H30	time value	CCITT-32

The server may reply to a Data Chunk Transmission in two ways:

- Positive Acknowledge or ACK (H11) (see above). The server states that the update is successful.
- Negative Acknowledge or NACK (H12) (see above). The server states that an error occurred during the transmission. Clock update is not effective.

Please note that the reply may require several seconds before being sent because updating the system clock may imply stopping the auto-acquisition and restoring the auto-acquisition status to the original value.

3.17 Open Connection (H50)

This is sent by the server to the consumer specified address to initiate a connection (push mode).

Header	Command	Flags	Request	CRC32
13 bytes	H50	1 byte	n-bytes request	CCITT-32

Where

Field Name	Description
Flags	Is a set of flags used to inform about the compression of the request.
Request	Request block. Request data within it may be compressed (zip algorithm) or plain, according to the isRequestCompressed flag. For detailed explanation of the plain data see **Open Connection**.

The following table explains the meaning of the flags byte.

7 (MSb)	6	5	4	3	2	1	0 (LSb)
RFU	RFU	RFU	RFU	RFU	isRequestCompressed	RFU	RFU

Where

Bit Name	Position	Description
RFU	0-1	Reserved for future use.
isRequestCompressed	2	The request block is compressed.
RFU	3-7	Reserved for future use.

3.18 Keep Alive (H51)

This message is periodically sent by the consumer to the server that initially created the connection via the Open Connection (H50) message.

Header	Command	Flags	CRC32
13 bytes	H51	1 byte	CCITT-32

Where

Field Name	Description
Flags	All flags are reserved for future use.

4 DATA FORMATS

4.1 Time Format

The following table describes how to compose the fields related to time. All values are big endian. The time range is the same of the .NET DateTime struct, so the time range is 12:00 January 1st 0001 to 11:59:59 December 31 9999.

Year	Month	Day	Hour	Minute	Second
2 byte	1 byte	1 byte	1 byte	1 byte	1 byte

E.g.: The time September 29th 2009, 08:59:27 AM becomes

Year	Month	Day	Hour	Minute	Second
2009	September	29th	08	59	27
H07D9	H09	H1D	H08	H3B	H1B

4.2 String Format

Where a field (e.g.: user name) requires a string, the string is the zero-terminated UTF-8 representation of that string, being the first byte the leftmost in the resulting array.

E.g.: The string niño becomes:

Unicode Char	n	i	ñ		o	Zero-termination
UTF-8	H6E	H69	HC3	HB1	H6F	0

5 AN AUTHENTICATION EXAMPLE

This example describes the authentication handshake between a server (e.g.: **TelevisCompact**) and a client (e.g.: a Consumer).

Let the username be niño and the password españa and the transaction time 29th September 2009, 08:59:27. The client connects to the server with an authentication request message:

ServiceType	Version	SendingTime							Length				Command	CRC32			
44	01	07	D9	09	1D	08	3B	1B	00	00	00	12	41	9C	97	70	65

The server replies with an Authetication Challenge message. For the sake of simplicity let the random byte vector be composed by just two bytes [0x07 0x25].

ServiceType	Version	SendingTime							Length				Command	Random		CRC32			
44	01	07	D9	09	1D	08	3B	1B	00	00	00	14	42	07	25	2A	3E	C3	B3

The client then computes the output (PW-SHA1) of the SHA-1 algorithm using the UTF-8 representation of the password, attaching it to the random bytes extracted from the challenge message:

Random		PW-SHA1 (SHA-1 applied to españa)																	
07	25	96	33	DC	28	E8	C0	16	9A	D8	E4	ED	22	9D	6D	8C	3A	03	09	09	C5

After that client computes the output (cryptographic reply) of the SHA-1 algorithm using the previous vector as an input, attaching to the end of it the zero-terminated UTF-8 representation of the username:

Cryptographic reply (SHA-1 applied to Random + PW-SHA1)																			Username (niño)					
9C	F5	F8	9F	BC	CD	DC	CA	9F	04	EC	9B	81	C3	30	D5	62	C4	3E	41	6E	69	C3	B1	6F	00

This vector is inserted into the authentication response as follows:

ServiceType	Version	SendingTime							Length				Command	Cryptographic Reply	Username	CRC32			
44	01	07	D9	09	1D	08	3B	1B	00	00	00	2C	43	20 bytes	6 bytes	AB	C3	A2	C7

If the authentication is successful the server replies with a positive acknowledge:

ServiceType	Version	SendingTime							Length				Command	CRC32			
44	01	07	D9	09	1D	08	3B	1B	00	00	00	12	11	F7	FC	21	91

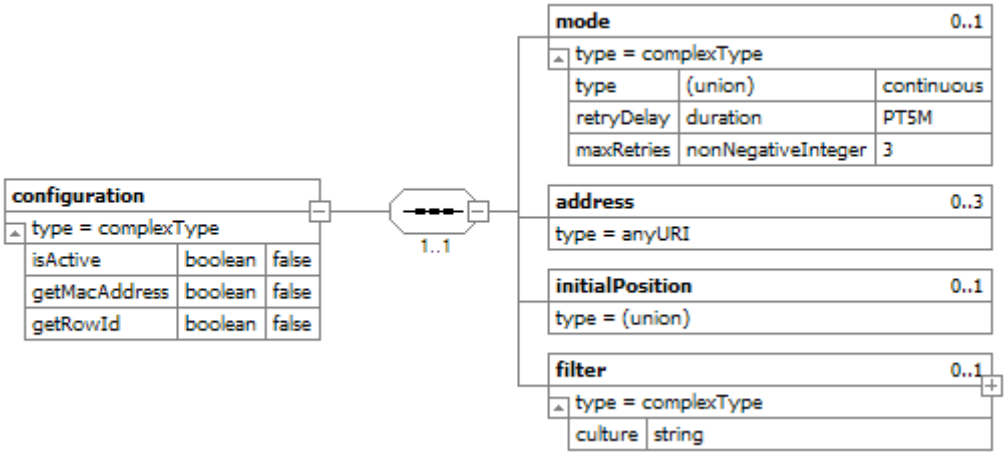
After that, the client may send a new message.

6 DATA TRANSFER XML FORMATS

Queries and data through the Consumer-System connection are XML *messages* wrapped in a binary protocol (see **Errore. L'origine riferimento non è stata trovata.**, **Errore. L'origine riferimento non è stata trovata.**). Furthermore, the *data transfer configuration* used by the **push** mode could be exported to or updated via an XML file. This section describes the XML format for **configuration**, *request query messages* and *retrieved historical data*.

6.1 Data Transfer Configuration

Data transfer configuration could be defined using an XML file; its XML Schema Definition (XSD) is described here.



- All direct attributes and elements are optional. If missing, a default behavior or value is used.
- The *data transfer configuration* may be activated (data are sent to the Consumer at specified time intervals) or deactivated (no data is pushed to the Consumer) using its *isActive* attribute. The default for this attribute is false. The other two optional attributes have this meaning:
- *getMacAddress*: is a boolean attribute that, if set to true, adds the System's MAC address to the retrieved data. If this attribute is missing, then it is assumed to be false.
- *getRowId*: is boolean attribute that, if set to true, adds a unique identifier to each data row in the retrieved data. If this attribute is missing, then it is assumed to be false.
- The *mode* element defines the triggering condition and retries policy for the data push.
- Its *type* attribute could be valued with either the continuous string (meaning that data are sent at each **synchronous save**), or with a time span defining the pace at which data are sent. Its default is continuous.
- *retryDelay* a non-negative duration representing the amount of time to wait before attempting a retry on a failed transmission. Its default is 5 minutes (PT5M).
- *maxRetries* is a non-negative integer number defining how many retry could be attempted. Its default is 3.
- If the *mode* element is missing, the default behavior is that of a continuous push with 5 minutes of retry delay and at most 3 retries.
- Any *address* element is a unique resource identifier (URI) defining address and port to be used for pushing data. Usually, at least one must be defined, and the first address is the primary target for the push whilst subsequent one(s) are considered backup addresses. If, at the time of transfer, the primary target is not available, the 2nd and 3rd one (if defined) are attempted before considering the transfer failed.
- If no *address* is defined, the **configuration** is considered not active independently by the *isActive* attribute.
- The *initialPosition* element could be valued as unchanged, oldest, newest, or a timestamp. It makes sense only before the first data transfer or after a configuration upgrade and defines the initial datum to be sent according to the following table. Default is unchanged.

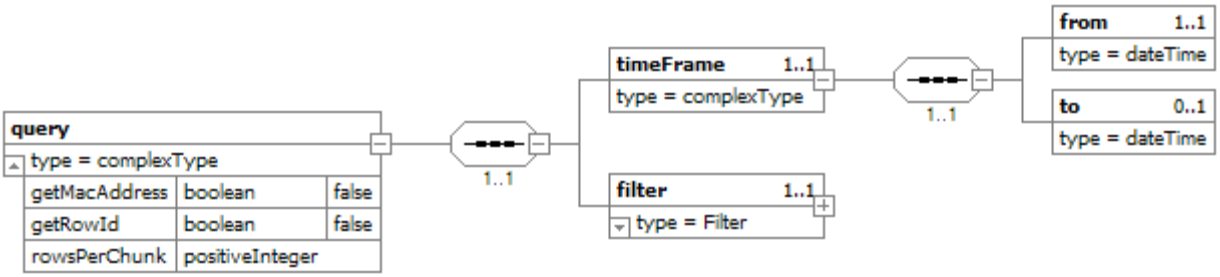
initialPosition	Semantics
unchanged	Initial position is not reset: the next data to be transferred will start from the oldest, still unsent, datum.
oldest	Initial position is set to oldest: the next data to be transferred will start from the

	oldest datum the system has memory of.
newest	Initial position is set to newest: the next data to be transferred will start from the next datum that will be saved since now.
2009-08-15T16:00:00	Initial position is set to August, 15th 2009 at 16 o'clock: the next data to be transferred will start from the datum that has been saved immediately after the given timestamp.
2999-12-31T23:59:59	Initial position is set to December, 31st 2999 just before midnight: the system will transfer no data until a datum is saved with a timestamp that follows the given one.

6.2 Request Query Message

A *request query message* is an XML document satisfying the following XML Schema Definition (XSD). It is used in **pull** mode by wrapping it in a specified command (see **Configuration Request (H20)**, above).

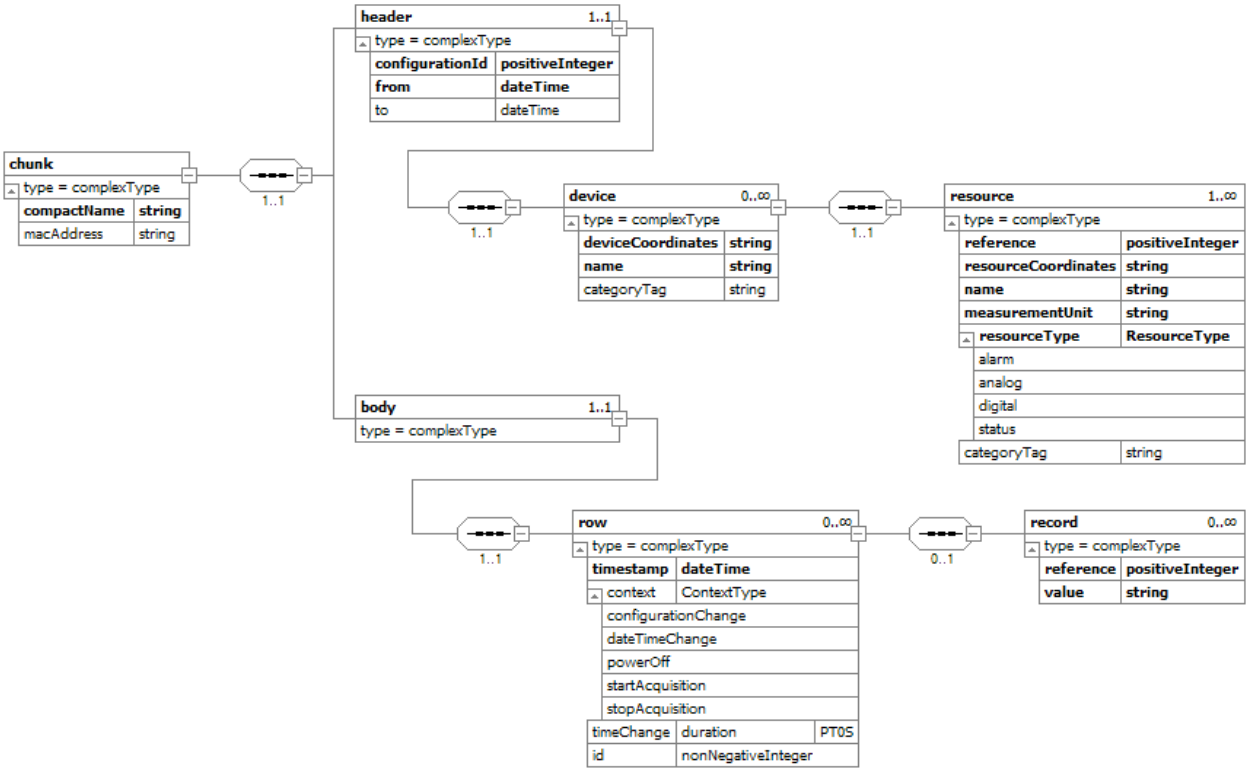
- The *query* element has two, optional, attributes:
- *getMacAddress*: is a boolean attribute that, if set to true, adds the System's MAC address to the retrieved data. If this attribute is missing, then it is assumed to be false.
- *getRowId*: is boolean attribute that, if set to true, adds a unique identifier to each data row in the retrieved data. If this attribute is missing, then it is assumed to be false.
- *rowsPerChunk*: defines the maximum number of rows a chunk can be composed of. If this attribute is missing, then a default value defined via a system generic setting is used (such setting's default is 100).



- The *filter* element refers to the schema definition specified in **Adaptive Filter Evaluation**, below. The *timeFrame* element has two children:
- *from*: is the timestamp of the first (oldest) datum of the query.
- *to*: is an optional element representing the last (newest) datum of the query. If missing, then all data since *from* timestamp is retrieved.
- Both timestamp must be expressed in the TelevisCompact local time.

6.3 Retrieved Historical Data

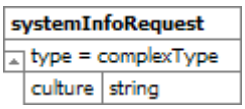
Historical data is retrieved in chunks, in order to avoid the generation of bulk *messages*. A single chunk is an XML document satisfying the following XML Schema Definition (XSD). It is used both in pull mode and push mode by wrapping it in a specified command (see Historical Data Request (H23), above).



- A chunk refers to a single **site configuration** and is split in *header* and *body*. The *chunk* element bears information about the specific plant via the *compactName* attribute or, if requested, the *macAddress* attribute.
- The *header* element collects all the **devices** and **resources** the **adaptive filter** has selected according to either the *request query message* (**pull** mode) or *data transfer configuration* (**push** mode). Header contains an attribute that uniquely identifies the site configuration it refers to (*configurationId* attribute), the starting validity time of the specified configuration (*from* attribute) and, possibly, the ending validity time of the specified configuration (*to* attribute). If *to* attribute is missing, then the site configuration is the active one (it is not yet expired).
- **Devices** and **resources** are optionally labeled with a **category tag** (if one has been defined for them).
- **Resources** have a *reference* attribute, whose purpose is to link their definition to their values in the *body* section. The *resourceType* attribute tells if the resource is an Analog, Digital, Status or Alarm one.
- The *body* element collects a list of *row* elements. Each *row* has a *timestamp* that refers to all the values of the row. If requested a unique identifier among the rows of a plant can be retrieved (optional *id* attribute). A row can be either a context change one or a data one.
- A context change *row* has no data, but its *context* attribute defines a global event happened on the TelevisCompact site. The *context* attribute may assume these values:
- *startAcquisition*: acquisitions have been started.
- *stopAcquisition*: acquisitions have been stopped.
- *powerOff*: system has been restarted (either willingly or by a power failure).
- *dateTimeChange*: system time has been changed.
- *configurationChange*: site configuration has been changed. When this row is present, its *timestamp* and *id* are the same of the row that immediately follows it.
- The *timeChange* attribute, when present, represent the amount of time of a system time change, and implies the *context* attribute is valued as *dateTimeChange*.
- A data *row* has at least one *record*. A *record* is a couple *reference/value* telling that the **resource value** of the referenced **resource** has been changed. The number of *records* of a *row* may be lower than the number of **resources** in the *header*; order is not guaranteed.

6.4 System Information Request

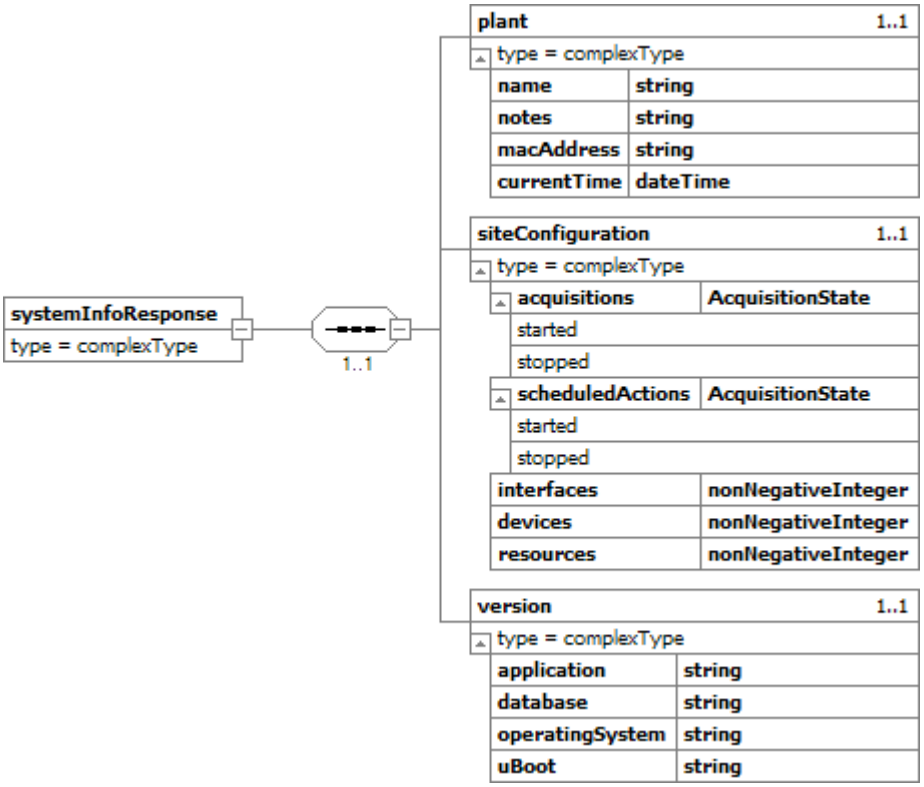
A *system information request* message signals the TelevisCompact to respond with a *system information response*.



- *culture* is an optional string that defines the culture the response will be formatted into. It is in the form <languagecode>-<country/regioncode> according to *Microsoft® specifications* based onto ISO 639-1 and ISO 3166 standards. If no culture is provided, the default system culture will be used.

6.5 System Information Response

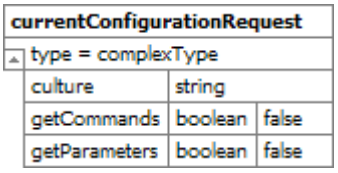
A *system information response* contains data about the TelevisCompact system.



- *plant* element contains the plant *name*, *notes*, its MAC address and the current date/time the Compact is set to.
- *siteConfiguration* contains overall information about the current site configuration: the *acquisitions* state (either started or stopped), the scheduled actions state (either started or stopped), and the total number of *interfaces*, *devices* and *resources*.
- *version* bears information about the *application* version, the *database* version, the OS version and the UBoot last compatible version.

6.6 Current Configuration Request

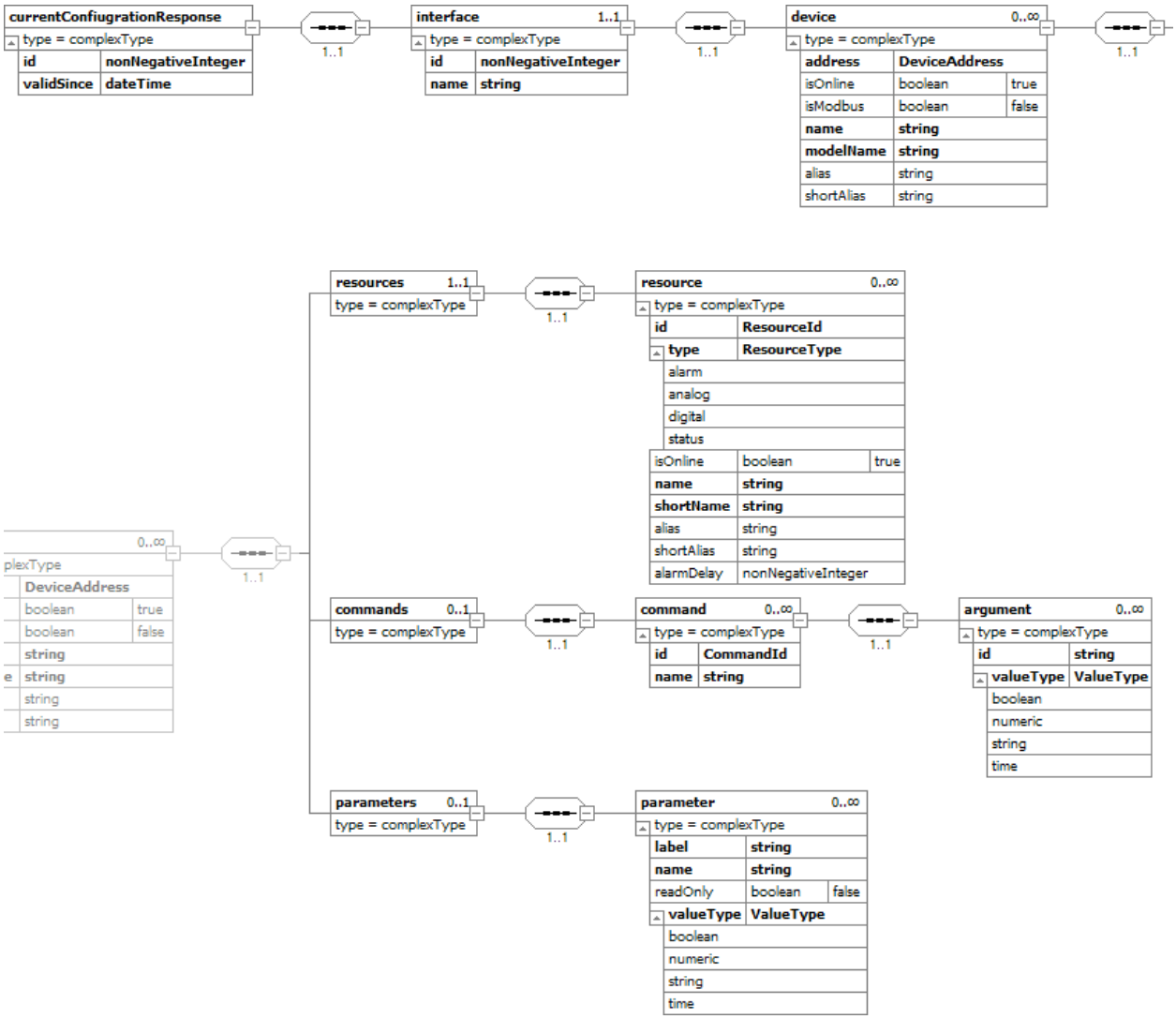
A *current configuration request* message describes the amount of information that the user requires to the TelvisCompact regarding the current site configuration.



- *culture* is an optional string that defines the culture the response will be formatted into. It is in the form <languagecode>-<country/regioncode> according to *Microsoft® specifications* based onto ISO 639-1 and ISO 3166 standards. If no culture is provided, the default system culture will be used.
- *getCommands* is an optional flag that defaults to false. If true, the response will include the list of supported commands for each device.
- *getParameters* is an optional flag that defaults to false. If true, the response will include the list of supported parameters for each device.

6.7 Current Configuration Response

A *current configuration response* message contains information about the current site configuration, and its level of details depends on the specified *current configuration request*.



The *currentConfigurationResponse* element always bears the following attributes:

- *id* is the configuration ID number.
- *validSince* is the timestamp representing the first instant of the configuration's validity interval.

There is an interface element for each interface (either physical or logical) defined in the current configuration. It has the following attributes:

- *id* is the interface ID number.
- *name* is the textual representation of the interface settings (e.g.: COM2 or 192.168.0.1).

There is a device element for each device (either physical or logical) defined in the interface. It has the following attributes:

- *address* is the device address expressed in the FAA:dEA format (satisfying the (1[0-5]|0?[\d]):(1[0-5]|0?[\d]) regular expression); e.g.: 00:00, 03:12, 14:14.
- *isOnline* is optional and defaults to true; it is present and its value is false only if the device has been put offline by the user.
- *isModbus* is optional and defaults to false; it is present and its value is true only if the device is a Modbus one.
- *name* is the device name (computed by TelevisCompact in order to be unique in the entire configuration).
- *modelName* is the name of the device model (e.g.: ID974 LX, EWCM 9000).
- *alias* is optional; it is present only if the user has defined an alternative name for the device.

- *shortAlias* is optional; it is present only if the user has defined an alternative short name for the device.

The *device* element contains at least a *resources* element child and, according to the *current configuration request* settings, an optional *commands* element and/or an optional *parameters* element.

resources contains several *resource* elements, one for each resource (either physical or logical) defined in the device. A *resource* element bears the following attributes:

- *id* is the resource ID, satisfying the [A-Z]{3}\d{5}(:\d+)?(-.+)? regular expression (e.g.: INP40000-1, STA00063, ALM40166-ext, STA00327:12, ALM40091:5-2nd).
- *type* is one of the following values: alarm, digital (either a digital input or a digital output), analog (either an analog input or an analog output), status; and represents the resource typology.
- *isOnline* is optional and defaults to true; it is present and its value is false only if the resource has been put offline by the user.
- *name* is the resource name (translated according to the culture defined in the *current configuration request*).
- *shortName* is the resource's short name (a culture-invariant short string computed on the resource ID).
- *alias* is optional; it is present only if the user has defined an alternative name for the resource.
- *shortAlias* is optional; it is present only if the user has defined an alternative short name for the resource.
- *alarmDelay* is optional; it is present only if the resource is of alarm type and only if its value differs from the default alarm delay defined in the system (usually 0 minutes). It represents the number of minutes of delay the user set for the resource.

commands is optional, and (when present) contains several *command* elements, one for each command the device supports. A *command* element bears the following attributes and children elements:

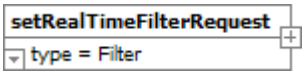
- *commandId* is the command ID, satisfying the [A-Z]{3}\d{5} regular expression (e.g.: FNC00001, FNC00008).
- *name* is the command name (translated according to the culture defined in the *current configuration request*).
- A series of 0 to many *argument* elements representing the command's arguments. Each *argument* has a textual *id* and a *valueType* (either boolean, numeric, string or time).

parameters is optional, and (when present) contains several *parameter* elements, one for each parameter the device exposes. A *parameter* element bears the following attributes:

- *label* is a string identifying the parameter (usually equals to the ones visible on the device's display; e.g.: SEt, HAL, dIF).
- *name* is the parameter name (translated according to the culture defined in the *current configuration request*).
- *readOnly* is optional and defaults to false; it is present and its value is true if the parameter cannot be written.
- *valueType* could be either boolean, numeric, string or time; represents the parameter value type.

6.8 Set Real Time Filter Request

A *set real time filter request* is a message that asks the TelevisCompact to store a filter on the current site configuration to be used when sending real time data.



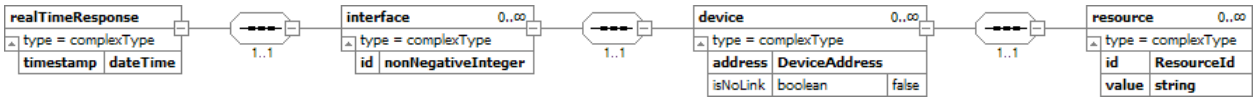
It is composed of a unique element of the *Filter* type (see **Errore. L'origine riferimento non è stata trovata. Errore. L'origine riferimento non è stata trovata.**, **Errore. L'origine riferimento non è stata trovata.**, for specifications). The filter is then persisted on the TelevisCompact until another sound *setRealTimeFilterRequest* overwrites it.

The file is saved (and could be retrieved via Web Server) at the following address: `<compact's address>/bin/RealTimeServiceFilter.xml`.

A backup file is copied at `<compact's address>/bin/RealTimeServiceFilter.backup.xml`.

Real Time Response

A real time response contains information about the current values of the current configuration's resources that have been filtered using a *set real time filter request*.



- *timestamp* indicates the moment the data has been retrieved.

There is an *interface* element for each filtered interface.

- *id* is the number that identifies the interface.

There is a *device* element for each filtered device belonging to the *interface*.

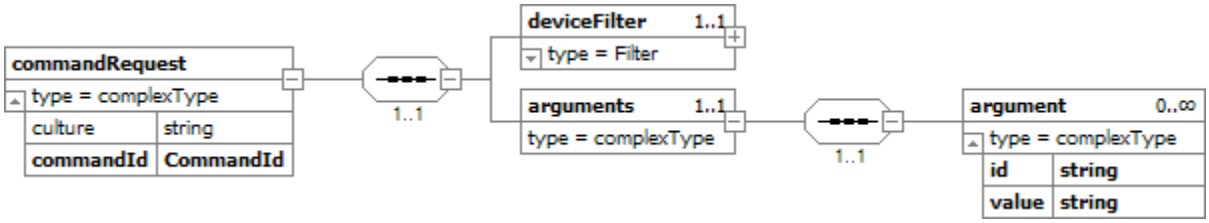
- *address* is the device address expressed in the FAA:dEA format (satisfying the (1[0-5]|0?[\d]):(1[0-5]|0?[\d]) regular expression); e.g.: 00:00, 03:12, 14:14.
- *isNoLink* is an optional boolean value that defaults to false. If present its value is true and means that, at the moment the response is produced, the specified device is in No-Link state.

There is a *resource* element for each filtered resource belonging to the *device*.

- *id* is the resource ID, satisfying the [A-Z]{3}\d{5}(:\d+)?(-.+)? regular expression (e.g.: INP40000-1, STA00063, ALM40166-ext, STA00327:12, ALM40091:5-2nd).
- *value* is the textual representation of the resource's last known value, formatted according to the system culture defined for the TelevisCompact. It could bear the special values of ??? (if the resource value is still unknown), --- (if the resource's device is in No-Link) or * * (if the resource value is erroneous).

6.9 Device Command Request

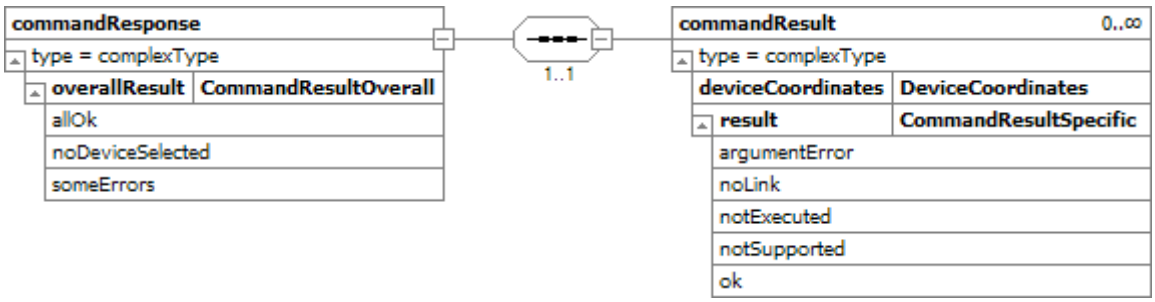
A *device command request* message contains a filter defining the set of devices the command must be performed onto, along with the command to be performed and its optional arguments.



- *culture* is an optional string that defines the culture the arguments must be parsed into. It is in the form <languagecode>-<country/regioncode> according to *Microsoft® specifications* based onto ISO 639-1 and ISO 3166 standards. If no culture is provided, the default system culture will be used. Note that this culture could be different from the one defined in the *deviceFilter* (e.g.: a filter on the resources' English names could be created while the command's arguments should be parsed in Spanish).
- *commandId* is a string that should satisfy the [A-Z]{3}\d{5} regular expression (e.g.: FNC00001, FNC00008) and identifies the command to be performed; command IDs usually start with "FNC".
- Each *argument* bears an *id* (a string of text) and a *value* to be set to (a string of text that will be parsed according to the request's *culture*).
- *deviceFilter* is an adaptive filter (see **Errore. L'origine riferimento non è stata trovata.** **Errore. L'origine riferimento non è stata trovata.**, **Errore. L'origine riferimento non è stata trovata.**) that, when applied to the current configuration, retrieves a subset of its resources. The command will be applied to any device whose at least one resource appears in the filtered subset.

6.10 Device Command Response

A *device command response* contains information about the successes and/or failures of a *device command request*.



overallResult gives a hint on how well the command performed on the whole set of filtered devices.

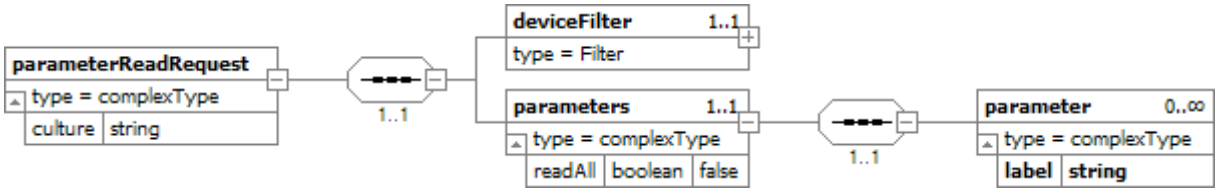
overallResult	Semantics
noDeviceSelected	The request's filter was too strict, thus excluding all devices from the result. In this case the list of *commandResult*s is empty
allOk	At least one device has been selected and the command has been correctly executed on all devices
someErrors	At least one device has been selected and the command execution failed on at least one of them

There will be a *commandResult* element for each device that the filter selected, and its *deviceCoordinates* attribute specifies the device the *result* is about.

result	Semantics
ok	Command has been successfully executed on the specified device
notSupported	Command has not been executed because the specified device doesn't support it
noLink	Command has not been executed because the specified device was in No-Link state
notExecuted	Command has not been executed for an unknown reason
argumentError	Command has not been executed because the provided arguments were wrong in number, have a wrong ID, or for a syntactic error in their values

6.11 Parameter Read Request

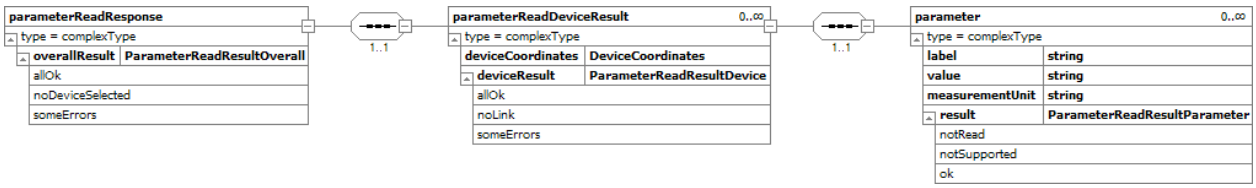
A *parameter read request* message contains a filter defining the set of devices whose parameters must be read, along with the list specific parameters to read.



- *culture* is an optional string that defines the culture the response will be formatted into. It is in the form <languagecode>-<country/regioncode> according to *Microsoft® specifications* based onto ISO 639-1 and ISO 3166 standards. If no culture is provided, the default system culture will be used.
- *deviceFilter* is an adaptive filter (see **Errore. L'origine riferimento non è stata trovata.** **Errore. L'origine riferimento non è stata trovata.**, **Errore. L'origine riferimento non è stata trovata.**) that, when applied to the current configuration, retrieves a subset of its resources. The parameters will be read from any device whose at least one resource appears in the filtered subset.
- *readAll* attribute of the parameters element is an optional flag that defaults to false. If set to true it implies that the request is intended to read all the device parameters (the underlying list of *parameter* elements is expected to be empty and thus is ignored).
- Each *parameter* element defines a parameter the request is intended to read, specified by its label.

6.12 Parameter Read Response

A *parameter read response* contains information about the successes and/or failures of a *parameter read request* along with the values of the read parameters.



overallResult gives a hint on how well the read parameter session performed on the whole set of filtered devices.

overallResult	Semantics
noDeviceSelected	The request's filter was too strict, thus excluding all devices from the result. In this case the list of *parameterReadDeviceResult*s is empty
allOk	At least one device has been selected and all the requested parameters have been correctly read on all devices
someErrors	At least one device has been selected, but at least one of them encountered problems with at least one parameter reading operation

There will be a *parameterReadDeviceResult* element for each device that the filter selected, and its *deviceCoordinates* attribute specifies the device the parameters belong to. The *deviceResult* gives a hint on how well the read parameter session performed on the specified device.

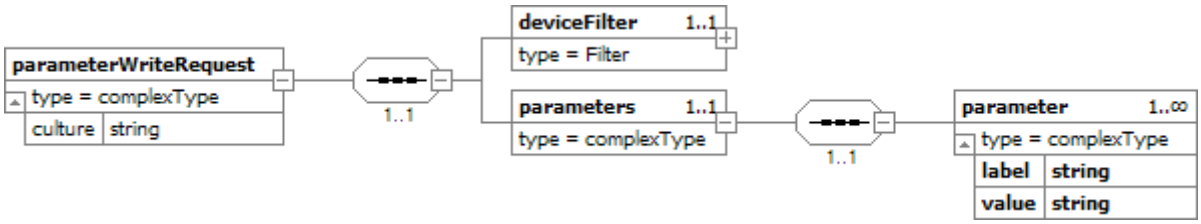
deviceResult	Semantics
allOk	All requested parameters have been successfully read from the device
noLink	Requested parameters have not been read because the specified device was in No-Link state; in this case the *parameter* list will be empty
someErrors	At least one parameter has not been read for some reason

There will be a *parameter* element for each *label* specified in the request. *value* contains a textual representation of the parameter value (rendered in the *culture* specified with the request), and *measurementUnit* contains the measurement unit of the specified parameter. *result* specifies the success or failure of the reading.

result	Semantics
ok	Parameter has been correctly read
notSupported	Parameter has not been read because the device has no parameter with the specified *label*; in this case the *value* attribute will be an empty string
notRead	Parameter has not been read for an unknown reason (e.g.: a timeout on the network); in this case the *value* will be an empty string

6.13 Parameter Write Request

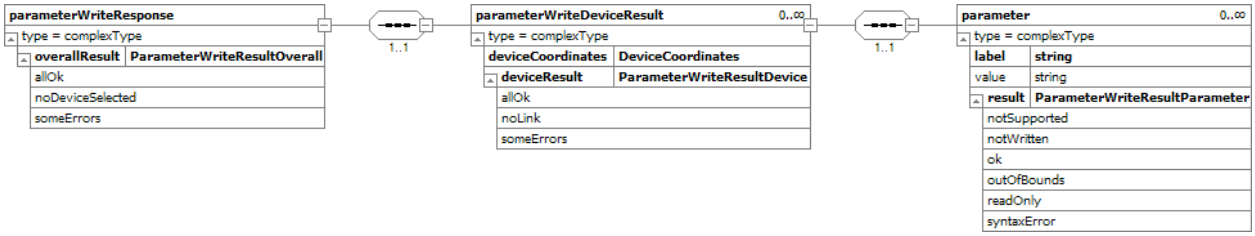
A *parameter write request* message contains a filter defining the set of devices whose parameters must be written, along with the list specific parameters and values to write.



- *culture* is an optional string that defines the culture the response will be formatted into and the culture values to be set are parsed from. It is in the form <languagecode>-<country/regioncode> according to *Microsoft® specifications* based onto ISO 639-1 and ISO 3166 standards. If no culture is provided, the default system culture will be used.
- *deviceFilter* is an adaptive filter (see **Errore. L'origine riferimento non è stata trovata.** **Errore. L'origine riferimento non è stata trovata.**, **Errore. L'origine riferimento non è stata trovata.**) that, when applied to the current configuration, retrieves a subset of its resources. The parameters will be set on any device whose at least one resource appears in the filtered subset.
- Each *parameter* element defines a parameter the request is intended to write (specified by its *label*) along with the *value* to set.

6.14 Parameter Write Response

A *parameter write response* contains information about the successes and/or failures of a *parameter write request*.



overallResult gives a hint on how well the write parameter session performed on the whole set of filtered devices.

overallResult	Semantics
noDeviceSelected	The request's filter was too strict, thus excluding all devices from the result. In this case the list of *parameterWriteDeviceResult*s is empty
allOk	At least one device has been selected and all the requested parameters have been correctly written on all devices
someErrors	At least one device has been selected, but at least one of them encountered problems with at least one parameter writing operation

There will be a *parameterWriteDeviceResult* element for each device that the filter selected, and its *deviceCoordinates* attribute specifies the device the parameters belong to. The *deviceResult* gives a hint on how well the write parameter session performed on the specified device.

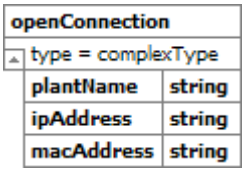
deviceResult	Semantics
allOk	All requested parameters have been successfully written on the device
noLink	Requested parameters have not been written because the specified device was in No-Link state; in this case the *parameter* list will be empty
someErrors	At least one parameter has not been written for some reason

There will be a *parameter* element for each *label* specified in the request. *value* contains a textual representation of the parameter value after the write operation (rendered in the *culture* specified with the request). *result* specifies the success or failure of the writing operation.

result	Semantics
ok	Parameter has been correctly written
notSupported	Parameter has not been written because the device has no parameter with the specified *label*
outOfBounds	Parameter has not been written because otherwise its value would be set outside its current bounds
readOnly	Parameter has not been written because it is protected from write
syntaxError	Parameter has not been written because the specified value couldn't be parsed (in the specified culture) as a consistent value for its type
notWritten	Parameter has not been written for an unknown reason (e.g.: a timeout on the network)

6.15 Open Connection

An *open connection* message is sent by the **TelevisCompact** to initiate a connection to a remote client.



It brings information about the name of the plant (*plantName*), the IP address he owns on its local network (*ipAddress*), and its MAC address (*macAddress*).

7 ADAPTIVE FILTERS BEHAVIOR

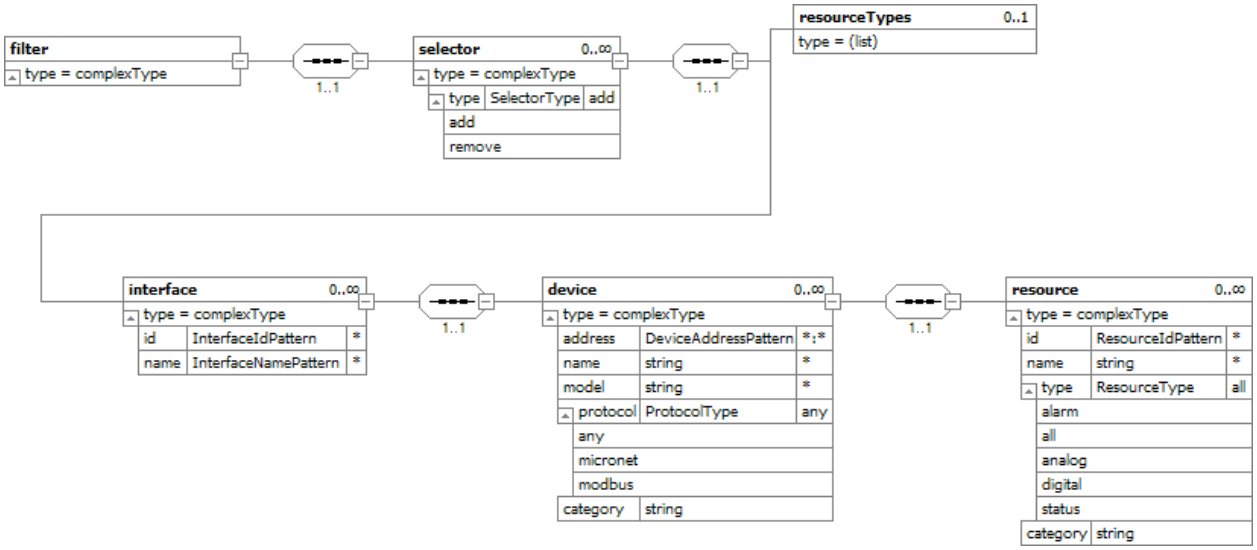
Most of the services exposed need to know which are the **resources** whose data are to be transferred, the devices a command must be performed onto, etc. **Adaptive filters** represent a language to identify **interfaces**, **devices** and **resources** independently by the specific **site configuration** (using **coordinates** and **calculated names**) and in a compact way (by mean of wildcards and defaults).

- To provide a usable set of **interface ids**, **device coordinates** and **resource coordinates**, the **adaptive filter** must be evaluated against a specific **site configuration** and, possibly, a specific culture (language).

7.1 Adaptive Filter Evaluation

Evaluation starts with an "empty pool" of **resources**. The *filter* element has an optional *culture* attribute that could indicate the language **resources** must be translated into (filter on **resource** names may need it).

- Then, each **adaptive selector** is applied in order (order matters). An **adaptive selector** retrieves a subset of the specified **site configuration**'s **resources** and then its **type** specifies if this subset must be added to o removed from the filter's pool (if type is omitted, the selector is considered to be additive).
- After all **adaptive selectors** have been evaluated and the pool has been consolidated, all **resources** in the pool are being retrieved along with the minimum set of **devices** and **interfaces** the **resources** belong to.



7.2 Adaptive Selector Evaluation

An **adaptive selection** works in two ways.

- Selects **resources** according to their **resource type** (each of the specified site configuration's **resources** whose type is in the indicated list falls into the selection subset), and
- Selects **interfaces**, **devices** and **resources** in a hierarchical way.
- The result of both selections enters the subset (that will be either added or removed from the pool according to the **selector**'s type).

7.3 Hierarchical Evaluation

Selection of **interfaces**, **devices** and **resources** comes in this order and through successive refinements.

- What really matters is the final set of **resources**, so **interface** and **device selectors** are only a mean to narrow the selection scope.
- Each of the three types of **adaptive selectors** (**interface**, **device** and **resource**) has a twofold filter possibility: via id (or address) and via its **calculated name**. These filters are applied with AND logic, meaning that an item must satisfy both filters to be included in the selection subset.
- Both filters can use wildcards characters:
- A question mark ? means *any single character or no character*.
- An asterisk * means *any substring or no substring*.

- The escape character to represent question marks, asterisks and the escape characters themselves is the backslash \. So, the escaped characters are: \?, * and \\.
- If one of these filters is not specified, the default for it is * (selects any item).

7.3.1 Adaptive Interface Selector Evaluation

A list of **adaptive interface selectors** may be defined within an **adaptive selector**, and each **interface selector** is evaluated. The items selected by all **interface selectors** are combined with OR logic, meaning that all of them will be included in the selection subset.

An **interface** may be selected by its name, by its id, or both.

- A filter on name assumes that the **interface** name is "COMx" for a **serial network interface** (where 'x' is the COM number), "aaa.bbb.ccc.ddd" for a **LanAdapter** one (the representation of the LanAdapter IP address) and "Logical" for the TelevisCompact logical interface. Filter is interpreted as well-formed if it satisfies this regular expression: \s*(*|Logical|COM([\d\?]+|*)|((*|[2\?][5\?][0-5\?]|[2\?][0-4\?][\d\?]|[0-1\?][\d\?][\d\?]|[\d\?][\d\?]|[\d\?])(\.(*|[2\?][5\?][0-5\?]|[2\?][0-4\?][\d\?]|[0-1\?][\d\?][\d\?]|[\d\?][\d\?]|[\d\?])){3}))\s*.
- A filter on id assumes the id is the textual representation of a non-negative integer value. The regular expression for this filter is: \s*(*|\d+)\s*.

7.3.2 Adaptive Device Selector Evaluation

A list of **adaptive device selectors** may be defined within an **adaptive interface selector**, and each **device selector** is evaluated. The items selected by all **device selectors** are combined with OR logic, meaning that all of them will be included in the selection subset.

A **device** may be selected by its name, its model, its address, its protocol or a combination (in AND logic) of the four.

- A filter on name uses the **device calculated name**.
- A filter on model uses the **device model**.
- A filter on address assumes the **device address** is "xx:yy" (family and low-address, with xx and yy between 00 and 15). The regular expression for this filter is: \s*(*|([0\?]?[\d\?]|[1\?][0-5\?]|*):([0\?]?[\d\?]|[1\?][0-5\?]|*))\s*.
- A filter on protocol may be micronet, modbus or any. The default is any.

A **device selector** may be accompanied by additional information: the **category tag**. Category tag is a label that, if the selector is additive, will be applied to each **device** the selection has found.

7.3.3 Adaptive Resource Selector Evaluation

A list of **adaptive resource selectors** may be defined within an **adaptive device selector**, and each **resource selector** is evaluated. The items selected by all **resource selectors** are combined with OR logic, meaning that all of them will be included in the selection subset.

A **resource** may be selected by its name, by its id, by its type or any combination (in AND logic) of the three.

- A filter on name uses the resource **calculated name**, optionally translated using the specified language (if present) or the **default system language** (if no one has been specified).
- A filter on id assumes the **resource id** is "AAAxxxxx-y:z" (**resource code** with – optionally – placeholder and tiebreaker). The regular expression for this filter is: \s*(*|([A-Z\?]{3}|*)([\d\?]{5}|*)(-([^\-]+?|*))?(:([\d\?]+|*))?)\s*.
- A filter on **resource type** may be alarm, analog, digital, status or any. The default is any.

A **resource selector** may be accompanied by additional information: the **category tag**. Category tag is a label that, if the selector is additive, will be applied to each **resource** the selection has found.

7.4 XML Structure and Examples

Data transfer configuration (in **push**) is defined with XML structure (whose file form is exported with the **Errore. L'origine riferimento non è stata trovata.** **Errore. L'origine riferimento non è stata trovata.** use case, **Errore. L'origine riferimento non è stata trovata.**); and so is the **data transfer** *request query message* (**pull**). They both needs to define an **adaptive filter**; and the XML structure is the same in both cases.

```xml
<filter culture="en-GB">
  <selector type="add">
    <!-- Selects all the analog and digital resources of any interface and device -->
    <resourceTypes>analog digital</resourceTypes>
    <!-- Selects all the NoLink alarms from any device of the serial interface COM2 and assigns to all of them the "CommunicationFailure" tag -->
    <interface name="COM2">
      <device address="*:*">
        <resource id="ALM00300" category="CommunicationFailure" />
      </device>
    </interface>
  </selector>
</filter>
```

Example 1

- Example 1 shows the XML structure of an **adptive filter** that selects all analog and digital inputs of any **device** under any **interface**, plus the NoLink alarms (id ALM00300) of every **device** under the serial **interface** COM2.
- Data is retrieved with the en-GB culture (British English).
- Absent filters (on **interface id**, **device** name, **device** model, and **resource** name) are considered as *. Absent **resource type** is considered any.
- The filter also permits to label all the NoLink **alarm resources** of the COM2 **interface** (and not others) with a CommunicationFailure **category tag**.

```xml
<filter culture="en-GB">
  <selector>
    <!-- Selects all the resources whose translated name starts with "Analog input" -->
    <interface><device><resource name="Analog input*"/></device></interface>
    <!-- Selects all the resources of all devices of the 11th family laying on any LanAdapter interface belonging to the 192.168.x.x mask -->
    <interface name="192.168.*.*"><device address="11:*"/></interface>
  </selector>
  <selector type="remove">
    <!-- Unselects all digitals and alarms of any interface -->
    <resourceTypes>digital alarm</resourceTypes>
    <!-- Unselects any status of devices whose name contains the word "meat" or the word "ID983 LX" -->
    <interface>
      <device name="*meat*"><resource id="STA*"/></device>
      <device name="*ID983 LX*"><resource id="STA*"/></device>
    </interface>
  </selector>
</filter>
```

Example 2

- Example 2 shows the XML structure of a more complex filter with two **adaptive selectors**.
- The first selector is additive (**type** is missing) and does two things:
- Selects (adds) all the **resources** whose **calculated name** (in British English) starts with Analog input. Because of filters on **interface** and **device** are missing, the **resource** filter's scope is the whole set of **resources**.
- Selects (adds) all the **resources** of any **device** on the **family** 11 that lie beyond any LanAdapter on the 192.168.x.x subnet mask. Because **resource** elements are missing altogether, the selector selects all of them in the given scope.
- After the evaluation of this selector, the pool of filtered **resources** comprehends both "Analog input*" **resources** (any resource whose British English-translated name starts with the Analog input string) and LanAdapter 11th-family-device **resources**.
- The second selector is subtractive (**type** = remove) and does three things:
- Selects (removes from the pool) all the digital input- and alarm-typed **resources**.
- Selects (removes from the pool) all the **resources** whose **id** starts with STA from any **device** whose **calculated name** contains the word meat. These selections could also be accomplished with a filter on **resource type** status.
- Selects (removes from the pool) all the status **resources** from any **device** whose **calculated name** contains the word ID983 LX.
- After the evaluation of the second selector, being this one subtractive the original pool reduces in size.

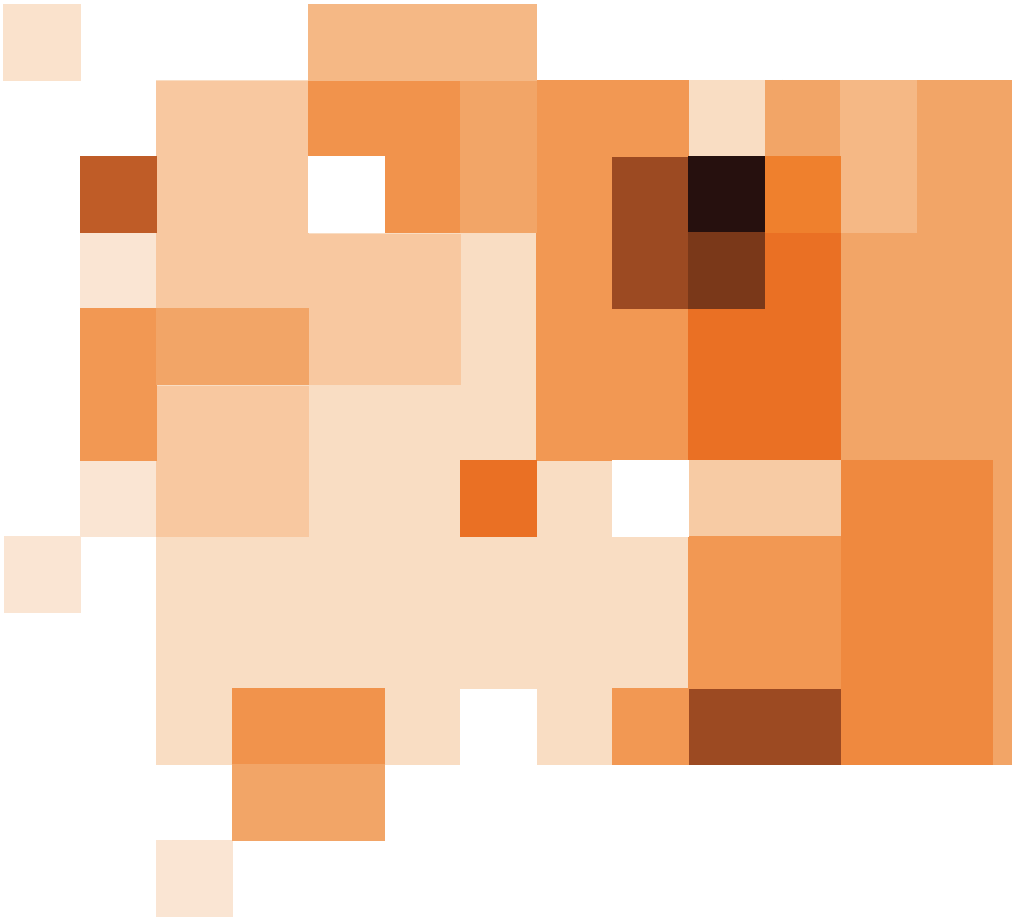
8 ANALITIC INDEX

Eliwell Controls S.r.l.
Via dell' Industria, 15 Zona Industriale Paludi
32010 Pieve d' Alpago (BL) Italy
Telephone +39 0437 986 111
Facsimile +39 0437 989 066

Sales:
+39 0437 986 100 (Italy)
+39 0437 986 200 (other countries)
saleseliwell@invensys.com

Technical helpline:
+39 0437 986 300
E-mail: techsuppeliwell@invensys.com

www.eliwell.it



User Protocol
2011/11/0
Cod: Protocol