OCIT®

Open Communication Interface for Road Traffic Control Systems

Offene Schnittstellen für die Straßenverkehrstechnik

OCIT Outstations

Basic Functions for Field Devices

OCIT-O_Basis_V3.0_D01

OCIT Developer Group (ODG)

OCIT Outstations

Basic Functions for Field Devices

Document: OCIT-O_Basis_V3.0_D01

Issued by: OCIT Developer Group (ODG)

Contact: www.ocit.org

Table of contents

Document history

Version Status	Distribu-tion list	Date	Comment
V3.0 D01	ODG internal	2015-08-06	Draft version of OCIT-O AG5 2.4 Operation identifier Content revised
		2016-12-05	Operating status archive removed
		2018-01-30	Beta version (draft status
		2018-03-14	Specifications KD V3.0
V3.0 A01	PUBLIC	2018-03-15	For OCIT-C V3.0 ODG Homepage

Specifications

The **OCIT Outstations configuration document OCIT-O KD V3.0** contains an overview of all the specifications whose copyrights are managed by the ODG and arranges versions and revision levels according to:

- associated specifications of the interface "OCIT outstations for traffic signal controllers" with reference to the corresponding OCIT-C specifications,
- gives information on the use of the transmission profiles and
- provides an overview of packages of specifications for interfaces for the use of which a nominal fee is required by ODG

The current issue of the document is published on www.ocit.org.

1 Introduction

Provided in this document are definitions about functions that are available in the traffic signal controllers, but also in traffic measuring points and in other typical field devices used in road traffic control systems in a similar fashion, such as archives or the messages "Door open", "Fault", etc. It is not required that OCIT-outstations-compatible devices support all the functions defined here. They only support those functions that are necessary for the relevant purpose and design. The definitions apply to field devices and control centers.

2 Special definitions

2.1 System time

The traffic signal controllers have local clocks. Their exact setting is made in the field devices, which can use the time-standardization service **NTP version 4 (RFC 591 1305)** of the control center for this. The time-standardization service compensates for the time errors caused by the transmission time between the control center and the field device. For additional definitions see the document OCIT-O rules and protocols.

In addition, the device time can be queried directly by the control center and the control center time can be queried by the field device (see System object field device 4.1.1 und System Object Control Center). These queries are prone to time errors caused by the transmission time between the control center and the field device.

2.2 Detecting faults in the transmission path

Can be supplemented with definitions in the OCIT-O Profile_nn documents.

A "fault in the transmission path" is understood here as a complete failure in the transmission path for several seconds, such as one that might occur if the connection is interrupted ("communication fault") or if there is a power supply failure ("power outage"). Transmission faults are caused by defective system components. Sporadic transmission faults may temporarily lead to comparable fault patterns.

Transmission faults may be caused by:

- Failure of the central computer or the field device
- Failure of the transmission units in the field device or the control center
- Interruption in the transmission path
- Power outage

Transmission faults of this kind are detected if telegrams are missing, while how quickly this can be detected depends on how often telegrams are sent:

- **Detection possibility in the control center:**
  There is no response to the telegrams coming from the control center.
- **Detection possibility in the field device:**
  No telegrams arrive from the control center.

Control telegrams can be sent at regular intervals for detection, and if none are received, the presence of a transmission fault can be concluded. Detection possibilities based on the functions of the transmission device such as carrier monitoring are not listed here because they depend on the type, and the transmission devices for the OCIT outstations are not required.

2.2.1 Distinguishing transmission faults

Transmission faults detected by the field device generates an OCIT outstation message **Communication Fault** (4.2.12) and make their way to the center device via the query of the default message archive (4.2.9).

The OCIT outstation message "Communication Fault" includes "Power Outage" as a cause of the fault. However, there is an option to differentiate the "power outage" from other causes.

Note: Due to relevant technology selected to detect a communication fault or the amount of time between telegrams, the times of origin of the "Communication Fault" messages in the control center and in the field device may differ significantly!

2.2.1.1 Differentiation after the fault is corrected

After the communication fault is corrected, if a power outage was the cause, OCIT outstations messages that make it possible for the control center to pinpoint the cause after the fact are transmitted from the field device to the control center (standard message archive).

- **Power OFF** with details about the time of the outage (4.2.12)
- **Power ON** (4.2.12)
- **Communication ok** (4.2.12)

This allows the control center to detect the cause of the power outage at a later time and to pinpoint the original message "Communication Fault".

2.2.1.2 Differentiation immediately after occurrence

This option allows the field devices to immediately communicate a power outage, thus allowing the control center to immediately narrow down the "Communication Fault" to the fault at hand.

To do so, the field devices require a buffering of the power supply (short-term UPS) in order to continue supplying power to the device components needed for the message for the duration required for the message routine. Two methods are defined for transmitting information about the power outage, which vary in terms of the length of the necessary buffer time for the power supply.

Version a) power outage message via standard message archive

> Message **Power OFF** with information about the time of the failure in the default message archive. Several transactions are needed to collect the message from the standard message archive. Choose the correct length for the buffer time of the field device.

Version b) additional power outage message via event list

> In the event of a power outage, the field device calls up the **EvListe::OnNetzAus()** method in the event destination registered for the standard message archive. The method is called up on the btpplHi channel to prioritize the transmission. The OCIT address of the field device and the previous identifier of the fault are transmitted in the method's parameters. In addition to calling up the method, the message Power OFF is logged in the standard message archive, which is either transmitted immediately or after the power supply returns. The same process identifier is used that is transmitted with **EvListe::OnNetzAus()**. This allows the control center to uniquely assign event and message.
>
> It is up to the manufacturers' discretion whether they want to offer this optional feature, "Detecting a power outage immediately after its occurrence". If this option is offered, the following definition applies: For several seconds, the field device is capable of continuing OCIT outstations communication after an operating power outage and to issue the relevant message. The estimated buffer

times are at around 30 seconds for fixed connections for version a) and at around 5 seconds for version b). For dial-up connections, the estimated necessary buffer time is at least at one minute for both versions. However, the buffer times of the power supply are not defined in the OCIT outstations.

Recommendation for message management in a control center:
To prevent an incorrect interpretation of the time of the power outage, the power outage message archived in the control center should only be derived from the message "Power OFF" from the standard message archive. To get information immediately, the EvListe::OnNetzAus() can be displayed as the status.

2.3 Counting method for numbered elements

- The addressing of numbered items such as signal groups and detectors etc. begins with the index value 1. The Index is not mapped: Index 1 addresses item 1 etc. This ensures that the index value with the number used by the users matches the number of a numbered item.

- The time count starts with the time at 0. Time 0 refers to the first time cycle, from its beginning to its end.

2.4 Operation identifier

SYSJOBID is used to assign messages to operations.

Throughout the entire system, operations are performed from different sources (manually or automatically). For traffic signal controllers, for example, these are central operator activation of a group, an automatic-time function for switching or TA switching via a local traffic-actuated logic (TA). These operations may impact several devices. In order to be able to monitor the operations based on separate messages, operation identifiers are introduced. All messages with the same operation identifier within a few days belong to the same operation.

What this requires is that each **operation initiator** (system component that is able to initiate certain operations) uses a unique number. OCIT outstations uses permanently assigned number ranges. The operation identifier is an integral part of the selected operations. It consists of origin identifier and task number. The origin identifier is a unique tag for a system component that exercises control over operations or is a trigger for other operations. The task number is a number consecutively issued for operations.

Because of the operation identifier

- the field device can organize the internal processes and

- the control center can document the different operators and systems.

The operation identifier must be unique during the maximum expected runtime of a control command.

Field device messages caused by operations of use or change operations adopt the operation identifier of the triggering system component. This way, the operation and response can be documented in the control center.

The task number is always issued by the system component determined by the origin identifier.

Additional details are defined in the definitions for certain field devices:

Mandatory:

It is imperative to use the operation identifier.

Examples of objects that carry the operation identifier:

Control center switch requests, current status messages, remote service, supply transactions, messages. Details can be found in the relevant object definitions.

Rule:

The operation identifier of the system component that exercises control over a switch status is entered, even if the switch status does not change upon switching the task (for example: selection of the signal program),
or the operation identifier of the system component that triggered a process (for example: supply transaction).

Operation identifier:

Operation identifier					Name
Source				Task number	
System name	Origin				
Subsystem	Type	Subtype	In-stance		
1 = Control center 2 = System access 3 = Field device	0 = No de-tails	-	0 to 63 0 to 63 0 to 65535	0 to 65535 0 to 65535 0 to 63	
	1 = ZAUT	0 = No details 1 = Day plan 2 = City timetable 3 - 15 free, number for logic			Automatic time function / control clock (City timetables, e.g. for football matches)
	2 = TA logic	0 = No details 1 - 15 unoccupied, number for logic			Traffic-actuated logic or application programs in field device/control center
	3 = Service accesses	0 = No details 1 = Console integrated 2 = Console separate 3 = Service PC 4 = Special intervention 5 = Blocking 6 - 15 unoccupied, other service accesses			Service accesses via local and central con-soles / PCs
	4 = Trans-mission systems	0 = No details 1 = Outbound switch request 2 = Transmission fault 3 - 15 unoccupied, for more detailed infor-mation on the transmis-sion system component			Faults in the transmis-sion system

Operation identifier					Name
Source				Task number	
System name	Origin				
Subsystem	Type	Subtype	In-stance		
	5 = Monitor-ing	0 = No details 1 = Signal monitoring 2 = Intergreen time monitoring 3 = Conflict monitoring 4 = Minimum green monitoring 5 = Minimum red moni-toring 6 = Cycle time monitor 7 - 15 unoccupied, for more detailed infor-mation on the monitor-ing equipment			Status changes due to internal monitor
	6 = Supply data server	0 - 15 unoccupied			
	7 = Proces-sor data server	0 - 15 unoccupied			
	8 = C2X	0 = No details 1 = DENM 2 - 15 unoccupied			
	9 - 15 unoc-cupied				

Note: "Unoccupied" means: To be kept unoccupied for later developments in the standard and not to be used for project-specific solutions.

Subsystem identifiers:

ID Num-ber:	Subsystem
0	Not defined
1	Control center
2	System access
3	Field device

Formats:

2^{32}				2^0
Subsystem	**Type**	**Subtype**	**Instance**	**Task number**
2 bit	4 bit	4 bit	Control center: 6 bit	Control center: 16 bit
			System access: 6 bit	System access: 16 bit
			Field device: 16 bit	Field device: 6 bit

Examples:

Operation of central automatic time function (ZAUT), control center 0:

1	1	1	0	Task number	Day plan

Operation via central system components, control center 0:

1	3	1	0	Task number	Central operation (manual)

Operation via local, separate console (e.g. manual control panel), field device 317:

3	3	2	317	Task number	Separate console

For example, lamp replacement:

The service engineer replaces lamps on traffic signal controller number 32.

To do this he first switches off the system on site with his service PC, then he replaces the lamps, runs a trial with signal program 1 and finally signs out again:

Subsystem = Traffic signal controller (3)
Type = Service access (3)
Subtype = Service PC (3)
Instance = FNr (32)
Task number (1)

==> SYSJOBID=11 0011 0011 0000000000100000 000001B=0xCCC00801

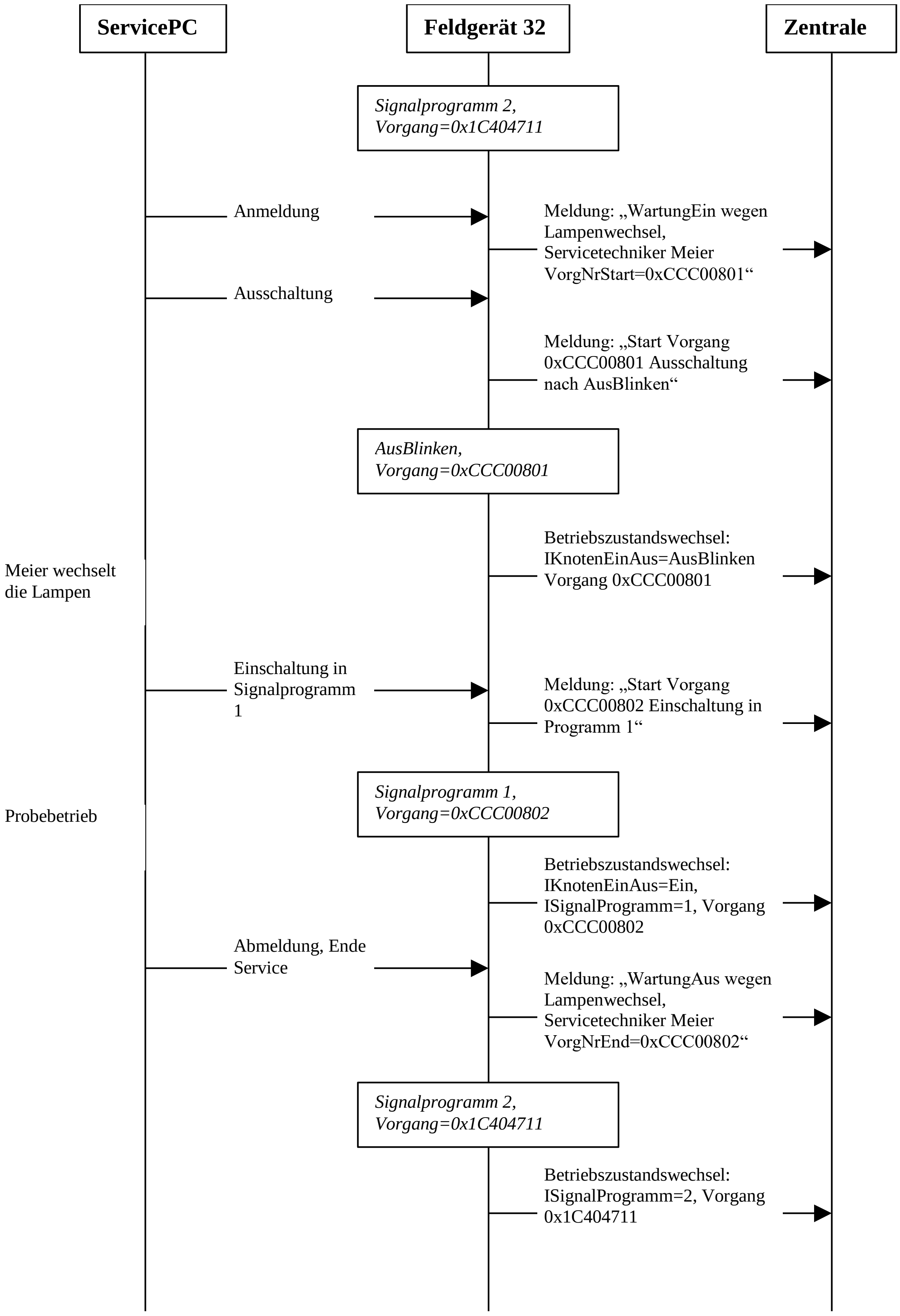

ServicePC
Feldgerät 32
Zentrale
Signalprogramm 2, Vorgang=0x1C404711
Anmeldung
Meldung: „WartungEin wegen Lampenwechsel, Servicetechniker Meier VorgNrStart=0xCCC00801“
Ausschaltung
Meldung: „Start Vorgang 0xCCC00801 Ausschaltung nach AusBlinken“
AusBlinken, Vorgang=0xCCC00801
Betriebszustandswechsel: IKnotenEinAus=AusBlinken Vorgang 0xCCC00801
Meier wechselt die Lampen
Einschaltung in Signalprogramm 1
Meldung: „Start Vorgang 0xCCC00802 Einschaltung in Programm 1“
Signalprogramm 1, Vorgang=0xCCC00802
Probebetrieb
Betriebszustandswechsel: IKnotenEinAus=Ein, ISignalProgramm=1, Vorgang 0xCCC00802
Abmeldung, Ende Service
Meldung: „WartungAus wegen Lampenwechsel, Servicetechniker Meier VorgNrEnd=0xCCC00802“
Signalprogramm 2, Vorgang=0x1C404711
Betriebszustandswechsel: ISignalProgramm=2, Vorgang 0x1C404711

Figure 1 Flowchart for the lamp replacement example

3 System access points

The following system access points are provided in an OCIT outstations system:

- Central system access
  Makes it possible to connect service tools in the control center and also allows access to the field devices. The central LAN is used for the application "Central system access point", and layers 2 and 1 are adjusted accordingly.

- Local system access
  Makes it possible to connect service tools in the field device and also allows access to the control center and other field devices. At this time, there have not been made any definitions for the "Local system access" application.

3.1 Central system access

Note: The scope of functions was expanded as compared to the previous version (preparation for user supply). Note the version of the field device.

System access in the control center consists of one or more interfaces that allow communication with the field devices. They are logically fully identical interfaces with regard to how they lead to the field devices; however, the connection is made via LAN. An OCIT control center must offer at least one system access point:

Interfaces	Mandatory	Project-specific/ Manufacturer-specific
Number	1	> 1
Transmission profile	10 or 100 Mbps Base T Ethernet RJ -45 connector	ISDN or other ser-vices
Protocol	OCIT-O like to the field devices	

However, the manufacturers may provide additional access points and also ISDN connections or other services.

The operator / control center manufacturer provides the following information for each central system access:

- IP address of the system access computer to be connected

- IP address of the gateway computer (if necessary)

- IP address of the name server (DNS)

- OCIT control center numbers to be used by the system access point, OCIT field device numbers.

One field device carries out all commands that arrive via the central system access point.
A valid switching request is accepted as if it arrived from the control center. The "last come - first serve" principle applies. This may be in conflict with the switching requests of the control center. The status change of the field device is visible in the control center via the actual vector and the initiator can be detected via the SysJobld.

In the past, the central system access point was meant for the device suppliers' experts, who with it run their own devices from the control center or from remote locations, e.g. the device supply, or test device functions. As of OCIT-O version 2.0, this scope of application has been expanded, the central system access point is now also used for the user supply with supply tools of any manufacturer. To show clearly how much responsibility is associated with the use of the central system access point for the overall functions of the system, the following rule is set in OCIT-O TSC V2.0:

- Upon delivery of the field devices, the password entered for commands via the central system access point is not the default OCIT-O password, but instead an OCIT-O password that only the manufacturer knows.

- From the central system access point, if this OCIT-O password is not known, the user can only transmit objects via the central system access point that are not protected with the SHA-1 algorithm and that do not affect the system functions. However, data supplies and control commands are protected with SHA-1 and cannot be transmitted.

- If the customer (operator) needs all of the functions of the central system access point, this must be ordered separately. This clearly shows the operator's responsibility for the use of the central system access point. Upon receipt of this separate request, the field device supplier is then either to configure the field devices with the standard OCIT-O password or with one specified by the customer. If this password is known, it will be possible to supply the field devices via the central system access point and to set the switching requests.

4 Object definitions

For data definitions, see OCIT-O-Basis-TYPE_Vy.y.xml.

4.1 System objects

OType numbers of the system objects, Member=0 (OCIT outstations):

OType	Name	Path (from field device)
815	System object field device	./.
817	System object Re-moteDevice	ZNr(USHORT)/FNr(USHORT)

4.1.1 System object field device

Note: The scope of functions was expanded as compared to the previous version (ExtendedInstanceInfo). Note the version of the field device.

The purpose of this object is to provide general information via the field device. In addition, new communication partners can be announce with the field device or their passwords can be changed.

SytemobjektFeldgeraet (0:815)

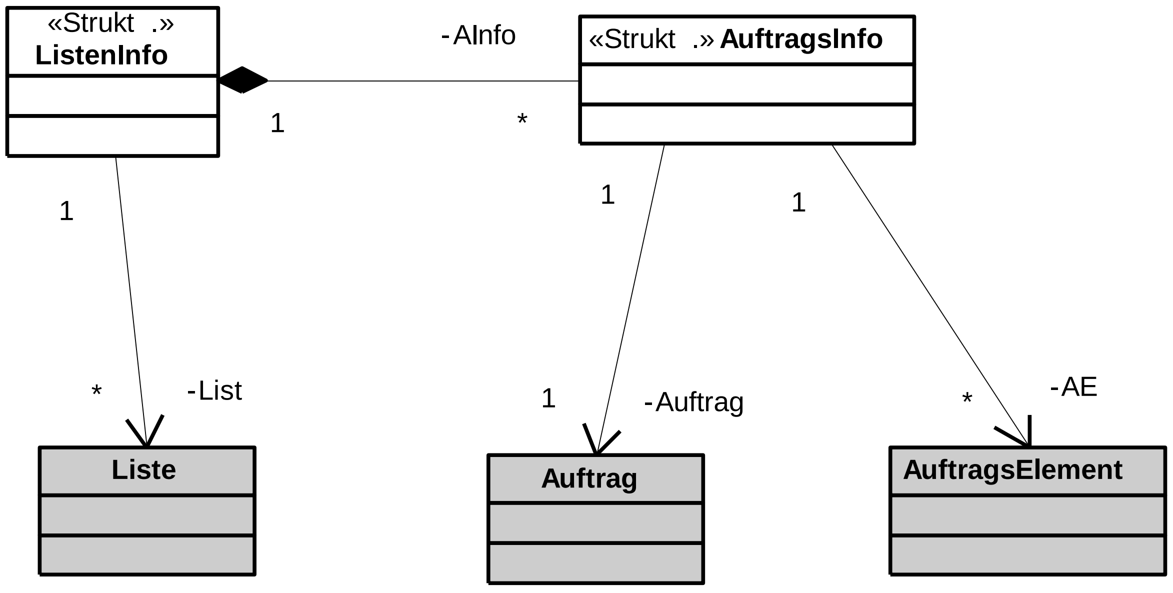
SystemobjektFeldgeraet		
METHOD	Name	Description
100	**GetGeraeteID**	Reads the manufacturer, version and device type. When updated, these parameters may change. The version identifiers supplied by the GetGeraeteID method allows the device's supported scope of functions to be queried.
	Output parameters	
	RetCode	OK: Function was not correctly performed.
	FgType	1: Control center 2: System access 3: Field device
	Member	Refer to OCIT-O protocol for ID of the device manufacturer
	Devicetype	Type of device that is connected (manufacturer specific name).
	Version	OCIT-O version: Historic version identifiers are:

SystemobjektFeldgeraet		
METHOD	**Name**	**Description**
		1 (stands for 1.0 or 1.1), 1.0 and 1.1. As of Version 2.0, use the supported OCIT-O Version (see reference documents about OCIT-O version) in the form x.y. Example: 2.0 or 2.1
	SubVersion	Manufacturer specific version identifiers, e.g. software ver-sions.
	APVersion	Version of the application program software
101	**Cre-ateRemoteEntry**	creates a new remote entry
	Input parameters	
	ZNr, FNr	ZNr, FNr of the external device
	RemoteType	1: Control center 2: System access 3: Field device
	Output parameters	
	RetCode	OK: Entry was created. TOO_MANY: Entry table is full. EXISTS_ALREADY: RemoteDevice already exists.
102	**DropRemoteEntry**	deletes the remote entry
	Input parameters	
	ZNr, FNr	ZNr, FNr of the external device
	Output parameters	
	RetCode	OK: Entry was created. PARAM_INVALID if the entry to be deleted does not exist.
103	**GetTime**	returns the device's current time
	Output parameters	
	RetCode : RetCode	OK: Time was delivered.
	Zeit : ZEITSTEMPEL_UTC	Device's current time
	ZEITZONE : SLONG	Time zone: UTC deviation from local device time in sec-onds; e.g. +3600 seconds for CET, or +7200 for CEDT (Central European daylight time). East of Greenwich, al-ways with a plus sign.
	ZEITQUELLE	Time source (unknown, quartz, control center, DCF, GPS)
104	**InstanceInfo**	Supplies all entity references that match the reference indi-cated in the following. The path may also only be partially specified. The method supplies a max. of 255 references to entities that are of the specified type or a specialization thereof and the path of which begins like the specified

SystemobjektFeldgeraet		
METHOD	**Name**	**Description**
		path. This method may be used, for example, to read out all tasks for a list.
	Input parameters	
	key : BaseObjType?^3	key.RefLen key.Member key.Otype Member, OType specifies the basic data type from which the method delivers entity references. key.path .. Path parameters depending on the type specified above. Parameters can be left out of the end of the path.
	Output parameters	
	RetCode	OK Function was performed correctly (even if class is known but no entity was found). PARAM_INVALID if the class referenced by the key is unknown. TOO_MANY if there are more than 255 entity references. No reference is returned.
	Path[ ] : paths	Entity reference list consisting of: paths.quantity paths[ ].RefLen paths[ ].Member paths[ ].Otype paths[ ]. ... Path parameter depending on the type specified for the runtime.
105	**ExtendedInstance Info**	ExtendedInstanceInfo has the same function as the InstanceInfo, with the only difference that there are 65535 possible return values.
106	**GetListConfig**	Supplies the configuration for one or for several lists, for all or for one selected field device. A max. of 65535 list info instances are returned with the data about the jobs and the job elements.
	Input parameters	
	Anzahl	Number of the following list elements
	ListenNr	Array of list numbers for which configurations are queried. If an array is empty (length zero), the configuration of all lists is returned -
	ZnrFnrFilter.ZNR	Filter, if not equal to null values, only the list info of these ZNR is returned, if null values, list info for all locally defined ZNRs are returned.

SystemobjektFeldgeraet		
METHOD	**Name**	**Description**
		Only important for the control center!
	ZnrFnrFilter.FNR	Filter, if not equal to null values, only the list info of these FNR is returned, if null values, list info for all locally defined FNRs are returned. Only important for the control center!
	Output parameters	
	RetCode	OK. Function was not performed correctly (even if the input parameter listNrs is empty and not changeable list exists). PARAM_INVALID if at least one of the requested lists does not exist or fnr filter is unknown. TOO_MANY: If there are more than 65535 paths and data of lists, tasks and task elements.
	listObjects : Listen-Info[]	List with list info structures consisting of: - List reference (from ZNR) - AInfo : Task info[] consisting of: - Task reference (from list) - Task element including data
107	**GetDetExtChan-nels**	Returns a list of numbers of all the binary inputs that sup-port advanced detector values.
	Output parameters	
	RetCode	OK: Function was not correctly performed.
	channels[ ]	Array of numbers of all the binary inputs that support ad-vanced detector values.

Overview of the parameter structure of GetListConfig:



Note: The version identifiers supplied by the GetGeraeteID method allows the device's supported scope of functions to be queried. Especially the VERSION element makes it possible, for example for the control center to read scope of supported versions for a field device based on its OCIT version status and to operate accordingly.

The OCIT version identifier has the format Version.Subversion, that is

OCIT-Outstations V1.0	"1.0" or "1" (downward compatability)
OCIT-Outstations V1.1	"1.1"
OCIT-Outstations V2.0	"2.0"
OCIT-Outstations V3.0	"3.0"

4.1.2 System Object Control Center

Because from the BTPPL perspective the control center is a field device, the system object FieldDevice also exists in the control center.

4.1.3 System object RemoteDevice

The purpose of this object is to provide general information via the field device. In addition, new communication partners can be announce with the field device and their passwords can be changed.

The Remote Device objects provide individual passwords for access to other IP addresses or to other field devices. The OCIT-O password can be changed with this object. The instance with ZNr and FNr that are the same as those included in the field device uses the default OCIT-O password for unknown IP addresses.

RemoteDevice (0:817)

RemoteDevice		
Path: ZNr, FNr		
METHOD	**Name**	**Description**
0	**Get**	
	Output parameters	
	IpAdresse	IP address that is set for this remote device
	IpName	IP host name of this remote device
	FgTyp	1: Control center 2: System access 3: Field device
100	**SetPassword**	Field devices know at least the following passwords: • Password for the field device itself (pre-programmed with "OCITPASSWORD" upon delivery) • Password for the control center (pre-programmed with "OCITPASSWORD" upon delivery) • Password of the replacement control center • Password of the central system access • Password for unknown IP addresses (default) An IP address, ZNr, FNr and host name of the approved communications partner is assigned to each OCIT-O password, except for the default one. BTPPL calculates the suitable OCIT-O password based on the sender IP address.
	Input parameters	

RemoteDevice		
	NewPassword : UBYTE[20]	Permitted characters: [a...z], [A...Z], [0 to 9] The array is structured in the following manner: NewPassword[0...11]=neues OCIT-O Passwort [0...11] XOR Schleier[0...11] NewPassword[12...19]=Schleier[12...19] If the OCIT-O password has less than 12 characters, it will be filled with binary zeros. The OCIT-O password is made unidentifiable by the XOR operation with a veil. The veil is formed with the aid of the SHA-1 algorithm. For this purpose, the following formula is performed with the device's old OCIT-O password: Veil := SHA-1(old OCIT-O password + ´.´ + ZNRDestinationAddress + ´.´ + FNRDestinationAddress + . Veil + old OCIT-O password + ´.´ + ZNRDestinationAddress + ´.´ + FNRDestinationAddress) The VEIL is a hexadecimal defined array with the following content: 00h: 49 61 65 21 20 49 61 65 21 20 50 68 20 6E 67 6C 10h: 75 69 20 6D 67 6C 77 20 6E 61 66 68 20 43 74 68 20h: 75 6C 68 75 20 52 20 6C 79 65 68 20 77 61 67 6E 30h: 20 6E 61 67 6C 20 66 68 74 61 67 6E Example: If the old OCIT-O password 'OCITPASSWORD' and the field device is linked to the control center 12 with device number 567, the veil is the SHA-1 checksum with the character sequence 00h: 4F 43 49 54 50 41 53 53 57 4F 52 44 2E 31 32 2E ; OCITPASSWORD.12. 10h: 35 36 37 49 61 65 21 20 49 61 65 21 20 50 68 20 ; 567 20h: 6E 67 6C 75 69 20 6D 67 6C 77 20 6E 61 66 68 20 30h: 43 74 68 75 6C 68 75 20 52 20 6C 79 65 68 20 77 40h: 61 67 6E 20 6E 61 67 6C 20 66 68 74 61 67 6E 4F; 50h: 43 49 54 50 41 53 53 57 4F 52 44 2E 31 32 2E 35; OCITPASSWORD.12.5 60h: 36 37 ; 67 This will change the veil depending on the field device, even if the OCIT-O password is the same for all devices. The field device calculates the same veil (with its old OCIT-O password). If this is again placed over the text, the new OCIT-O password becomes significant. It will be used for all further operations.
	Output parameters	
	RetCode	OK: OCIT-O password changed. ACCESS_DENIED; Control center passwords may only be changed from the control center.

4.1.4 RemoteService

Interface objects for Remote (via the IP interface) Service PC. This allows a service PC to prevent switching control center switch requests. Trial switches can be performed as well. Service is always run for the entire field device.

There are several sources for controlling commands, where each source has its own priority:

1. Controlling or servicing "locally"
2. Controlling or servicing "remotely"
3. Control center
4. Local command selection, e.g. automatic time function (lowest priority)

Theoretically, a remote service intervention can be seen as an extended manual device.

As opposed to local service interventions, during remote service interventions, the device is unable to determine the end via the door contact. This is why OCIT provides for a limited-time command to set service mode.

Path: ./. (only one instance per field device)

RemoteService (0:208)

RemoteService		
METHOD	**Name**	**Description**
0	**Get**	
	Output parameters	
	VorgangsNr : SYSJOBID	Identifier for the process that requested the remote servicing, or ZERO if no remote servicing is active.
	EndZeit :ZEITSTEMPEL_UTC	Time indicating how long remote servicing is active. End-Time==0 if no servicing is active.
	ServiceGrund	Indicates why servicing was requested.
16	**StartService**	This allows a service PC to request control over a device. The device may only issue control rights for a single access at any one time. If remote servicing is already active, the method returns ACCESS_DENIED.
	Input parameters	
	EndZeit :ZEITSTEMPEL_UTC	Time indicating how long remote servicing is active. End-Time==0 if no servicing is active.
	VorgangsNr : SYSJOBID	Identifier for the process that requested the remote servicing, or ZERO if no remote servicing is active.
	ServiceGrund	Indicates why servicing was requested.
	Output parameters	

RemoteService		
METHOD	**Name**	**Description**
	RetCode	OK: Entry was created. PARAM_INVALID, INTERVALL_INVALID,ACCESS_DENIED
17	**EndService**	A service PC returns the control rights with the command EndService. After that, the device again considers control center switch requests. In exceptional cases it may be neces-sary to set this command from the control center.
	Output parameters	
	RetCode	OK, (even if remote servicing was not ac-tive)ACCESS_DENIED

4.1.4.1 Direct external access to the field device

Direct external access to the field devices via dial-up connections/networks or the local service interface are the responsibility of the field device manufacturer. They are implemented for each project or manufacturer specifically.

If this kind of direct external access is established to the field device in addition to the connection to the control center, it is critical that the OCIT-compatible devices meet the following:

Start of access: Message MAINTENANCE_ON to the control center.

End of access: Message MAINTENANCE_OFF to the control center.

For direct dial-up connections, it is possible that the control center becomes discon-nected during the time of direct access to the field device, and as a result a connec-tion fault is detected. This fault state can be corrected after access is ended because the field device sends an appropriate message to the control center.

4.2 Messages and measurement values (archives)

In OCIT outstations, messages and measurement values are stored in the archives of the field devices. The methods that the control center uses to query these ar-chives are the same for both messages and measurement values. OCIT outstations combines measurement value and message archives under a common interface.

4.2.1 Properties of the archives

Selected data and messages for the field device are collected in archives. OCIT-O provides the following elements for this purpose (for details, see sect. 4.2.9):

- A general archive interface, which allows archives to be easily managed,
- standard and optional archives for messages,
- measurement value archives that can be defined by the control center during runtime. In order for messages to maintain a defined storage depth, OCIT outstations provides separate archives for messages.

The archive interface for messages and measurement values has the following benefits:

- Measurement values can be added without requiring that extensions are made in OCIT outstations.
- Data is transmitted in compressed form.
- Depending on the manufacturer, there is the option to define additional archives.
- Measurement values can be retrieved or archived several times.
- The number of messages that can be entered into the archive can be increased.
- Depending on the manufacturer, there is an option to expand individual messages (with new message parts).
- It is easy to filter out the original message even if a message was expanded with new message parts.
- Updates to the control center with new messages keeps the control center fully updated.
- The control center can read the entirety of the archive that still exists.
- There is the option to incrementally transmit to control center 1 while at the same time reading the entire content of control center 2 (e.g. system access).
- Data loss does not occur until an overflow in the cache, not because of a transmission fault alone.
- The control center can access the most recent entry (for current status when a connection is being established).

4.2.2 The archive interface

Messages and measurement values are processed in a common interface. The data structures and the defined functions of the interface are structurally identical for messages and measurement values.

Messages and measurement values are stored in "lists". There are several lists in the field device, which store different data. The "tasks" define which data is stored, which are then stored in the corresponding list. This configuration can be read during operation. There are pre-defined lists that are unalterably defined by the device manufacturer and dynamic lists that can be configured by the control center during operation. Up to 256 different tasks are possible for each list.

Each list has its own buffer in which the dynamic data is stored. The buffer is designed as a ring buffer, which is set to overwrite the oldest sets each time. The size of the ring buffer can be configured, however it cannot be changed while measurement values are being registered.

A ring buffer consists of a "second frame". It is possible to create multiple second frames during the same second. However, a second frame is always characterized by the time (to the second) for which the second frame is created. There are only second frames for the times during which data was stored. A second frame only contains so-called "task frames". A task frame stored the dynamic data that were requested by a task. There are different structures of task frames for messages and for measurement values. Both messages as well as measurement values are stored entirely in a secondary frame. Multiple measurement values and messages are possible for each second frame. Normally, messages and measurement values are stored in different lists.

The basic workflow looks as follows: The field device stores new data in its ring buffer but does not automatically transmit the data to the control center. The data is only transmitted to the control center upon request by it. This makes it possible to store the data to the second, however they can also be transmitted in larger time intervals. In addition, the control center can be notified (with events) if the buffer exceeds a fill level that was defined by the control center. In addition, events can be triggered if certain scenarios occur (e.g. if a lamp has burned out). Data is retrieved via BTPPL method calls that are protected not only by PPP and TCP but also via a 16-bit checksum (Fletcher) and can optionally also be protected by a 160-bit checksum (SHA-1).

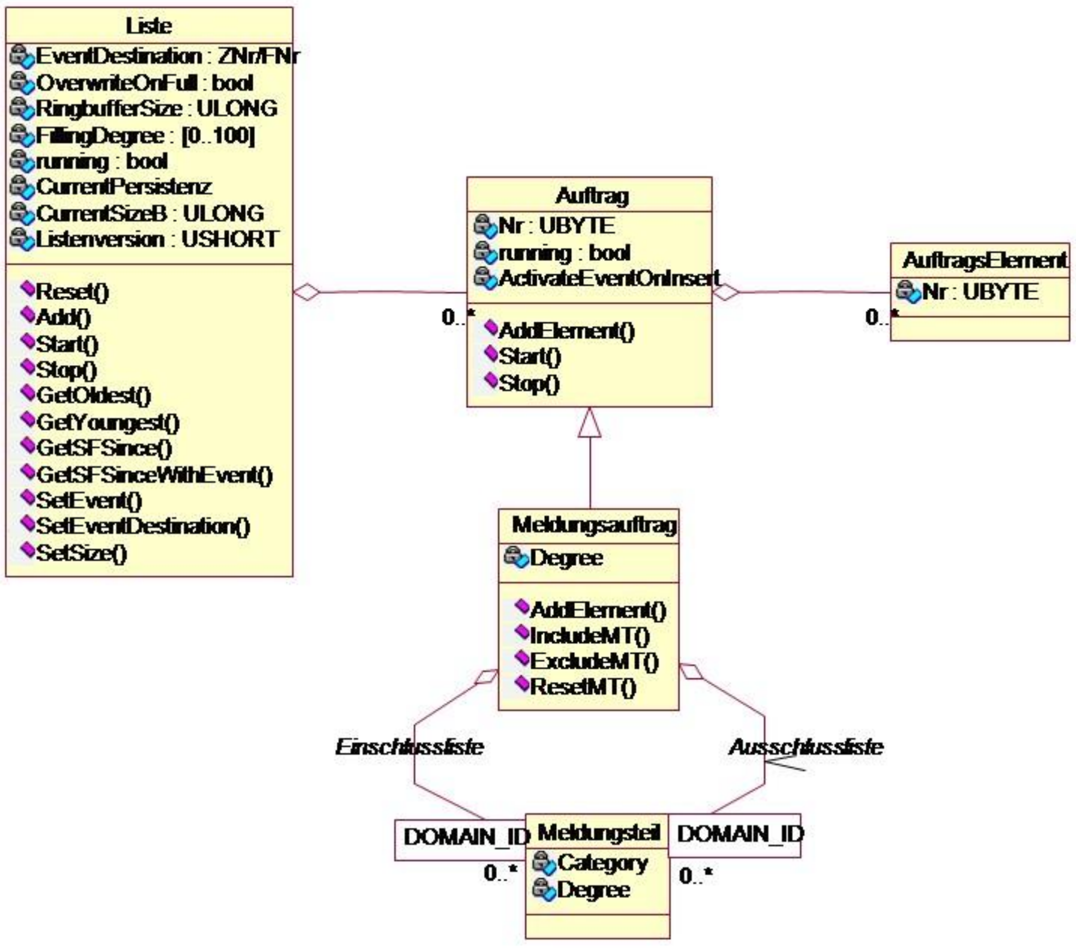


Figure 2: Schema of the archive interface

4.2.3 Elements of the archive interface

The archive interface is broken down into the following abstract elements:

- A **List** of managed tasks and the corresponding dynamic values, such as measurement values or messages. Which values are recorded and saved is defined by the tasks. There are static lists that can be redefined and dynamic lists that can be re-configured by the control center.
- **Tasks** belong to a list and define, which dynamic values should be stored. A task consists of one or more task elements.
- **Task elements** refer to the objects (process variables) and include the information about which values of these objects are recorded. Normally, a type of task element refers exactly to a type of OCIT outstations object, the data of which is recorded. However, it is possible that a process variable of more than one task element is used (in different lists). Each element in OCIT outstations for which a task element exists can be used as a **data source**.
- The **task frame** is the result of a task. Even a task that consists of several task elements will always generate exactly one task frame.

- The dynamic data is stored in **second frames**. A second frame includes, for example, a series of measurement values in the form of task frames that were created in the same second, or all message parts that belong to a message.
- A special task frame is the **message task** used to record messages. OCIT compliant messages consist of a **main message part** and 0 - n **additional message parts** (optional message parts). The main message part determines the message's semantics. This is why a message is broken down into message parts, so that different manufacturers can expand upon an already existing standard message. The grouping illustrates that the message parts belong together semantically. A **message part** consists of an identifier and a set of parameters, which defines the message precisely. The identifier is made up of the manufacturer identifier (member number) and the type identifier (OType).

OType number of the lists in the control center, Member=0:

OType	Name	Path (from field device)
400	List	List(UBYTE)
401	EvList	EvList
402	Task	List(UBYTE)/TaskNumber(UBYTE)
405	Message task	List(UBYTE)/TaskNumber(UBYTE)
430	TaskElement	List(UBYTE)/TaskNumber(UBYTE)/TENr(UBYTE)

All objects—except the event—support the standard function 'Get'. They do not support the function 'Set'. The returned parameters are described in more detail in the XML file.

4.2.4 List

A list has two aspects: Which tasks exists and as such, which dynamic data is transferred, is stored in a static part. There is a **checkword (list version)** for these tasks, which is reset for every time the tasks are changed, thus making it easy to check whether the transmitted dynamic data and the task supply are compatible with each other.

The dynamic part of the list (ring buffer) stores the data that accumulates in "second frames". Then, this data can be retrieved by the control center. There is the option to retrieve data more than once, because they remain stored even after they have been retrieved. As such, even system access points, for example, can use the current measurement values. System access points are not set up to delete jobs or to register themselves at the device as an event destination (device is **not** multi-master capable). The data is stored in a ring buffer, in which the old values are simply overwritten.

In addition, the control center also has the option to have an event sent if a fill-level configured in advance has been reached. In addition, there is an option to specify how much space should be used for a ring buffer of a list.

The lists have fixed addresses (absolute path to the device, irrespective of relative intersections) and transmit data to all relative intersections.

Each second frame has a timestamp (to the second) and a position number. The position number is needed to be able to differentiate between several elements with the same timestamp. The same position number must not appear in the ring buffer of the same list twice, however gaps are allowed (for example, 10 can be followed by 50). There is no need to store the position numbers in the device. Instead, a 32-bit memory address or a file position can also be used. Of course, the ring buffer must be large enough to ensure that it can accept more elements than the maximum number that can be generated per second. (The tuple time/position may only appear once per list).

It can be defined per list, whether or not queries must be secured when submitted. This is necessary, because on the one hand a secure transmission system resource is needed for very large data volumes (online measurement values), yet on the other hand certain lists are relevant to security.

Summary:

- The checkword allows a consistency check between the list configuration and retrieved data.
- The list instances are pre-defined but not started up.
- The lists can be given a fixed address (path absolute to the device, irrespective of the relative intersection).
- The list manages data frames.
- The data is retrieved via the list's interface.
- The list includes a fill level, for which, optionally, the list sends an event to the control center (fill level in %).
- The list does not have any fixed property that defines how much space is reserved for the frames.
- What can be defined is how the system must behave in the event of overfilling (stop or overwrite).
- The maximum size of the ring buffer is a property of the list. The list or the corresponding tasks prevent defining tasks with second frames that are larger than the ring buffer.
- Each list has a ring buffer of second frames that can be queried. A queried second frame is not deleted, but instead remains in the ring buffer until it is overwritten.
- Each second frame has a timestamp (to the second) and a position number (unique for the entire ring buffer).
- The data of one or several tasks is stored in a second frame.
- It is possible that multiple second frames exist for the same second. The data for a taks within the same second is always within the same frame.
- It is possible for the same task number to appear multiple times in a single second frame (especially where messages are concerned).
- It can be defined per list, whether or not queries must be secured when submitted.

- Lists may trigger events in **one** different field device (usually the control center). An event is a method call of the EvList object. An event is triggered if there is a significant change to the status of the list.

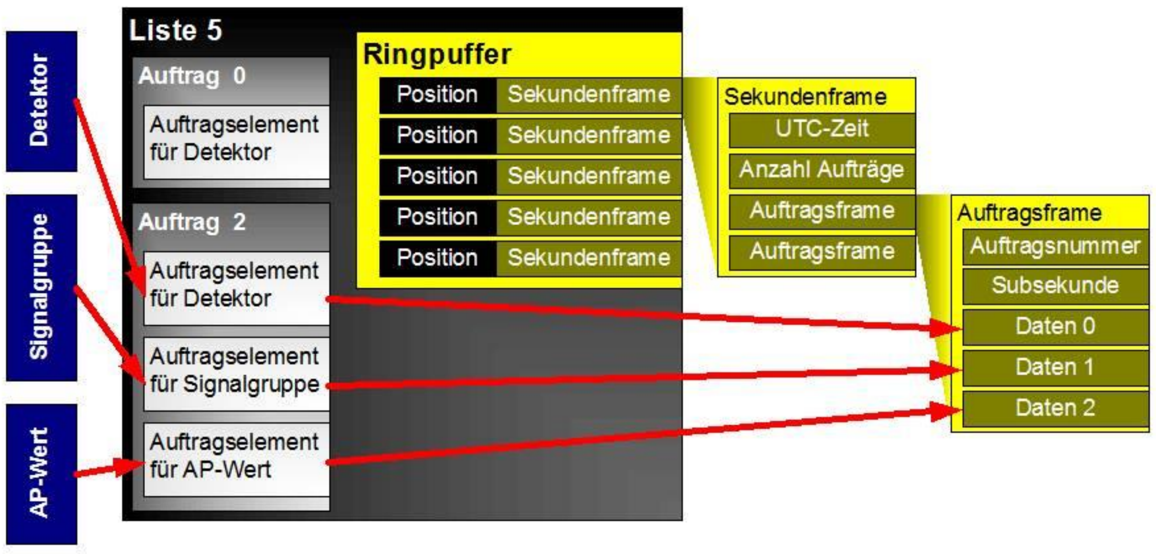


Figure 3: Schema of a list

4.2.4.1 Transmission format

The field device only saves elements chronologically in the ring buffer for each list. It does not save if elements were already read by the control center.

Each entry in the ring buffer (second frame) has its own timestamp (UTC), that is, with a second range. It is possible that several elements with the same timestamp exist within a list. In the event of a time correction, they may also show backward skips. The device notifies of time corrections with an operating message (4.2.12). This allows the control center to re-establish the chronological context of the data.

Each second frame consists of a list of task frames. The structure of a task frame depends on the type (member, OType) of the task frame (caution: this Member/OType combination of the task frame is not the message number and is not transferred). Task frames for messages include variable parameters and among other things the Member/OType combination of the message (that is also transmitted).

Each second frame can be uniquely identified within a list by its timestamp and a position number (within the ring buffer). The position number must be unique within the ring buffer, but it may have gaps. Conceivable as a position number, for example, is a file-offset within a file in the field device or the storage address for the element in the memory. This makes it possible to define the position number as ui4. The position number must only include the value range 0x0 to 0xfffffffe, thus it cannot include 0xffffffff.

If a list of second frames is transmitted from the field device into the control center, you also transmit the timestamp as well and the position number of the previous element and the timestamp and position number of the last second frame that was transmitted.

4.2.4.2 Methods for List

Generally, all methods that are run on the list are authenticated. The only exception is GetSFSince, because GetSFSince does not make any changes to the list itself. GetSFSinceEvent is always transmitted authenticated, however an authentication is not needed in the answer, if AuthenticateAnswer is set to 0.

> Rule: Important data such as operating messages are authenticated; large data volume that is not pertinent, such as measurement values, are not authenticated to benefit of the performance.

A task with the number 0 exists implicitly in each list. With it, the messages "Suspend", "Unsuspend", "StartTask" and "StopTask" are added to the list. These four messages are not inserted in the standard message archive.

The message "time skip" is inserted into all lists with task 0.

List (0:400)

List		
METHOD	**Name**	**Description**
100	**GetOldest**	Read oldest list element
	Output parameters	
	RetCode	OK: Oldest second frame (SF) delivered correctly NO_SF: List does not contain any second frame
	PosNr	Position number of the delivered SF
	Listenversion	Version identifier (checkword) that is changed by the field device each time a task is changed. The version number is also returned to the list during startup, allowing the control center to determine if a change took place. As can be seen based on the data structures directly, the version number for the list is not the version number for the second frame and as such it is not stored in the list. So, if the task is changed for a stopped task, the user of the control center is himself responsible for the side effects.
	Sekundenframe	Oldest second frame (caution: This frame contains a list made up of task frames!)
101	**GetYoungest**	Read youngest list element
	Output parameters	
	RetCode	OK: Youngest second frame delivered correctly O_SF: List has no second frames at all
	PosNr	Position number of the delivered SF

List		
METHOD	**Name**	**Description**
	Listenversion	Version identifier that is changed by the field device each time a task is changed. The version number is also returned to the list during startup, allowing the control center to determine if a change took place. As can be seen based on the data structures directly, the version number for the list is not the version number for the second frame and as such it is not stored in the list. So, if the task is changed for a stopped task, the user of the control center is himself responsible for the side effects.
	Sekundenframe	Youngest second frame (caution: This frame contains a list made up of task frames!)
102	**GetSFSince**	Read second frames from transmitted time/position number in the task of their creation. Only second frames are returned that are entered in the ring buffer after this element (not accounting for time conversion, these are only the younger elements. Where time conversions come into play, the output is definitely based on the task of entry, not based on the timestamp) If there is no second frame to a timestamp/position number, this method starts from the first element entered into the ring buffer that has a UTC timestamp "younger" than the transmitted time (and then continues in the task of the entry). If this does not exist either, the method supplies: RetCode==NO_SF, if pulled up successfully, RetCode== SF_FOLLOW or RetCode==SF_NOFOLLOW is supplied. This method does not make any changes to the current fill level, nor to the fill level that triggered the list.
	Input parameters	
	Zeit : ZEITSTEMPEL_UTC	Time from which elements are read
	PosNr	Position number from which reading begins. The first element to be supplied is the element following Time.PosNo. ZeroValue=0xffffffff The position number 0xffffffff must not occur.
	MaxAnzahl	Maximum number of elements to be read
	Output parameters	
	RetCode	SF_FOLLOW: Second frames are supplied correctly and other second frames, which were entered after these, are in the list SF_NOFOLLOW: Second frames are supplied correctly and no subsequently entered second frames are in the archive NO_SF: List does not contain second frame that fulfills this condition.

List		
METHOD	**Name**	**Description**
	AbZeit : ZEITSTEMPEL_UTC	Timestamp of the second frame that is entered immediately prior to the transmitted second frame in the ring buffer or 0 if no such second from is in the ring buffer.
	AbPosNr	Position number of the second frame that is entered immediately prior to the transmitted second frame in the ring buffer or 0 if no such second frame is in the ring buffer. If the system reads with GetSFSince multiple times one time after the next and no elements were overwritten, the 'FromTime of this call" == Time of the last secondary frame of the last call" and "FromPosNo of this call" == 'PosNo of the last second frame of the last call'. If there is no more older element to the first sent one, FromPosNo is undefined and FromTime === 0.
	BisZeit: ZEITSTEMPEL_UTC	Timestamp of the last element sent in the following, thus from element [quantity-1].
	BisPosNr : ui4	Position number of the last element sent in the following, thus from element [quantity-1].
	Listenversion	Version identifier that is changed by the field device each time a task is changed. The version number is also returned during startup, allowing the control center to determine if a change took place. As can be seen based on the data structures directly, the version number for the list is not the version number for the second frame and as such it is not stored in the list. So, if the task is changed for a stopped task, the user of the control center is himself responsible for the side effects.
	Anzahl	Quantity of following second frames
	Sekundenframes	Read messages (warning: Each message again includes a list made of message parts)
103	**GetSFSince-WithEvent**	GetSFSinceWithEvent is a combination of the methods GetSFSince() and SetEvent(). The parameters LastTime and LastPosNo of SetEvent are the last second frames returned by GetSFSince. This method may only be called up by the device registered as EventDestination (control center). Note: This command is not intended for system access points. The control center must intercept commands of this kind and acknowledge with a negative response (ACCESS_DENIED), if they are routed via the control center. Commands that are sent to the field device directly are detected based on the source IP address and intercepted there.
	Input parameters	
	Zeit: utc	Time from which elements are read

List		
METHOD	**Name**	**Description**
	PosNr	Position number from which reading begins. The first element to be supplied is the element following Time.PosNo.
	MaxAnzahl	Maximum number of elements to be read
	Fill	Fill level in % at which the event is triggered. See SetEvent.
	AuthenticateAnswer	0 (FALSE): The respondent does not need to be authenticated (no SHA-1 checksum needed). If it is still authenticated, the checksum must be correct. 1 (TRUE): The respondent must be authenticated.
	Output parameters	
	RetCode	SF_FOLLOW: Second frames are supplied correctly and other second frames, which were entered after these, are in the list SF_NOFOLLOW: Second frames are supplied correctly and no subsequently entered second frames are in the archive NO_EVENT, if the event cannot be entered for whatever reason NO_SF: List does not contain second frame that fulfills this condition. ACCESS_DENIED Access not allowed because it was not triggered by EventDestination.
	AbZeit	Timestamp of the second frame that is entered immediately prior to the transmitted second frame in the ring buffer or 0 if no such second from is in the ring buffer.
	AbPosNr	Position number of the second frame that is entered immediately prior to the transmitted second frame in the ring buffer or 0 if no such second frame is in the ring buffer. If the system reads with GetSFSince multiple times one time after the next and no elements were overwritten, the 'FromTime of this call" == Time of the last secondary frame of the last call" and "FromPosNo of this call" == 'PosNo of the last second frame of the last call'. If there is no more older element to the first sent one, FromPosNo is undefined and FromTime === 0.
	BisZeit: ZEITSTEMPEL_UTC	Timestamp of the last element sent in the following, thus from element [quantity-1].
	BisPosNr	Position number of the last element sent in the following, thus from element [quantity-1].

List		
METHOD	**Name**	**Description**
	Listenversion	Version identifier that is changed by the field device each time a task is changed. The version number is also returned during startup, allowing the control center to determine if a change took place. As can be seen based on the data structures directly, the version number for the list is not the version number for the second frame and as such it is not stored in the list. So, if the task is changed for a stopped task, the user of the control center is himself responsible for the side effects.
	Anzahl	Quantity of following second frames
	Sekundenframes [ ] : Sekundenframe	Read messages (warning: Each message again includes a list made of message parts)
104	**SetEvent**	Assigns the list if the specified fill level "Fill" is exceeded to call the method EvList::OnFull() in the specified device (typically in the control center). The transmission parameters LastTime and LastPosNo mark the starting position at which to determine the current fill level. Every time the fill level is exceeded after entering a second frame, the OnFull method (4.2.4.3, ArchiveEvent object) is called up. If a value of > 100 is entered for a fill level, no new event is triggered. At a fill level of 0, the event is triggered after each new entry.
	Input parameters	
	LastTime	Time when the last element was retrieved
	LastPosNr	Position number of the last element that was retrieved
	Fill	Maximum fill level at which the EvList::OnFull is triggered. Fill=0 → trigger OnFull after each entry Fill>100 → never trigger OnFull
	Output parameters	
	RetCode	OK if the event can be entered; NO_EVENT, if the event cannot be entered for whatever reason ACCESS_DENIED Access not allowed because it was not triggered by EventDestination.
105	**Start**	Starts the list, i.e. the tasks go live. Even the aggregation of AEAggragat starts. The ring buffer is deleted as soon as the list is started. In some lists, depending on the manufacturer, it may be that it cannot be stopped (and started).
	Input parameters	

List		
METHOD	**Name**	**Description**
		None
	Output parameters	
	RetCode	OK: Data collection started NOT_POSSIBLE: In principle, the list cannot be started. NOT_INACTIVE: Data recording has already started (list was not deleted) ERROR: Command cannot be executed (tasks configuration is incomplete)
	Listenversion	Version number that is also returned for GetSFSinceXXX.
106	**Stop**	Stops data recording of the list. After the list has stopped, it is still possible to retrieve the ring buffer. In some lists, depending on the manufacturer, it may be that it cannot be stopped (and started).
	Input parameters	
		None
	Output parameters	
	RetCode	OK: List is stopped, however is also returned for a list that has already stopped. NOT_POSSIBLE: In principle, the list cannot be stopped. ERROR: Command could not be executed.
107	**Reset**	Stops the list. Removes all of the tasks added with AddTask from the list. The ring buffer is deleted. Sets the EventDestination to the control center. If an EventDestination was entered, an Onlinevalidate is sent to the old EventDestination.
	Input parameters	
		None
	Output parameters	
	RetCode	OK RetCode!= OK -> Command could not be executed.
	ListenversionAlt, ListenversionNeu	ListVersion before Reset ListVersion before Reset

List		
METHOD	**Name**	**Description**
108	**AddAuftrag**	Adds a new task to the list. OCIT outstations only requires that new tasks can be added into a stopped list. It is up to the manufacturer, whether to implement this for already started lists. The new list version is not added until the newly added task has started. For a stopped list, the new task does not start, as long as the list has not started. The task is started once the list starts. Once a list has started, the new task does not start until it is started with a start command for the task.
	Input parameters	
	Member OType	Provides the type of the task to be added. The domain specified must be derived from the task.
	Output parameters	
	RetCode	OK PARAM_INVALID task type specified with Member/OType is unknown, task was not set up. NOT_INACTIVE The task must not be started in task to execute this method BUFFER_TOO_SMALL: Supplied if the second frame is so large that fewer than four entries can fit in the ring buffer NOT_POSSIBLE: Too many tasks (a maximum of 255 tasks is possible)
	AuftragsNr	Task number for the newly added task if RetCode ==OK. The new task can be addressed with the path List()/TaskNo().
	ListenversionAlt, ListenversionNeu	ListVersion before AddTask ListVersion before AddTask
109	**SetEventDestina-tion**	Sets the destination of all events in this list. Once an EventDesitnation is set, this method trigger the Event OnlineValidate for the old event destination. Warning: EventDestination is set first and the event is then called up. The EventDestination is not even withdrawn if Event OnInVali-date returns with a timeout error! Once events are set, they stay set.
	Input parameters	
	ZNr, FNr	CNr/FNr for the device that should receive the traps.
	Output parameters	
	RetCode	RetCode! =OK -> Command could not be executed.

List		
METHOD	**Name**	**Description**
	ListenversionAlt, ListenversionNeu	ListVersion before SetEventDestination ListVersion after SetEventDestination
110	**SetSize**	Sets the size of the ring buffer in bytes. The device determines how large the list can really be and returns the set value. The size must be set while the list is offline and deletes any entries still available in the list. The device attempts to approximately meet the size of the buffer.
	Input parameters	
	Persistenz : UBYTE	Defines which parts of the list should be kept after a power outage: • None NoneThe entire list is reset to default values after a power outage. • Tasks The list tasks remain, the data (content of the ring buffer) is lost. The list is refilled automatically after a power outage. All Both the list tasks as well as the content of the ring buffer is retained through a power outage.
	ListeSizeP	Minimum size of the list desired as a percentage of the remaining available storage. A size is chosen that results in more bytes, but no more than 100%. Other lists are not reduced in size.
	Output parameters	
	RetCode	OK: The command was executed NOT_INACTIVE: The list must not be started.
	CurrentPersistenz	Persistence set for this list.
	CurrentSizeB	Size of the list in bytes as set, or rather size of the current list if no change is possible.
	CurrentSizeP	Size of the list as a percentage of the storage available prior to being called up.
	ListenversionAlt, ListenversionNeu	ListVersion before SetSize ListVersion after SetSize
111	**SetOverwriteOnFull**	Defines the behavior if ring buffer is full.
	Input parameters	
	OverwriteOnFull	If set (true), the list overwrites the old data frames (ring buffer). Otherwise, the list stops. The default setting after reset is true (ring bugger).
	Output parameters	

List		
METHOD	**Name**	**Description**
	RetCode	OK, NOT_POSSIBLE if the device does not support this setting.
112	**Suspend**	Halts list recording without actually stopping it. The ring buffer remains. Once the list is suspended, the operating message Suspend is saved in the list.
	Input parameters	
		None
	Output parameters	
		OK: The command was executed.
113	**Unsuspend**	If the list was halted with Suspend, this "Suspend" is withdrawn and the operating message Unsuspend is saved in the list's ring buffer.

4.2.4.3 Event handlers for lists in the control center

There is **one** OBJTYPE Evlist instance for all lists of a field device in the control center. An event is equal if the destination (control center), method and input parameters match.

The field devices does not send the next event until it has received an acknowledgement to the previous equal event or once it is again activated (with List::SetEvent(), List::GetSFSinceWithEvent(), task::ActivateEvent()). This makes a flow control possible for events accounting for the duration and route of the transmission as well as for processing the event by the control center. The transmission speed of the event adapts to the available bandwidth and the current transmission volume.

The event handlers are called up in the following manner:

EvList (0:401)

EvList		
METHOD	**Name**	**Description**
200	**OnFull**	Is called up by the field device in the EventDestination (typically the control center) if the fill level has been exceeded.
	Input parameters	
	ZNr FNr	Sender field device
	Liste	List number of the list for which the fill level was exceeded.

EvList		
METHOD	**Name**	**Description**
	Output parameters	
	RetCode	Is ignored. Only necessary to receive the send confirmation for the flow control. The user function should always return OK.
201	**OnInvalidate**	Is called up by the field device in the EventDestination (typically the control center) if another event destination is set.
	Input parameters	
	ZNr FNr	Triggering field device
	ListenNr	List of the event that was reset.
	ZNrNeu FNrNeu	The device numbers of the new event destination. This may also be identical to the device number of the querying device.
	Ausgabeparameter	
	RetCode	Is ignored. Only necessary to receive the send confirmation for the flow control. The user function should always return OK.
202	**OnInsert**	Is called up by the field device in the EventDestination (typically the control center) if the dynamic data of a started task was entered.
	Input parameters	
	ZNr FNr	Triggering field device
	ListenNr	List of the event that was reset.
	AuftragNr	Task that triggered the event.
	Output parameters	
	RetCode	Is ignored. Only necessary to receive the send confirmation for the flow control. The user function should always return OK.
203	**OnNetzAus**	Is called up by the field device in the EventDestination (typically the control center) if a power outage is detected.
	Input parameters	
	ZNr, FNr	Triggering field device
	Vorgangskennung:SYSJOBID	Operation identifier of the fault, identical to operation identifier of the corresponding PowerOFF message in the default message archive

EvList		
METHOD	**Name**	**Description**
	Ausgabeparameter	
	RetCode	Is ignored. Only necessary to receive the send confirmation. The user function should always return OK.
204	**OnTransaction-StateChanged**	Is called up by the field device in the EventDestination of the transaction (typically equal to the VD server) if the state of the transaction has not changed, i.e. once the status transition has occurred.
	Input parameters	
	Ref2Device: ZNR_FNR	Triggering field device
	TransactionRef: BaseObjType	Reference to the transaction for which the state was changed.
	OldState: TRANSACTION_STATE	The old state of the transaction.
	NewState: TRANSACTION_STATE	The new state of the transaction.
	Output parameters	
	RetCode	Is ignored. Only necessary to receive the send confirmation for the flow control. The user function should always return OK.

Note: The list::SetEvent() in the traffic signal controller must only be called up by the EventDestination (control center) currently set at the traffic signal controller. If the IP address needs to determined for testing purposes, you will find ZNr, FNr of the EventDestination in the header of the RemoteDevice. A new event destination may be set for any randomly defined RemoteDevice in the traffic signal controller.

4.2.5 Second frame/Task frame

Each second frame consists of one or more list of task frames. A task frame consists of one task number (1 byte) as well as the dynamic data that was generated by the task. For measurement values, this data normally only contains the attribute data, for messages the Member/OType identifier is also sent that represents the "message number" of the message part. Optionally, the manufacturer can specify a format string for each message part in the TYPE file, with which the control center converts the message part into a legible format. If the format ring is missing, the control center outputs the data in an arbitrary format.

For messages, the parameter set that describes the message more precisely follows the Member/OType identifier. The structure and the length of the parameter sets depend on the Member/OType identifier. In order to be able to skip unknown message parts, a 2-byte length of the parameter set is saved for each message part after the Member/OType field.

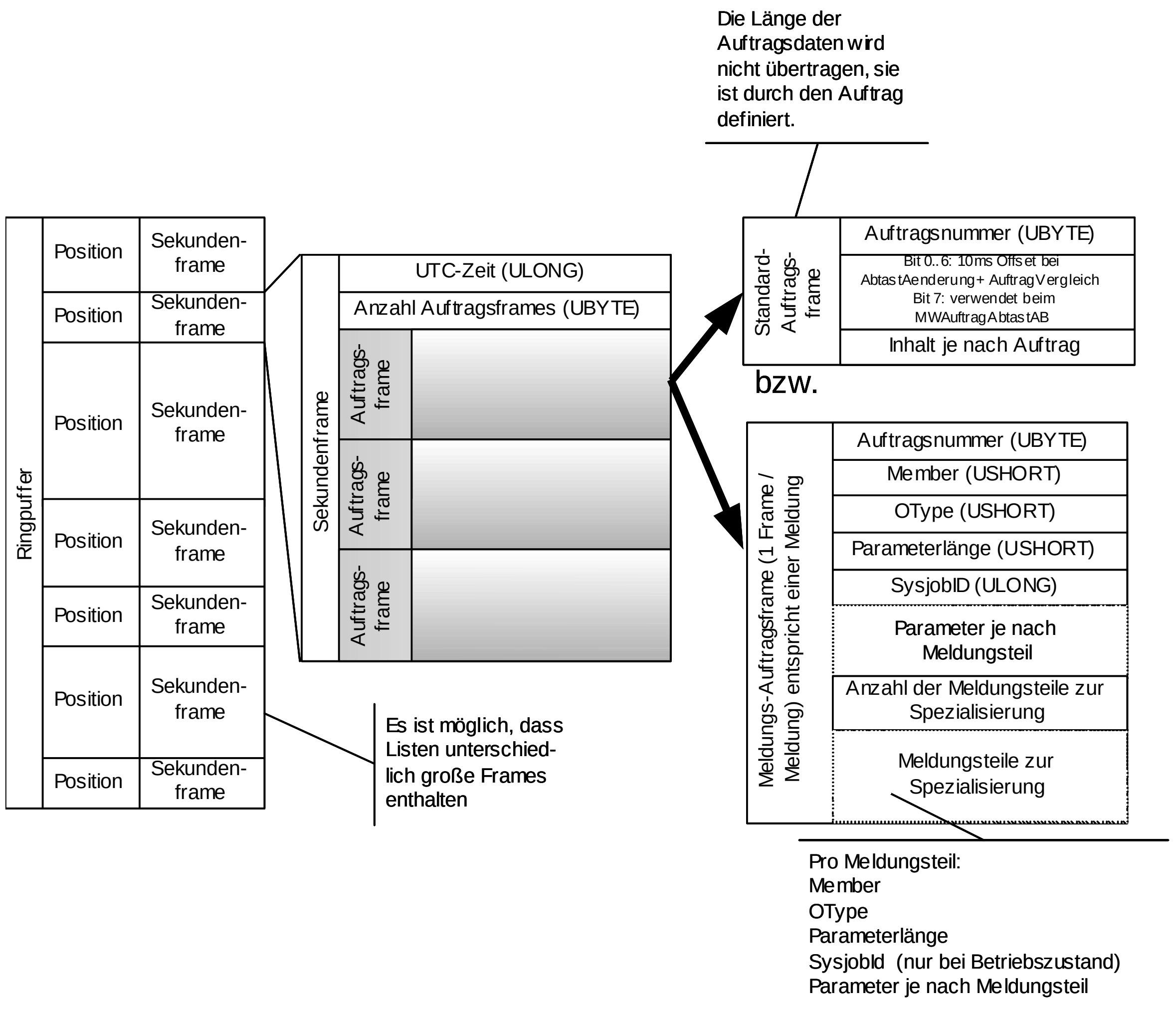

Die Länge der Auftragsdaten wird nicht übertragen, sie ist durch den Auftrag definiert.
Ringpuffer
Position
Sekunden-frame
Sekundenframe
UTC-Zeit (ULONG)
Anzahl Auftragsframes (UBYTE)
Auftrags-frame
Standard-Auftrags-frame
Auftragsnummer (UBYTE)
Bit 0..6: 10ms Offset bei AbtastAenderung+ AuftragVergleich
Bit 7: verwendet beim MWAuftragAbtastAB
Inhalt je nach Auftrag
bzw.
Meldungs-Auftragsframe (1 Frame / Meldung) entspricht einer Meldung
Member (USHORT)
OType (USHORT)
Parameterlänge (USHORT)
SysjobID (ULONG)
Parameter je nach Meldungsteil
Anzahl der Meldungsteile zur Spezialisierung
Meldungsteile zur Spezialisierung
Es ist möglich, dass Listen unterschiedlich große Frames enthalten
Pro Meldungsteil:
Member
OType
Parameterlänge
SysjobId (nur bei Betriebszustand)
Parameter je nach Meldungsteil

Figure 4: Schema: Seconds and task Frame

4.2.6 Task

A series of tasks belongs to each list, which defines which dynamic data will be saved in the list. Each task belongs to exactly one list; no tasks are planned for multiple lists.

A task is uniquely identified within a list with a UBYTE (task number). Thus, the task can be reached via the path Device/List()/task(). For dynamic lists, a task does not exist from the start, but instead it must be created via the method "ListAddJob". For static lists (e.g. messages), tasks already exist.

As opposed to the list, the task is a "virtual basic class", i.e. there is a series of special orders that can be used in the lists, however not the basic task itself. Which task type should be used is specified in the "List.AddJob". It is possible that not every list accepts every task type.

Some task types may be compiled of task elements, in which details are then saved defining which data should be transmitted. Other task types already include data implicitly that are transmitted there, e.g. the task type for messages.

- There are different types of tasks.
- All tasks are ObjTypes under the relevant list (path: List()/Task())
- A task aggregates several task elements.

Each task implicitly includes a condition. Once the condition is met, the dynamic data of the task is recorded and is stored in the ring buffer's second frame.

Each task may be started and stopped if necessary. Starting and stopping a task only means that the values for this task were not recorded and were not entered in the ring buffer. The list remains in operation with the other tasks. If necessary, the task can be changed in stopped state. In order to be able to meaningfully analyze the list, a task's start and stop as well as the halting of the entire list and time skips are entered in the list as messages. Once the task has been changed and is then restarted, the version number for the list is changed accordingly and the changed list number are also then returned to the GetSFSinceXXX answers.

It is possible that tasks exist that cannot be started or stopped. This applies in particular to default message lists and their tasks. It is also possible that no new tasks can be added to certain lists. Not all newly created tasks are started. A task cannot be started separately if the list has not started. When the list starts, all tasks are started (irrespective of whether they had started in the meantime or not). If the list is running, the task is not stopped and started separately.

When starting a list and when starting a task while a list is running, a (new) list version is always returned.

To ensure that no terminals can change the tasks of lists, the following applies: The field device rejects all change orders that do not original from the EventDestination.

4.2.6.1 Methods for all tasks

Task (0:402)

Task		
METHOD	Name	Description
119	**ActivateEvent**	Activates or deactivates the event, which is triggered when an element of this task finds its way into the list.
	Input parameters	
	ActivateEventOnInsert	0: The event is deactivated 1: The event is activated
	Ausgabeparameter	
	RetCode	OK: The event has been successfully activated or deactivated.
	ListenversionAlt, ListenversionNeu	ListVersion before ActivateEvent ListVersion after ActivateEvent
120	**AddElement**	General function for all tasks. Is called up when a new element needs to be added to the task. AddElement can only be called up if the task has not yet been started. This way it is possible to query task elements for "old" data as well.
	Input parameters	
	Member OType	Task element to be added
	Output parameters	
	RetCode	OK : is returned if the task element was able to be added NOT_POSSIBLE : is called up if the task type does not allow any task elements, such as for messages, R09 and AMLi NOT_INACTIVE: is returned if theoretically task elements could be added, but the task has started. PARAM_INVALID: is returned if the task element type is unknown.
	AENr	Number of the task element that is being added. For != OK, the value is undefined.
	ListenversionAlt, ListenversionNeu	ListVersion before AddElement ListVersion after AddElement
121	**Start**	Starts the task if the list has started (and it is possible to start the task separately). If the list has not started, an error is returned. Start triggers a message entry which may be entered in the list itself.
	Input parameters	

Task		
METHOD	**Name**	**Description**
		None
	Output parameters	
	RetCode	OK: Is returned once the task was successfully acti-vated. NOT_POSSIBLE: Is output if the task type (or the list) does not allow the task to be started or the list is not active.
	Listenversion : USHORT	New version number of the list that belongs to the task. If RetCode !=OK, the value is not defined.
122	**Stop**	Stops the task if the list has started (and it is possible to stop the task separately). If the list has not started, an error is returned. Stop triggers a message entry which may be entered in the list itself.
	Input parameters	
		None
	Output parameters	
	RetCode	OK: Is returned once the task was successfully start-ed. NOT_POSSIBLE: Is output if the task type (or the list) does not allow the task to be stopped.

4.2.6.2 Message task

There is one message task per message degree (information, warning, error, critical error) for each message list. A message list does not need to have all message de-grees (and thus message task). The degree of the main message part defines which "messages" are processed by which task.

For message tasks that by default are an integral part of archives, it is defined for each archive specifically how messages are arranged in the Include and/or the Ex-clude List.

The following statements apply to message tasks set up by a control center:

- After being set up by the control center, a message task is given Degree I
- All messages are in the Exclude List, i.e. by default the task does not generate any message frames. After ResetMT, all message that correspond to the degree of the task are added in the Include List.

The elements may be queried via the get function for the relevant task element. A detailed lock and unlock is not necessary, because a list is only managed by one participant.

Message task (0:405)

Message task		
METHOD	**Name**	**Description**
120, 121, 122	**AddElement, Start, Stop**	See 4.2.6
130	**IncludeMT**	Is called up if a main message part or a message part of the OsActualVector should be added to this message degree (and as such to the message task). If the entry already exists in the Exclude List, it is removed from the Exclude List.
	Input parameters	
	Member OType	Main message part
	Output parameters	
	RetCode	OK: Is returned once the message part was successfully entered. NOT_POSSIBLE: Is output if the message parts cannot be added onto or removed from this list. PARAM_INVALID: The specified message part is not a main message part known by the device (or an entirely different object). NOT_INACTIVE: Even though changes can be made while the task is running without any problems, depending on the manufacturer (and possibly depending on the project), it is possible that no changes are made in the Include and Exclude List once a list has started. In this case, NOT_ACTIVE is returned.
	ListenversionAlt, ListenversionNeu	ListVersion before IncludeMT ListVersion after IncludeMT
131	**ExcludeMT**	Is called up if a main message part or a message part of the OsActualVector should be removed from this message degree (and as such from the message task). As such, it is possible that a main message part is not transferred. If the entry already exists in the Include List, it is removed from the Include List.
	Input parameters	
	Member OType	Main message part
	Output parameters	

Message task		
METHOD	**Name**	**Description**
	RetCode	OK: Is returned once the message part was successfully entered. NOT_POSSIBLE: Is output if the message parts cannot be added onto or removed from this list. PARAM_INVALID: The specified message part is not a main message part known by the device (or an entirely different object). NOT_INACTIVE: Even though changes can be made while the task is running without any problems, depending on the manufacturer (and possibly depending on the project), it is possible that no changes are made in the Include and Exclude List once a list has started. In this case, NOT_ACTIVE is returned.
	ListenversionAlt, ListenversionNeu	ListVersion before ExcludeMT ListVersion after ExcludeMT
132	**ResetMT**	The Include and Exclude Lists are reset to the device's internal default values.
	Input parameters	
		None
	Output parameters	
	RetCode	OK: Is returned once the message part was successfully entered. NOT_POSSIBLE: Is output if the message parts cannot be added onto or removed from this list.
	ListenversionAlt, ListenversionNeu	ListVersion before ResetMT ListVersion after ResetMT
133	**GetInEx**	The Include and Exclude list is returned. All of the messages generated by the device can either be found in the Include List or in the Exclude List for the task.
	Input parameters	
		None
	Output parameters	
	IncludeAnzahl: USHORT IncludeMT[ ]: {Member, OType}	List of the current Include message parts.
	ExcludeAnzahl: USHORT ExcludeMT[ ]: {Member, OType}	List of the current Exclude message parts.
	RetCode	OK
134	**SetDegree**	Apply a new message degree to the message task. The Include and Exclude list is not changed.

Message task		
METHOD	**Name**	**Description**
	Eingabeparameter	
	Meldungsdegree	New message degree
	Ausgabeparameter	
	RetCode	OK: The message degree is assigned successfully
	ListenversionAlt, ListenversionNeu	ListVersion before SetDegree ListVersion after SetDegree

4.2.7 Message

There is no message per se in the common message and measurement value model. Instead, a MSGJobFrame is entered in a seconds frame for each message. Breaking down messages into message parts makes it possible to expand default messages.

If, for example, the default message "red lamp error" is to be expanded by XY, a XY message part unique to the manufacturer is defined, which is transmitted every time there is a "red lamp error" message. This provides an easy way to use filters to listen for "red lamp errors" and to nonetheless perform expansions specifically for a manufacturer.

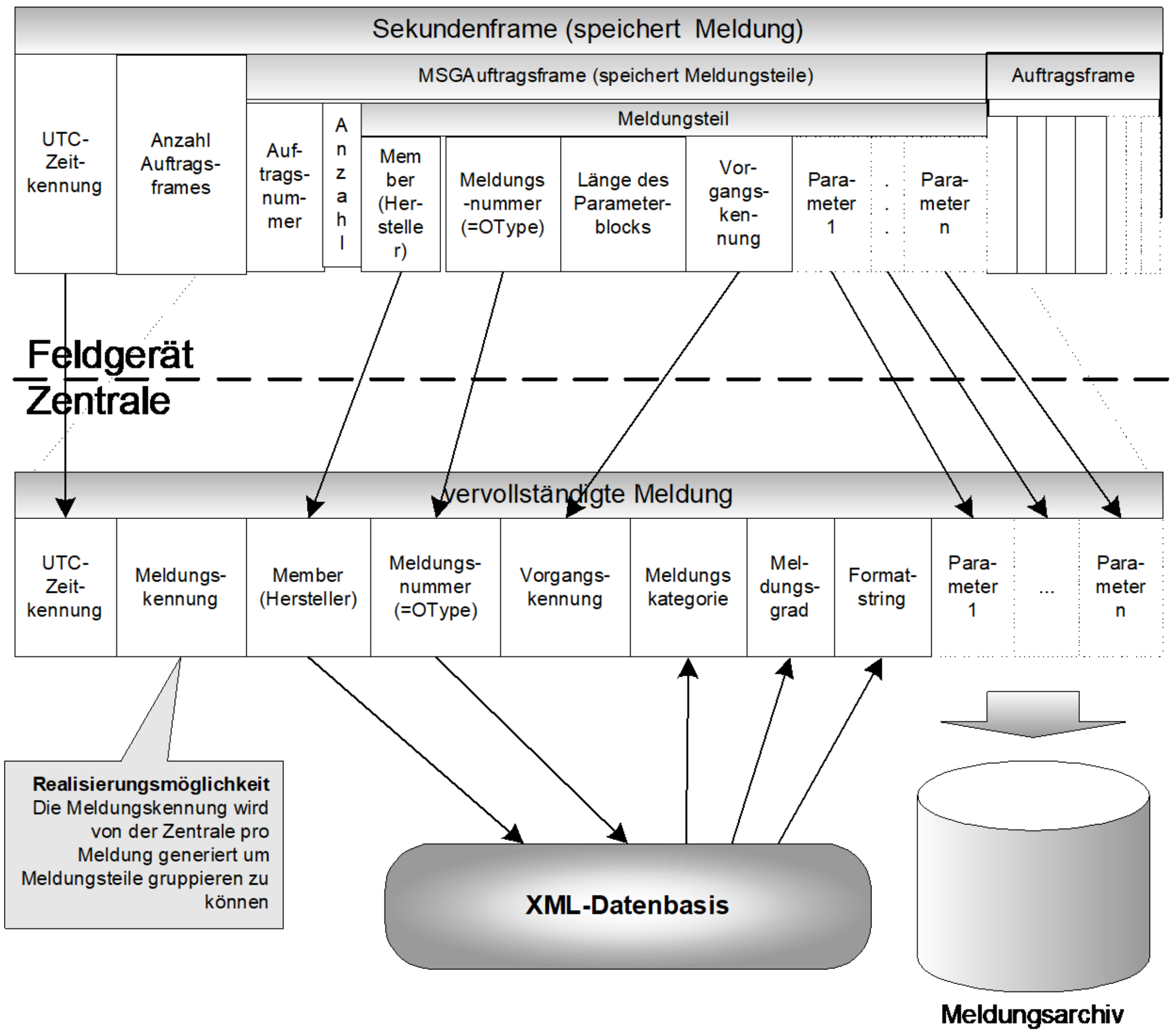


Figure 5: Schema on handling messages

4.2.8 Message part

Each message part is stored in a special task element. A task number is stored per message part, which is not relevant to the message itself. The number of messages parts and the Member/OType identifier that specifies the message number follow the task number. The following length of the block of parameters is meaningful if a message part is unknown in the XML file for some reason. In this case, all of the following message parts and also all of the following second frames/messages can no longer be evaluated. The minimum value of the length is 4 (length of the process identifier; see sect. 2.4).

There is no need to give a process identifier to each message part; in cases such as these, a 0 is transferred. The advantage of a field of this kind is that the control center is by all means capable of finding entries belonging to a certain process even for a wide variety of message types.

The parameters are different for each message and are defined in the XML file so that they can be evaluated in the control center.

A 'message' always consists of exactly one main message part and 0 to n secondary message parts.

Message parts that are only derived from the MESSAGE part (and not the main message part) can only be used as secondary message part. Message parts derived from the main message part can be both a main as well as secondary message part.

Exception to this rule: OsActualVector - it is entered as a main message in the operating state archive.

4.2.8.1 Category and severity of a message part

OCIT-Outstations defines a series of categories and severities for messages. The category and the severity of the message can only be defined at the main message part.

These categories and severities must be defined at two points: The default category and the default severity of the message is saved in the XML file.

The following categories and severities are defined:

MessageDegree	**Description**
Information (0)	Has no impact on traffic.
Warning (1)	Has no impact on traffic, however it should be processed.
Error (2)	Has no significant impact on traffic.
Critical error (3)	Has significant impact on traffic.

Message category	**Description**
Other (0)	None of the following categories.
Devices hardware (1)	Devices hardware in general.
Target pattern error (2)	Signal monitoring: Controller attempted to set a defective pattern.
Actual pattern error (3)	Signal monitoring: Lamp failure in the traffic lights / keys, a pattern does not appear or appears where not intended.
User program (4)	Traffic-related messages in user programs.
Transmission system (5)	Communication to control center (it can be expected that critical errors in this category are not sent to the top immediately).
Operating system, (6)	System error and miscellaneous.
Firmware (7)	Non-traffic-related user programs.
Power supply (8)	Messages relating to power supply.
Clock (9)	Messages relating to time errors.
Detectors (10)	Messages relating to detectors.

Message category	**Description**
Operation status (11)	Operation status messages.

4.2.8.2 Definition of message parts

Messages are defined as a list of MESSAGEPARTS, which are expanded STRUCTDOMAINs. The message part is coded as IdData. A message part is uniquely identified with the MEMBER and OTYPE values defined therein. The parameters of the message part are illustrated as components (DECL entries).

The only limitation relates to the size (and quantity) of elements. It is defined that the number of atomic elements (numbers or strings) must not exceed 32. What this means is that a parameter set, for example, can be made up of 32 simple domains or, for example, of an array of 4 structured domains, with 8 simple domains each, etc. As such, references are interpreted as structured domains, for which the transmitted path elements are the elements of the structure.

The added information about the message type is defined by the manufacturer in the relevant fields:

Formatstring: For each MESSAGEpart, a format string may be defined that briefly characterizes the message. It is the responsibility of the control center to analyze this format string (see sect. 4.2.8.4).

MessageCategory: The error locations or sender is restricted by the category.

MessageDegree: Degree of restriction of the traffic regulation function of the intersection controller

4.2.8.3 Structure of a message part

All message parts must be derived from the MSGPART MESSAGEPART. The MSGPART is only a special STRUCTDOMAIN, for which three class attributes are pre-defined: CATEGORY, DEGREE and FORMAT. CATEGORY contains the message category as a number, DEGREE contains the MessageDegree as a number and FORMAT contains the format string.

The question now is why a message part is derived at all, since a message is already broken down into message parts. The part derived is also not supposed to be used to derive semantically similar message parts from each other, but instead is meant especially for those who do not like typing from having to write out the parameters entirely every single time semantically different messages were declared.

If a message is supposed to be expanded, then not the message part should be derived, but instead a new message part should be created with new parameters. This applies in particular to OCIT-Outstations messages. It is not permitted to specialize OCIT-Outstations message parts by way of derivation, because then the OCIT-Outstations message part will no longer be visible in the control center!

Semantically, the derivation means: The basis message is not transmitted to the control center! A derivation does not include a semantic derivation. So, if a basic message X exists and a message Y is derived from X, then transmitting Y does not mean that X is also reported. Instead, new message parts must be defined

OCIT-Outstations defines a series of message part that are listed further below.

4.2.8.4 Format string for message parts

For details see the document "OCIT-O rules and protocols".

With OCIT, it is possible to expand the standard by manufacturer-specific objects and methods. To also make these expansions available to different manufacturers when using systems from a variety of manufacturers, these objects must be described entirely as an xml file (<manufacturer>AddOns.xml). When doing so, the nomenclature defined in OCIT-Standard must be used. This is especially relevant for secondary messages. In order to have them automatically parsed from the control center to be displayed and to be able to process them, making it possible to show these messages on the screen in clear text, the format must be maintained exactly. Only a short characterized text is defined for the message.

The message texts must be structured (format) as shown below:

Example text no. @Parameter 1@ arrived from intersection @parameter 2@.

The possible values and meanings of the parameters included in the message must be described in an XML format.

4.2.9 Which archives exist?

The following entries are fixed in OCIT-Outstations:

In each field device:

1 The standard message archive for general fault messages such as errors

2 The Syslog archive for project specific information (general archive)

In order for messages and errors to maintain a defined storage depth, OCIT outstations provides separate archives for this.

The operating status archive is defined in the OCIT-O TSC with the number 0 and must not be used for any other applications.

For additional archives, see OCIT-O TSC. Provided there are, among other things, measurement values archives definable by the control center on the runtime. They have the same structure as the archives above and consist both of a list of measurement values tasks as well as one ring buffer each to absorb dynamic values.

4.2.10 Behavior in the event of a power outage

The ListAttribute CurrentPersistence gives details about the behavior in the event of a power outage for all lists. The standard report archive remains intact (list task structure and ring buffer content CurrentPersistence=All=2).

4.2.11 Transmission format of archive data (format of the message)

Each message part consists of one Member/OType identifier that characterizes the message. Following the Member/OType identifier is the length of the parameter block and the parameters belonging to the message.

4.2.12 Element descriptions for message archive

Note: The scope of functions was expanded as compared to the previous version (definitions for "Fault cleared" and "Power OFF"). Note the version of the field device.

The general archive is transmitted per message as a list of message parts; in case of many messages only the main message part arises. Complete implementation of the error messages defined here is not required since some error types do not occur for some TSSs. It is only required that those errors that occur are coded compatibly with OCIT-Outstations. Additionally, manufacturer-specific or project-specific message parts or messages are also possible.

Recommendation for message management in a control center:
Message forwarding for the purpose of troubleshooting / maintenance should as a rule not be done based on error messages, but instead after checking the current status of the device.

Main message parts OCIT-Outstations (Member = 0):

(MessageDegree I: Information, W: Warning, F: Fault, S: Serious fault)

OType	Short name	MessageDegree	Description
60000	Fault cleared	I	This message appears if a fault was present and was cleared. This main message part is only used for messages for which no special clearance message is defined. Often-times, the main message is also more precisely specified with additional secondary message parts. In the process, faults that were cleared can be listed as secondary mes-sages.
60001	Power OFF	S	Shows that time when the power supply was switched off. The message can be expanded with a manufacturer-specific secondary message, which may also differentiate the reason for the power outage.
60002	Power ON	I	Indicates "Power restored".
60003	System error	S	System error that has a major impact on the device's func-tions.
60012	Communications fault	W	Is entered if communication to the control center is inter-rupted.
60013	Communication ok	I	Is entered as soon as communication is again active.
60016	Clock faulty	W	Is entered if the clock is faulty. In this case, the device is required to procure the time from the control center so as to maintain synchronicity.
60017	Clock ok	I	The clock is ok again
60018	Maintenance ON	I	The control center is informed that the device is being ser-viced.
60019	Maintenance OFF	I	The control center is informed that maintenance has ended.
60020	Door open	W	The door closing contact reports: The device's door is open. (The message is only used if the door closing contact exists in the specific project)
60021	Door closed	I	The door closing contact reports: The device's door is closed. (The message is only used if the door closing contact exists in the specific project)
60026	Time skip	F	Is reported by the device if the time is corrected dramatical-ly. The timestamp for the message has a new time. Parameter: Time difference = Tnew – Told in seconds (SLONG)

OType	Short name	MessageDegree	Description
			Time source {quartz, control center, DCF, GPS}
60028	Suspend	I	Reports when a list is halted per Suspend. Parameter: List number.
60029	Unsuspend	I	Reports when a list is halted with Suspend and is then re-started with Unsuspend Parameter: List number.
60030	StartJob	I	Reports that a task was started Parameter: List number, task number
60031	StopJob	I	Reports that a task was stopped Parameter: List number, task number
60032	ResetList	I	Reports that a list was RESET. Parameter: List number Note: This message is only entered in the standard message archive (1).
60033	SyslogI	I	System message information. Parameter is a string.
60034	SyslogW	W	System warning.
60035	SyslogF	F	System error
60036	SyslogSF	S	Critical system error.
60039	DoorOpenDevice-Part	W	Optional [1]: Device part was opened. Message is logged as an additional message part for the message Door Open.
60040	DoorClosedDe-vicePart	I	Optional: Device part was closed. Message is logged as an additional message part for the message Door Closed.
60041	DoorOpenEVU-Part	W	Optional: EVU part was opened. Message is logged as an additional message part for the message Door Open.
60042	DoorClosedEVU-Part	I	Optional: EVU part was closed. Message is logged as an additional message part for the message Door Closed.
60043	DoorOpenCon-trolElement	W	Optional: Control element was opened. Message is logged as an additional message part for the message Door Open.
60044	DoorClosedCon-trolElement	I	Optional: Control element was closed. Message is logged as an additional message part for the message Door Closed.
60101 -	Special message	I	Project specific special message number 1 to 8

[1] OType 60039 to 60044: The option for the expanded "Door open messages" may require adjustments to the hardware of the field devices.

OType	Short name	MessageDegree	Description
60108	1 to 8		
60109 - 60116	Special message 9 to 16	W	Project specific special message number 9 to 16
60117 - 60124	Special message 17 to 24	F	Project specific special message number 17 to 24
60125 - 60132	Special message 25 to 32	S	Project specific special message number 25 to 32

The following parameters exist for messages Reset, Suspend and Unsuspend:

List number (UBYTE)	Number of the list that was suspended

The following parameters exist for messages Start Task and StopJob:

List number (UBYTE)	Number of the list for the task being processed.
JobNumber (UBYTE)	Number of the task that was started or stopped.

5 Procedures Message and Measurement Values

5.1 List with predefined tasks

The following lists have predefined tasks that can only be modified:

- The standard message archive for general fault messages such as errors (fault archive)
- The Syslog archive

If the lists are reset, the predefined, original tasks are re-established.

5.1.1 Objective

There is a "started" list from which the dynamic data can be retrieved.

5.1.2 Process

- **Reset the list (400:107):** The list's reset function has two meanings: For lists that have not started, it creates lists internally and for lists that have already started, it end the current run and resets the tasks and parameters to the default values. The result is in any event a freshly created list (ring buffer was deleted).

- **Defining the list size (400:110):** The manufacturer creates each list with a default size. The size of the buffer may be changed with the aid of the command. It is possible that there is no way to change the size of some lists. In this case, the current size is reported back.

- **Defining tasks**: Jobs are created in two steps. First, the type of task is created in the list. To do so, the type is transmitted and the task number is returned as a functions result. Then, the task object created in this manner is configured with its own methods. Please be aware that the different task elements may in some cases have different configuration functions.

- **Creating a task (400:108): The task is created in the list.** The function contains the number under which the task can be triggered and under which the dynamic data is stored in the ring buffer. There is an option to define for each task (4xx:119) whether an event should be triggered in addition to the entry of the task frame in the ring buffer.

- **Configuring the task**: Task configuration depends on the type of task. Measurement values typically consist of several task elements. These task elements must also be created.

- **Creating a task element (4xx:120)**: Task elements are created for all task types identically via function 120. The function returns the number of the task element that was added and then allows the task element to be configured.

- **Defining the event destination (400:109)**: For every list that is meant to trigger events an event destination is assigned. Normally, this is the control center, however a different destination may also be entered. All of the events for this list are then sent to this destination. If no destination is entered, the control center is used by default. Defining this triggers an event on the control center in any event, so that the control center knows which lists are in use.

- **Start of the list**: The list is started with the command (400:105).

5.2 Changing lists

The objective when making changes to lists is so that the old data can remain in the ring buffer without changing the ring buffer, allowing them to still be interpreted correctly. One must assume that a client only reads a partial area of data from the list. Based on the above, the following design decision was derived: **A task cannot be deleted without stopping the list and restarting it (whereby the buffer is deleted completely). It is possible, however, that a task is started so that the data volume is reduced.**

If a list should be changed, the design merely provides for completely deleting the list and then restructuring it. The critical UseCase to redefine the list. Instead of reading the list and redefining it, the list can be entirely redefined from the start, which will save the number of required call-ups.

The current configuration of the list can be determined using Get calls for the task elements. The Get calls return the configuration.

5.3 Changing the degree (of significance) of individual messages

For message archives, one task is used per degree of significance (the so-called MessageDegree). By default, all messages have one MessageDegree. The MessageDegree of a message is equal to the message degree of the main message part. All other message parts for the message are irrelevant for defining the MessageDegree.

Now it may be necessary for the specific project to change the MessageDegree. For this purpose, the functions IncludeMT and ExcludeMT exist for message tasks. IncludeMT is used to assign a message to a task, with ExcludeMT, the message is removed from a task. A message part that exists in more than on task is written in the ring buffer twice; a task that does not exist in any list is disregarded.

So, in order to change a degree, two commands are needed:

- With IncludeMT, the message is assigned to the new task (and thus the new degree; more precisely: the identifier of the main message part is assigned to the message).

- With ExcludeMT, the message is withdrawn from the task (and with it, the old degree).

What needs to be done is to first re-assign the message and to then delete it so that a message that occurs during the process is not lost.

If an element is removed from a task with ExcludeMT, it can be included back into the task with IncludeMT. IncludeMT automatically deletes the message from the Exclude list.

5.4 Retrieving data

Messages and measurement values are both retrieved in the same manner. Under no circumstances is data transmitted from the field device into the control center without being prompted. The events only notify a need for it to be retrieved. There are two different requirements for retrieving data:

- Retrieving data continuously. Example: The control center retrieves operation messages from the device in any event. It does not matter whether the data is retrieved singularly or in blocks; they are continuously retrieved from the device in any event (normal scenario for the communication control device - field device).

- Parts of the ring buffer are retrieved spontaneously if, for example, the maintenance terminal accesses the messages or the measurement values in the ring buffer (normally maintenance terminal - field device).

For an understanding of the sequence it is important to know: All elements that are stored in a ring buffer of a list are defined uniquely with a combination position number/UTC time. The combination occurs no more than once in the same list. The combination of position number and UTZ time is referred to as RIPID (Ring Buffer ID) in the following.

5.4.1 Continuous retrieval of data

Only one device continuously retrieves data, which is normally from the control center. The RIPID of the last element that was already retrieved is needed for continuous retrieval. It is transferred to the function GetSFSince (or GestSinceWithEvent, see below) as a parameter. The function returns the following values:

- A list of seconds frames [SecondsFrame1...SecondsFrameN]

- The RIPID of the "SecondsFrame0" that is "in front of" the SecondsFrame1 in the ring buffer.

- The RIPID of the SecondsFrameN.

If no elements were lost, the returned RIPID of the SecondsFrame0 is equal to the transferred RIPID. Otherwise, the elements would be lost.

It is possible that all new elements are read from the ring buffer with one call. In this case, elements are still in the buffer and consequently, it is returned as RetCode

SF_FOLLOW. Once all elements have been read out of the buffer, the RetCode_SF-NOFFOLLW is returned.

Function GetSFSinceWithEvent (400:103) is only intended for continuous retrieval. For this function, not only is data retrieved in the same manner as for GetSFSince, but in addition, an event is active that is triggered as soon as the list (starting with the element returned last) exceeds a certain fill level. So, the start position for the EventOnFull mechanism is highlighted.

5.4.2 Spontaneous retrieval of parts of the ring buffer

Normally, system access points and maintenance terminals trigger spontaneous retrieval of the ring buffer. Spontaneous retrieval is characterized by the fact that the maintenance terminal does not "know" which element should be retrieved at which position. However, the maintenance terminal does know the time range during which the retrieval should take place. In order to be able to retrieve data from the time range, we suggest take the following steps (T_{Start} = start time of the time range; T_{Stop} = end time of the time range).

- The maintenance terminal calls up the function GetSFSince, but with T_{Start} -1 and a guaranteed invalid position, the zero value.
- GetSFSince returns a series of seconds frames. Because only the quantity of the seconds frames is transferred, it can either be the last element at the time <=T_{Stop} or > as T_{Stop}
- As long as the time is the last element <=T_{Stop}, additional seconds frames with GetSFSince are retrieved from the ring buffer with the aid of the RIPID of the last returned seconds frame.

The function GetSFSincWithEvent is off limits for the system access! This is only permitted for those to which the EventDestination shows. There is no way to filter results by values. If necessary, this can be done in the terminal itself.

5.4.3 Triggered data retrieval

There are three different types of triggers that can be sent from the lists in the field device to the control center:

- OnFull once the fill level is reached
- OnInvalidate if the EventDestination changes
- OnInsert when entering certain dynamic values from started tasks.

The control center is responsible for reacting appropriately to events. The events do not contain any attribute data, but instead "only" the sender.

Typically, the second frames must be retrieved from the relevant lists in the event of OnFull and OnInsert.

The response for OnInvalidate strongly depends on the inner state of the control center. It makes sense that the change to the EventDestination takes place for the following events:

- The control center "failed" and its function is assumed by the replacement control center.
- Another device besides the control center created its own list and wants to use it.
- The control center delegates treating certain lists to another device.

5.4.4 Retrieving a dataset immediately after it occurs

There are two different requirements that are both covered: The tasks are started if only some tasks should be retrieved immediately from a list, e.g. for messages only errors and critical errors. In this case, the event is triggered if the task writes a value into the ring buffer.

If all data is always retrieved from a list after it was entered, the fill level of 0% can be set as well. If this is the case, the event OnFull is triggered after each new dataset is entered.

5.5 Determining a change in the list

You can determine whether the list was changed externally (via a system access point, etc.) or if it was changed during a system fault:

The control center (or the maintenance terminal) must remember the list version for each list. If there is any change in the list, i.e. if there is

- a change in the task,
- a change in the task element;
- a task element was added; and
- a task was added

the list version is changed. The list version is always increased to the next increment, so after a reset as well. If the entire list information should be deleted after a power outage, the version is installed with 0.

Warning: If tasks are merely started and stopped, the list version remains unchanged.

The list version does not change until the stopped task has restarted.

5.6 Changing a task while it is running

Note: Pre-defined tasks cannot be changed.

The following steps must be taken to change the task while it is running:

- The task is stopped by the control center (4xx:122).
- The task is changed in the device. (The list version doesn't change here either)
- The task is restarted by the control center. If there is a change, the list version in incrementally increased (4xx121).

As soon as the task has restarted, the list version is set to the latest version. The dynamic values of this task do not need to be entered.

The field device rejects all change tasks that do not originate from the EventDestination so the control center does not need to worry about the lists being changed unexpectedly by the maintenance terminals.

5.7 Synchronizing after a transfer fault

If a GetSFSince fails, simply repeat GetSFSince.

If a GetSFSinceWithEvent fails, the function GetSFSinceWithEvent must be repeated accordingly.

6 Figure

7 Glossary

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

OCIT-O_Basis_V3.0_D01