PRAESENSA

Public Address and Voice Alarm System

en Open Interface programming instructions

DISCLAIMER

Although every effort has been made to ensure the information and data contained in these Open Interface programming instructions is correct, no rights can be derived from the contents.

Bosch Security Systems B.V. disclaim all warranties with regard to the information provided in these instructions. In no event shall Bosch Security Systems B.V. be liable for any special, indirect or consequential damages whatsoever resulting from loss of use, data or profits, whether in action of contract, negligence or other tortious action, arising out of or in connection with the use of the information provided in these Open Interface programming instructions.

DOCUMENT HISTORY

Date	Version	Reason
06-12-2019	V1.00	1st edition

Table of Contents

PART 1 - OPEN INTERFACE PROTOCOL

Part 1 of the Open Interface programming instructions describes the Open Interface protocol of the PRAESENSA system.

1. INTRODUCTION

1.1 Purpose

Part 1 of this document describes the native communication interface of the PRAESENSA Public Address and Voice Alarm System.

1.2 Scope

This document is intended for persons, who want to integrate PRAESENSA in their applications with the PRAESENSA native communication interface. They must have knowledge about:

- The PRAESENSA system and its installation (see the PRAESENSA configuration and installation manuals).
- The TCP/IP protocol and how to communicate using TCP/IP.
- Optionally, the TLS protocol and how to secure communication using TLS (when using a secure connection).

Part 1 does not describe the high-level communication (Application Programming Interfaces, API). Refer to 'Part 2 - Open Interface Library' of this document for information about controlling PRAESENSA with high-level Microsoft Windows™ based languages. In this case Part 1 of this document does not apply.

NOTE: It is not possible to derive any rights from this document regarding the programming interface. Extensions and improvements on the Open Interface can be introduced in new versions of the PRAESENSA.

IMPORTANT: PRAESENSA (system controller) is able to communicate with up to twenty (20) Open Interface clients at the same time. This includes connection to Logging Servers.

1.3 Definitions, Acronyms and Abbreviations

SC	PRAESENSA system controller
OI	Open Interface
PA	Public address
OIP	Open Interface Protocol
LSB	Least Significant Byte
MSB	Most Significant Byte

1.4 References

The reference that must be used for this document is:

- The PRAESENSA configuration manual.
- Part 2 of this document > Open Interface Library.

1.5 Overview

Chapter 2 is a general introduction to the document.

Chapter 3 describes the protocol used for the communication

Chapter 4 describes the command messages to trigger functionality on the PRAESENSA system.

Chapter 5 describes the response messages to be expected after a command transmission.

Chapter 6 describes the notification messages sent by the PRAESENSA system.

Chapter 7 describes the diagnostic event structures.

Chapter 8 describes the event originator structures.

Chapter 9 gives an overview of all constants used in the open interface protocol and application messages.

Chapter 10 gives an overview of all error codes that can be sent back in the response messages.

1.6 How to read this document

In this document many messages are described which should be transmitted over the TCP connection. The description of each message is divided into several (optional) subsections. The meaning of each section is described below:

- **Purpose:**
  A global description of the purpose of the message. In case a group of messages is described (all using the same message structure), a short description is given for each message.
- **Parameter structure:**
  The parameters related to the message. When the message requires no parameters, no structure is described here.
- **Response message type:**
  In case the message is a command, the system controller returns a response message. In this section the response message type is referenced. Note that the described message is only valid when the response signals that the command succeeded without errors.
  Beside the described response messages it is also possible that the MESSAGETYPE_OIP_ResponseProtocolError is returned in case on protocol level a failure is detected.
- **Update notifications:**
  The notification messages that can be generated during the execution of the remote function. When there are no related notifications, then this part will be omitted.
- **Related messages:**
  The related messages in conjunction with the message described.

2. APPLICATION CONTROL OVERVIEW

2.1 Calls

2.1.1 Introduction

As PRAESENSA is a Public Address and Voice Alarm System, it is used to distribute background music, live speech and evacuation messages. All audio in the system is distributed in the form of calls.

2.1.2 Components

A call always consists of the following components:

- Priority (refer to chapter 2.1.3)
- Call content (refer to chapter 2.1.4)
- Routing (refer to chapter 2.1.5)

2.1.3 Priority

To each call, a priority is assigned. When two or more calls are addressed to the same zone or need shared resources (e.g. the message player), the system only starts the call with the highest priority. The range of priorities that is available for a call depends on the type of call.

Priority	Call type
0 to 31	BGM (Background Music) calls.
32 to 223	Business calls.
224 to 255	Emergency calls.

2.1.4 Call content

The content of a BGM call typically consists of an audio signal coming from a BGM source, such as a CD player or a tuner. The content of business calls and emergency calls can consist of:

- A start chime (optional).
- Pre-recorded message(s) (optional)
- Live speech (optional).
- An end chime (optional).

The major difference between business calls and emergency calls is that emergency calls can put the system in the emergency state.

2.1.5 Routing

The routing of the call is the set of zones to which the call is intended to be addressed. Whether the call actually is addressed to the selected zones depend on the priority of the call and its partiality (refer to chapter 4.5).

2.1.6 Restart call

A call can be configured to be automatically restarted when it is aborted by the system (for example when all zones are removed from the call). If the restart call option is not configured, the system will automatically restart all emergency calls without live-speech.

The restart call option matches the ‘Continue call’ option on the PRAESENSA call definition configuration page.

2.2 Diagnostics

As PRAESENSA is an emergency compliant system, it monitors its equipment and signals activity performed on the system.

Systems connected to a PRAESENSA system can subscribe to activity and equipment signals for long term storage and reporting facilities. To receive events from the PRAESENSA system, the connected system must subscribe. The following groups are identified:

- General Events
- Call Events
- Fault Events

After subscription for a group, all events currently present in the storage of the PRAESENSA system are sent. Existing events are signaled with the action-type OIACT_EXISTING. The last existing event is signaled with the action type OIACT_EXISTING_LAST. If the connected system subscribes to receive fault events and there are no fault events in the storage of the PRAESENSA system, the PRAESENSA system responds with a message with the action type OIACT_EXISTING_LAST and an event of type DET_NoFaults (refer to chapter 7.4.13). The event itself does not represent an actual system fault and is supposed to be ignored.
Newly created and updated events are notified conform the action described in chapter 9.2.9.

Fault events can be acknowledged and reset by the connected system. The system can choose to acknowledge all fault or specific faults.

2.3 Hardware connection

The communication between PRAESENSA and your system is based on top of a TCP/IP connection (refer to the next figure).

Over the TCP/IP connection, messages can be transmitted between your system and PRAESENSA. To set-up the TCP/IP connection, you must:

- Use the IP address of the system controller.
- Use port number 9401 (non-secure) or 9403 (secure).

3. PROTOCOL CONSIDERATIONS

3.1 Set-up a connection

After PRAESENSA has been started, the system controller listens to port **9401** and **9403**. The set-up of the TCP/IP connection must originate from your system using the **IP address** of the system controller and port **9401** or port **9403**. The connection between the PRAESENSA system and your system is based on a stream connection. This implies that messages may be transferred using multiple packets.

Port **9401** is used for non-secure connections and port **9403** is used for secure connections. For secure connections, TLS 1.2 is used. The PRAESENSA system uses a self-signed certificate file and is available for download from the PRAESENSA configuration web page (see the PRAESENSA configuration manual). A new certificate is generated for the system each time the system controller is reset to defaults. After the socket connect has been established, the login message (`MESSAGETYPE_OIP_Login`) is expected before any other message. The login message passes the **user name and password** to PRAESENSA for verification. If either the user name or the password is incorrect, an error is reported back. In this case, the socket connection is disconnected on demand of the system controller. If the user name and password are correct, all control functions of PRAESENSA become available.

3.2 Heartbeat

After the connection between your system and PRAESENSA has been established, the system controller of PRAESENSA starts the heartbeat checks of your system. The system controller checks if a message is received within 15 seconds after the last message. When the time between two messages is more than 15 seconds, the system controller considers the connection to be broken and closes the TCP/IP connection to your system.

It is advised to also run heartbeat checks of PRAESENSA on your system. To signal that the connection is still present, you must transmit a "`MESSAGETYPE_OIP_KeepAlive`" message (refer to chapter 3.4.3.3) to the system controller every 5 seconds when no other messages are ready for transmission.

3.3 Response times

When your system sends a message to the system controller, a response can be expected within 10 seconds. If your system does not receive a response within 10 seconds, your system can consider the connection to be broken.

3.4 Message format

3.4.1 Introduction

The communication between your system and PRAESENSA is based on messages. This chapter describes the structures that are used in the data field of the messages for PRAESENSA.

3.4.2 General Message Layout

Each message must have this layout:

MessageType	Length	Data

Defined in (c-style) structure format:

```
struct {
    DWORD  messageType;    // Message Type
    UINT   length;         // Message Length
    BYTE   data[];         // Message Data (length – 8 bytes)
};
```

Where:

messageType The “message-type”, which describes the content of the actual data passed. Refer to the various message-type definitions in

	sections below (§4, §5 and §6)
length	The total length of the message in number of bytes, including the sizes of the message-type and length. The length must match the actual transmitted size of bytes. Since the *MessageType* and the *length* are always present, the minimum size of the message is 8 bytes. The maximum size of a message is 128 Kbytes.
data	Data corresponding to the description of the message-type. The data represents a structure which format is explained hereafter together with the message-type.

NOTE:
The length of a specific message-type may vary due to the variable data. For example, when a message contains multiple strings, the length also depends on the sum of the sizes of the strings.

3.4.3 Conventions

In the sections and chapters below several structures are defined. These structures are defined using standard data types, which have defined sizes and usage. The following data types will be used:

3.4.3.1 Basic data types

BOOLEAN:	a 1 byte unsigned value with the values FALSE = 0 and TRUE = 1.
CHAR:	a 1 byte type representing an ASCII character.
BYTE:	a 1 byte unsigned value with the range 0 ... 255.
WORD:	a 2 byte unsigned value with the range 0 ... 65535.
SHORT:	a 2 byte signed value with the range -32768 ... 32767.
INT:	a 4 bytes signed value with the range $-(2^{31})$ ... $(2^{31}-1)$.
UINT:	a 4 byte unsigned value with the range 0 ... $(2^{32}-1)$.
LONG:	a 4 bytes signed value with the range $-(2^{31})$ ... $(2^{31}-1)$.
DWORD:	a 4 byte unsigned value with the range 0 ... $(2^{32}-1)$.

NOTE:
All numbers are represented in the little-endian[1] format. Between the data-type is no alignment present.

3.4.3.2 Variable length Data types

Beside the basic data type, variable length data types are used within the messages. In this section the variable length data types are described in term of basic data types.

String

A string is used to pass ASCII text within a message. A string is always variable in length.

```
struct {
    UINT    length
    CHAR    chars[Length];
} STRING;
```

Where:

length	String length in bytes (characters). Strings are limited in length to a maximum of 64 Kbytes. Note that the size of the length *parameter* is not included in the length.
chars	Actual string, not zero terminated.

Time structure

A time structure represents the date and time. It is generated by the PRAESENSA system. The time is mostly passed along with diagnostic events (see section §7) to indicate the actual date and time of creation and other changes.

[1] Little endian is a storage mechanism where the least significant byte is stored on the lowest address, followed by the more significant bytes. E.g. a WORD is represented in memory as two consecutive bytes where the LSB is stored on the lowest address and the MSB on the next address. For transmission over TCP, the LSB byte is transmitted first, followed by the MSB bytes

```
struct {
    DWORD    time;
} TIME;
```

Where:

time	UTC time in seconds since 1 January 1970, 00:00:00 hour.

Complex structure

Message can refer to structural information. These structures by itself described a complete set of information and will be described in the corresponding sections. The basic format of each structure is as follows:

```
struct {
    DWORD    structureType;   // Type of the structure.
    UINT     length;          // structure Length.
    BYTE     data[]           // structure data (length – 4 bytes)
} structureHeader;
```

Where:

structureType	Defines the “structure Type”, which describes the content of the structure data passed. Refer to the various structure type definitions in the sections below (§7 and §7.4.13).
length	The total length of the structure in number of bytes, including the sizes of the structure type and length. The length should match the actual transmitted size of bytes.
data	Data corresponding to the description of the structure-type. The data represents a structure which format is explained with the structure-type.

3.4.3.3 Comma separated lists

Commands sent to the PRAESENSA system do not accept spaces around the separation commas in lists of strings. However, notifications and results sent from the PRAESENSA system may contain a space after the separation comma.

3.5 Heartbeat message MESSAGETYPE_OIP_KeepAlive

Purpose:

The heartbeat message is a special message, which can be sent to the PRAESENSA system at any time. In normal circumstances the heartbeat message is transmitted every 5 seconds (when nothing else to transmit). The message is used to notify the PRAESENSA system that your system is still alive. The PRAESENSA system also sends heartbeat messages to indicate that the PRAESENSA system is still operational. You must check if two successive messages are received within 15 seconds.

Note that the heartbeat message is similar to the notification messages.

Parameter structure:

```
struct {
    DWORD         messageType;
    UINT          length;
    UINT          reserved1;
    UINT          reserved2;
} OIP_KeepAlive;
```

Where:

messageType	The message type indicator for the heartbeat message. Constant value `MESSAGETYPE_OIP_KeepAlive` (See section 9).
length	The total length of the Heartbeat message (16 bytes for this message).
reserved1	Session sequence number. Currently the *reserved1* is not used and should be set to the value zero (0).
reserved2	Message sequence number. Currently the *reserved2* is not used and should be set to the value zero (0).

3.6 Protocol fault message MESSAGETYPE_OIP_ResponseProtocolError

Purpose:

Any message sent towards the PRAESENSA system is checked against its boundaries (message size, string size, validity of the message-type, not logged in …). In case a mismatch is detected regarding the size, a universal error response message is returned. Response message as described in section 5 cannot be used, because the received message is not decoded nor processed.

Parameter structure:

```
struct {
    DWORD         messageType;
    UINT          length;
    UINT          reserved1;
    UINT          reserved2;
    UINT          errorCode;
    UINT          errorPosition;
} OIP_ResponseProtocolError;
```

Where:

messageType	The message type indicator for the message. Constant value `MESSAGETYPE_OIP_ResponseProtocolError` (See section 9).
length	The total length of the Protocol fault message (24 bytes for this message).
reserved1	Session sequence number. Currently the *reserved1* is not used and should be set to the value zero (0).
reserved2	Message sequence number. Currently the *reserved2* is not used and should be set to the value zero (0).
errorCode	The error code of the received message. For the possible error codes see section 10.
errorPosition	The byte offset in the message stream, where the fault is detected.

Related messages:

Any message received by the PRAESENSA system and is not conform the message guideline as described in 3.4.

3.7 Buffer overflow

Purpose:

Messages ready for transmission from the PRAESENSA system are queued. In case the receive speed of the connected system is too low, the queue may overflow (dependent on the number of generated events, resource update, etc.). Since the queue consumes internal PRAESENSA system resources, overflow detection is present, which disconnects the communication interface when the queue overflows its limit.

This may result in a loss of received events.

4. COMMAND MESSAGES

4.1 Introduction

Command messages can be sent to control the PRAESENSA system. Commands always result in a response from the PRAESENSA system. The expected response is referenced with each command or the generic response `MESSAGETYPE_OIP_ResponseProtocolError` is returned in case the message is corrupted. Each command message starts with a fixed number of fields, which are presented below in structure format.

NOTE:
In the time between the transmission of the command message and the reception of the response message, the PRAESENSA system can send notification messages.

```
struct {
    DWORD         messageType;
    UINT          length;
    UINT          reserved1;
    UINT          reserved2;
} COMMANDHEADER;
```

Where:

messageType	The command message type as documented in the sections below.
length	The total length of the command structure.
reserved1	Session sequence number. Currently the *reserved1* is not used and should be set to the value zero (0)
reserved2	Message sequence number. Currently the *reserved2* is not used and should be set to the value zero (0).

NOTE:
The initial two elements (refer to section 3.4.2) are repeated in the structure.

4.2 MESSAGETYPE_OIP_Login

Purpose:

Logs in on the PRAESENSA system with a user name and password.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    STRING           userName;
    STRING           password;
} OIP_Login;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_Login.
userName	The user name to logon with.
password	The password to logon with.

Response message type:

MESSAGETYPE_OIP_Response

4.3 MESSAGETYPE_OIP_GetNcoVersion

Purpose:

Gets the software release of the PRAESENSA system.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_GetNcoVersion;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetNcoVersion.

Response message type:

MESSAGETYPE_OIP_ResponseGetNcoVersion

4.4 MESSAGETYPE_OIP_GetProtocolVersion

Purpose:

Gets the protocol version of the Open Interface of the PRAESENSA system. Should be used to verify that your system is compatible with the PRAESENSA system.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_GetProtocolVersion;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetProtocolVersion.

Response message type:

MESSAGETYPE_OIP_ResponseGetProtocolVersion

4.5 MESSAGETYPE_OIP_CreateCallEx2

Purpose:

Creates (but does not start) a call with the given parameters.

Parameter structure:

```
struct {
    COMMANDHEADER          header;
    UINT                   priority;
    TOICallOutputHandling  outputHandling;
    TOICallStackingMode    stackingMode;
    UINT                   stackingTimeout;
    BOOLEAN                liveSpeech;
    UINT                   repeat;
    STRING                 routing;
    STRING                 startChime;
    STRING                 endChime;
    STRING                 audioInput;
    STRING                 messages;
    TOICallTiming          callTiming;
    STRING                 preMonitorDest;
    UINT                   liveSpeechAttenuation;
    UINT                   startChimeAttenuation;
    UINT                   sendChimeAttenuation;
    UINT                   messageAttenuation;
} OIP_CreateCallEx2;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_CreateCallEx2.
priority	Priority of the call. Ranges: 0 … 31: BGM call priority. Always partial call, regardless the partial setting. 32 … 223: Normal call priority. 224 … 255: Emergency call priority. Always partial call, regardless the partial setting.
outputHandling	Whether the call is partial, non-partial or stacked. There are three possible values: OICOH_PARTIAL, OICOH_NON_PARTIAL and OICOH_STACKED. See §9.2.11 for the value set description. Settings the output handling to anything other than OICOH_PARTIAL will result in ERROR_INVALID_PARAMETERS.
stackingMode	Whether a stacked call waits for all zones to become available or a stacked call waits for each zone to become available for replay. There are two possible values: OICSM_WAIT_FOR_ALL and OICSM_WAIT_FOR_EACH. See §9.2.12 for the value set

	description. This parameter is ignored when outputHandling is set to : OICOH_PARTIAL or OICOH_NON_PARTIAL.
stackingTimeout	Amount of minutes for a stacked call to wait for available resources. The time-out countdown is started at the moment the original call has ended. The accepted range is 1 to 255 minutes; the value OICST_INFINITE is used to wait infinitely. This parameter is ignored when outputHandling is set to OICOH_PARTIAL or OICOH_NON_PARTIAL.
liveSpeech	Whether or not the call has a live speech phase. `TRUE` = live speech, `FALSE` = no live speech.
repeat	How many times the messages should be repeated. Value can be: -1: Repeat infinity. 0: Play Message once. 1 … 32767: Repeat count. Note that the value 1 indicates one repeat, so the message is played twice.
routing	List of names of zone groups, zones and/or control outputs. The routing is formatted as a comma separated set of resource names. No spaces are allowed before or after the separation commas in the string.
startChime	The name of the start chime. May be empty, no leading or trailing spaces are allowed.
endChime	The name of the end chime. May be empty, no leading or trailing spaces are allowed.
audioInput	Name of the audio Input (only used when live speech is true). No leading or trailing spaces are allowed.
messages	List of names of prerecorded messages. The messages parameter is formatted as a comma separated set of message names. May be empty, but no spaces are allowed before or after the separation commas.
callTiming	Indicates the way the call must be handled. There are three possible values: OICTM_IMMEDIATE, OICTM_TIME_SHIFTED and OICTM_MONITORED. See §9.2.13 for the value set description. Setting the call timing to anything other than OICTM_IMMEDIATE will result in ERROR_INVALID_PARAMETERS.
preMonitorDest	The destination zone of the pre-monitor phase of a pre-monitored call. When the call is not pre-monitored, this value is ignored. This parameter is ignored when callTiming is set to OICTM_IMMEDIATE or OICTM_TIME_SHIFTED.
liveSpeechAttenuation	The attenuation to be used for the audio input during the live speech phase. Range: 0..60 dB.
startChimeAttenuation	The attenuation to be used for the chime generator during the start chime phase. Range: 0..60 dB.
endChimeAttenuation	The attenuation to be used for the chime generator during the end chime phase. Range: 0..60 dB.
messageAttenuation	The attenuation to be used for the message generator during the start prerecorded message phase. Range: 0..60 dB.

Response message type:

MESSAGETYPE_OIP_ResponseCallId.

Related messages:

MESSAGETYPE_OIP_StartCreatedCall
MESSAGETYPE_OIP_StopCall
MESSAGETYPE_OIP_AbortCall
MESSAGETYPE_OIP_AddToCall
MESSAGETYPE_OIP_RemoveFromCall

4.6 MESSAGETYPE_OIP_CreateCallEx3

Purpose:

Creates (but does not start) a call with the given parameters.

Parameter structure:

```
struct {
    COMMANDHEADER          header;
    UINT                   priority;
    TOICallOutputHandling  outputHandling;
    TOICallStackingMode    stackingMode;
    UINT                   stackingTimeout;
    BOOLEAN                liveSpeech;
    UINT                   repeat;
    STRING                 routing;
    STRING                 startChime;
    STRING                 endChime;
    STRING                 audioInput;
    STRING                 messages;
    TOICallTiming          callTiming;
    STRING                 preMonitorDest;
    UINT                   liveSpeechAttenuation;
    UINT                   startChimeAttenuation;
    UINT                   sendChimeAttenuation;
    UINT                   messageAttenuation;
    BOOLEAN                restartCall;
} OIP_CreateCallEx3;
```

Where:

header — Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_CreateCallEx2.

priority — Priority of the call.
Ranges:
- 0 … 31: BGM call priority. Always partial call, regardless the partial setting.
- 32 … 223: Normal call priority.
- 224 … 255: Emergency call priority. Always partial call, regardless the partial setting.

outputHandling — Whether the call is partial, non-partial or stacked. There are three possible values: OICOH_PARTIAL, OICOH_NON_PARTIAL and OICOH_STACKED. See §9.2.11 for the value set description. Settings the output handling to anything other than OICOH_PARTIAL will result in ERROR_INVALID_PARAMETERS.

stackingMode — Whether a stacked call waits for all zones to become available or a stacked call waits for each zone to become available for replay. There are two possible values: OICSM_WAIT_FOR_ALL and OICSM_WAIT_FOR_EACH. See §9.2.12 for the value set description. This parameter is ignored when outputHandling is set to : OICOH_PARTIAL or OICOH_NON_PARTIAL.

stackingTimeout — Amount of minutes for a stacked call to wait for available resources. The time-out countdown is started at the moment the original call has ended. The accepted range is 1 to 255 minutes; the value OICST_INFINITE is used to wait infinitely. This parameter is ignored when outputHandling is set to OICOH_PARTIAL or OICOH_NON_PARTIAL.

liveSpeech — Whether or not the call has a live speech phase. `TRUE` = live speech, `FALSE` = no live speech.

repeat — How many times the messages should be repeated.
Value can be:
- -1: Repeat infinity.
- 0: Play Message once.
- 1 … 32767: Repeat count.

Note that the value 1 indicates one repeat, so the message is played twice.

routing — List of names of zone groups, zones and/or control outputs. The routing is formatted as a comma separated set of resource names. No spaces are allowed before or after the separation

	commas in the string.
startChime	The name of the start chime. May be empty, no leading or trailing spaces are allowed.
endChime	The name of the end chime. May be empty, no leading or trailing spaces are allowed.
audioInput	Name of the audio Input (only used when live speech is true). No leading or trailing spaces are allowed.
messages	List of names of prerecorded messages. The messages parameter is formatted as a comma separated set of message names. May be empty, but no spaces are allowed before or after the separation commas.
callTiming	Indicates the way the call must be handled. There are three possible values: OICTM_IMMEDIATE, OICTM_TIME_SHIFTED and OICTM_MONITORED. See §9.2.13 for the value set description. Setting the call timing to anything other than OICTM_IMMEDIATE will result in ERROR_INVALID_PARAMETERS.
preMonitorDest	The destination zone of the pre-monitor phase of a pre-monitored call. When the call is not pre-monitored, this value is ignored. This parameter is ignored when callTiming is set to OICTM_IMMEDIATE or OICTM_TIME_SHIFTED.
liveSpeechAttenuation	The attenuation to be used for the audio input during the live speech phase. Range: 0..60 dB.
startChimeAttenuation	The attenuation to be used for the chime generator during the start chime phase. Range: 0..60 dB.
endChimeAttenuation	The attenuation to be used for the chime generator during the end chime phase. Range: 0..60 dB.
messageAttenuation	The attenuation to be used for the message generator during the start prerecorded message phase. Range: 0..60 dB.
restartCall	Indicates if the call should be restarted after an interruption.

Response message type:

MESSAGETYPE_OIP_ResponseCallId.

Related messages:

MESSAGETYPE_OIP_StartCreatedCall
MESSAGETYPE_OIP_StopCall
MESSAGETYPE_OIP_AbortCall
MESSAGETYPE_OIP_AddToCall
MESSAGETYPE_OIP_RemoveFromCall

4.7 MESSAGETYPE_OIP_StartCreatedCall

Purpose:

Starts a previously created call. If the call was started successfully, call state update notification messages are sent.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    TOICallId        callId;
} OIP_StartCreatedCall;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_StartCreatedCall.
callId	Identification of the call, returned by `createCallEx2` (§4.5) and `createCallEx3` (§4.6). See §9.2.2 for the value set description.

Response message type:

MESSAGETYPE_OIP_Response

Update notifications:

MESSAGETYPE_OIP_NotifyCall
MESSAGETYPE_OIP_NotifyResources

Related messages:

MESSAGETYPE_OIP_CreateCall
MESSAGETYPE_OIP_StopCall
MESSAGETYPE_OIP_AbortCall
MESSAGETYPE_OIP_AddToCall
MESSAGETYPE_OIP_RemoveFromCall

4.8 MESSAGETYPE_OIP_StopCall

Purpose:

Stops a previously created or started call.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    TOICallId        callId;
} OIP_StopCall;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_StopCall.
callId	Identification of the call, returned by `createCallEx2` (§4.5) or `createCallEx3` (§4.6). See §9.2.2 for the value set description.

Response message type:

MESSAGETYPE_OIP_Response

Update notifications:

MESSAGETYPE_OIP_NotifyCall
MESSAGETYPE_OIP_NotifyResources

Related messages:

MESSAGETYPE_OIP_CreateCallEx2
MESSAGETYPE_OIP_CreateCallEx3
MESSAGETYPE_OIP_StartCreatedCall
MESSAGETYPE_OIP_AbortCall
MESSAGETYPE_OIP_AddToCall
MESSAGETYPE_OIP_RemoveFromCall

4.9 MESSAGETYPE_OIP_AbortCall

Purpose:

Aborts a previously created or started call.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    TOICallId        callId;
} OIP_AbortCall;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_AbortCall.
callId	Identification of the call, returned by `createCallEx2` (§4.5) or `createCallEx3` (§4.6). See §9.2.2 for the value set description.

Response message type:

MESSAGETYPE_OIP_Response

Update notifications:

MESSAGETYPE_OIP_NotifyCall
MESSAGETYPE_OIP_NotifyResources

Related messages:

MESSAGETYPE_OIP_CreateCallEx2
MESSAGETYPE_OIP_CreateCallEx3
MESSAGETYPE_OIP_StartCreatedCall
MESSAGETYPE_OIP_StopCall
MESSAGETYPE_OIP_AddToCall
MESSAGETYPE_OIP_RemoveFromCall

4.10 MESSAGETYPE_OIP_AddToCall

Purpose:

Adds routing to a previously created or started call.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    TOICallId        callId;
    STRING           routing;
} OIP_AddToCall;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_AddToCall.
callId	Identification of the call, returned by `createCallEx2` (§4.5) or `createCallEx3` (§4.6). See §9.2.2 for the value set description.
routing	List of names of zone groups, zones and/or control outputs to be added to the call. A comma separates each name in the routing list. No spaces are allowed before or after the separation commas in the string.

Response message type:

MESSAGETYPE_OIP_Response

Update notifications:

MESSAGETYPE_OIP_NotifyResources

Related messages:

MESSAGETYPE_OIP_CreateCallEx2
MESSAGETYPE_OIP_CreateCallEx3
MESSAGETYPE_OIP_StartCreatedCall
MESSAGETYPE_OIP_StopCall
MESSAGETYPE_OIP_AbortCall
MESSAGETYPE_OIP_RemoveFromCall

4.11 MESSAGETYPE_OIP_RemoveFromCall

Purpose:

Removes routing from a previously created or started call.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    TOICallId        callId;
    STRING           routing;
} OIP_RemoveFromCall;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_RemoveFromCall.
callId	Identification of the call, returned by `createCallEx2` (§4.5) or `createCallEx3` (§4.6). See §9.2.2 for the value set description.
routing	List of names of zone groups, zones and/or control outputs to be removed from the call. A comma separates each name in the routing list. No spaces are allowed before or after the separation commas in the string.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyResources

Related messages:
MESSAGETYPE_OIP_CreateCallEx2
MESSAGETYPE_OIP_CreateCallEx3
MESSAGETYPE_OIP_StartCreatedCall
MESSAGETYPE_OIP_StopCall
MESSAGETYPE_OIP_AbortCall
MESSAGETYPE_OIP_AddToCall

4.12 MESSAGETYPE_OIP_CancelAll

Purpose:
Cancels all available stacked calls that were started by this connection.

Parameter structure:
```
struct {
    COMMANDHEADER    header;
} OIP_CancelAll;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_CancelAll.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyCall
MESSAGETYPE_OIP_NotifyResources

Related messages:
MESSAGETYPE_OIP_CreateCallEx2
MESSAGETYPE_OIP_CreateCallEx3
MESSAGETYPE_OIP_StartCreatedCall
MESSAGETYPE_OIP_CancelLast

4.13 MESSAGETYPE_OIP_CancelLast

Purpose:
Cancels (if still available) the last stacked call that was started by this connection.

Parameter structure:
```
struct {
    COMMANDHEADER    header;
} OIP_CancelLast;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_CancelLast.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyCall
MESSAGETYPE_OIP_NotifyResources

Related messages:
MESSAGETYPE_OIP_CreateCallEx2
MESSAGETYPE_OIP_CreateCallEx3
MESSAGETYPE_OIP_StartCreatedCall
MESSAGETYPE_OIP_CancelAll

4.14 MESSAGETYPE_OIP_AckAllFaults

Purpose:

Acknowledges all fault events. Because the fault alarm depends on the states of all fault events, it also acknowledge the fault alarm. If the start of the fault alarm changes state, it results in the message MESSAGETYPE_OIP_NotifyAlarm (if subscribed, see §6.3).

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_AckAllFaults;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_AckAllFaults.

Response message type:

MESSAGETYPE_OIP_Response

Update notifications:

MESSAGETYPE_OIP_NotifyAlarm (alarm-type equals OIAT_FAULT)
MESSAGETYPE_OIP_NotifyDiagEvent

Related messages:

MESSAGETYPE_OIP_ResetAllFaults
MESSAGETYPE_OIP_ReportFault

4.15 MESSAGETYPE_OIP_ResetAllFaults

Purpose:

Resets all fault events. Because the fault alarm depends on the state of all fault events, this can possibly reset the fault alarm, when the faults are resolved. If the fault alarm changes state, it results in the message MESSAGETYPE_OIP_NotifyAlarm (if subscribed, see §6.3).

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_ResetAllFaults;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ResetAllFaults.

Response message type:

MESSAGETYPE_OIP_Response

Update notifications:

MESSAGETYPE_OIP_NotifyAlarm (alarm-type equals OIAT_FAULT)
MESSAGETYPE_OIP_NotifyDiagEvent

Related messages:

MESSAGETYPE_OIP_AckAllFaults
MESSAGETYPE_OIP_ReportFault

4.16 MESSAGETYPE_OIP_ReportFault

Purpose:

Reports a general fault diagnostics event in the system. The fault is reported as a User-Injected-Fault, which is notified as diagnostic event DET_UserInjectedFault.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    STRING           description;
} OIP_ReportFault;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ReportFault.
description	Textual representation of the fault to be reported.

Response message type:
MESSAGETYPE_OIP_ResponseReportFault

Update notifications:
MESSAGETYPE_OIP_NotifyAlarm (alarm-type equals OIAT_FAULT)
MESSAGETYPE_OIP_NotifyDiagEvent

Related messages:
MESSAGETYPE_OIP_AckAllFaults
MESSAGETYPE_OIP_ResetAllFaults
MESSAGETYPE_OIP_AckFault
MESSAGETYPE_OIP_ResolveFault
MESSAGETYPE_OIP_ResetFault

4.17 MESSAGETYPE_OIP_AckFault

Purpose:
Acknowledges a specific diagnostic fault event. Because the fault alarm depends on the states of all fault events, it can possibly acknowledge the state of the fault alarm of the system (in case it was the last non-acknowledged fault). If the state of the fault alarm changes, it results in the message MESSAGETYPE_OIP_NotifyAlarm (if subscribed, see §6.3).

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    TOIEventId       eventId;
} OIP_AckFault;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_AckFault.
eventId	Identification of the diagnostic fault event. See §9.2.1 for the value set description.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyAlarm (alarm-type equals OIAT_FAULT)
MESSAGETYPE_OIP_NotifyDiagEvent

Related messages:
MESSAGETYPE_OIP_AckAllFaults
MESSAGETYPE_OIP_ResetAllFaults
MESSAGETYPE_OIP_ReportFault
MESSAGETYPE_OIP_ResolveFault
MESSAGETYPE_OIP_ResetFault

4.18 MESSAGETYPE_OIP_ResolveFault

Purpose:
Resolves the fault injected by with the message MESSAGETYPE_OIP_ReportFault. The received `eventId` of the `reportFault` message is the parameter.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    TOIEventId       eventId;
} OIP_ResolveFault;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ResolveFault.
eventId	Identification of the diagnostic fault event, received by the MESSAGETYPE_OIP_ResponseReportFault message. See §9.2.1 for the value set description.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyDiagEvent

Related messages:
MESSAGETYPE_OIP_ReportFault
MESSAGETYPE_OIP_AckFault
MESSAGETYPE_OIP_ResetFault

4.19 MESSAGETYPE_OIP_ResetFault

Purpose:
Resets a specific diagnostic fault event. Because the fault alarm depends on the states of all fault events, it can possibly reset the state of the fault alarm of the system (in case it was the last non-reset fault). If the state of the fault alarm changes, it results in the message MESSAGETYPE_OIP_NotifyAlarm (if subscribed, see §6.3).

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    TOIEventId       eventId;
} OIP_ResetFault;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ResetFault.
eventId	Identification of the diagnostic fault event. See §9.2.1 for the value set description.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyAlarm (alarm-type equals OIAT_FAULT)
MESSAGETYPE_OIP_NotifyDiagEvent

Related messages:
MESSAGETYPE_OIP_AckAllFaults
MESSAGETYPE_OIP_ResetAllFaults
MESSAGETYPE_OIP_AckFault
MESSAGETYPE_OIP_ReportFault
MESSAGETYPE_OIP_ResolveFault

4.20 MESSAGETYPE_OIP_AckEvacAlarm

Purpose:
This message acknowledges the emergency alarm. If the state of the emergency alarm changes, it results in the message MESSAGETYPE_OIP_NotifyAlarm (if subscribed, see §4.31).

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_AckEvacAlarm;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_AckEvacAlarm.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyAlarm (alarm-type equals OIAT_EVAC)

Related messages:
MESSAGETYPE_OIP_ResetEvacAlarmEx

4.21 MESSAGETYPE_OIP_ResetEvacAlarmEx

Purpose:

Resets the emergency alarm. Whether or not running evacuation priority calls are aborted can be specified. If the state of the emergency alarm changes, it results in the message MESSAGETYPE_OIP_NotifyAlarm (if subscribed, see §4.31).

Parameter structure:

```
struct {
    COMMANDHEADER    header;

    BOOLEAN          bAbortEvacCalls
} OIP_ResetEvacAlarmEx;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ResetEvacAlarmEx.
bAbortEvacCalls	Whether or not currently running evacuation priority calls must be aborted. `TRUE` = abort running evacuation priority calls, `FALSE` = do not abort running evacuation priority calls

Response message type:

MESSAGETYPE_OIP_Response

Update notifications:

MESSAGETYPE_OIP_NotifyAlarm (alarm-type equals OIAT_EVAC)

Related messages:

MESSAGETYPE_OIP_AckEvacAlarm

MESSAGETYPE_OIP_ResetEvacAlarm

4.22 MESSAGETYPE_OIP_IncrementBgmVolume

Purpose:

Increments the BGM volume of the passed routing with 3 dB.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    STRING           routing;
} OIP_IncrementBgmVolume;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_IncrementBgmVolume.
routing	List of names of zone groups and/or zones. A comma separates each name in the routing list. No spaces are allowed before or after the separation commas in the string.

Response message type:

MESSAGETYPE_OIP_Response

Related messages:

MESSAGETYPE_OIP_DecrementBgmVolume
MESSAGETYPE_OIP_SetBgmVolume

4.23 MESSAGETYPE_OIP_IncrementBgmChannelVolume

Purpose:

Increments the BGM volume of a channel with 3 dB.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    STRING           channel;
} OIP_IncrementBgmChannelVolume;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_IncrementBgmVolume.

channel The BGM channel name as present in the PRAESENSA configuration.

Response message type:
MESSAGETYPE_OIP_Response

Related messages:
MESSAGETYPE_OIP_DecrementBgmChannelVolume
MESSAGETYPE_OIP_SetBgmVolume
MESSAGETYPE_OIP_GetBgmChannelNames

4.24 MESSAGETYPE_OIP_DecrementBgmVolume

Purpose:
Decrements the BGM volume of the passed routing with 3 dB.

Parameter structure:
```
Struct {
    COMMANDHEADER    header;
    STRING           routing;
} OIP_DecrementBgmVolume;
```

Where:

header Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_DecrementBgmVolume.

routing List of names of zone groups and/or zones. A comma separates each name in the routing list. No spaces are allowed before or after the separation commas in the string.

Response message type:
MESSAGETYPE_OIP_Response

Related messages:
MESSAGETYPE_OIP_IncrementBgmVolume
MESSAGETYPE_OIP_SetBgmVolume

4.25 MESSAGETYPE_OIP_DecrementBgmChannelVolume

Purpose:
Decrements the BGM volume of a channel with 3 dB.

Parameter structure:
```
Struct {
    COMMANDHEADER    header;
    STRING           channel;
} OIP_DecrementBgmChannelVolume;
```

Where:

header Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_DecrementBgmChannelVolume.

channel The BGM channel name as present in the PRAESENSA configuration.

Response message type:
MESSAGETYPE_OIP_Response

Related messages:
MESSAGETYPE_OIP_IncrementBgmChannelVolume
MESSAGETYPE_OIP_SetBgmVolume
MESSAGETYPE_OIP_GetBgmChannelNames

4.26 MESSAGETYPE_OIP_SetBgmVolume

Purpose:
Sets the BGM volume of the given routing.

Parameter structure:
```
struct {
    COMMANDHEADER    header;
```

```
    INT             volume;
    STRING          routing;
} OIP_SetBgmVolume;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_SetBgmVolume.
volume	Volume of the BGM. Value range: 0 ... –96 (dB). Use –96 (dB) to mute the BGM.
routing	List of names of zone groups and/or zones. A comma separates each name in the routing list. No spaces are allowed before or after the separation commas in the string.

Response message type:

MESSAGETYPE_OIP_Response

Related messages:

MESSAGETYPE_OIP_IncrementBgmVolume
MESSAGETYPE_OIP_DecrementBgmVolume

4.27 MESSAGETYPE_OIP_AddBgmRouting

Purpose:

Adds a routing to a BGM channel. Either all specified routing is added or, in case of an error, no routing at all.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    STRING           channel;
    STRING           routing;
} OIP_AddBgmRouting;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_AddBgmRouting.
channel	The BGM channel name as present in the PRAESENSA configuration.
routing	List of names of zone groups and/or zones. A comma separates each name in the routing list. No spaces are allowed before or after the separation commas in the string.

Response message type:

MESSAGETYPE_OIP_Response

Update notifications:

MESSAGETYPE_OIP_NotifyBgmRouting

Related messages:

MESSAGETYPE_OIP_RemoveBgmRouting
MESSAGETYPE_OIP_ToggleBgmRouting
MESSAGETYPE_OIP_SetBgmRouting

4.28 MESSAGETYPE_OIP_RemoveBgmRouting

Purpose:

Removes a routing from a BGM channel. Either all specified routing is removed or, in case of an error, no routing at all.

Parameter structure:

```
Struct {
    COMMANDHEADER    header;
    STRING           channel;
    STRING           routing;
} OIP_RemoveBgmRouting;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_RemoveBgmRouting.

channel	The BGM channel name as present in the PRAESENSA configuration.
routing	List of names of zone groups and/or zones. A comma separates each name in the routing list. No spaces are allowed before or after the separation commas in the string.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyBgmRouting

Related messages:
MESSAGETYPE_OIP_AddBgmRouting
MESSAGETYPE_OIP_ToggleBgmRouting
MESSAGETYPE_OIP_SetBgmRouting

4.29 MESSAGETYPE_OIP_ToggleBgmRouting

Purpose:
Toggles a routing in a BGM channel. When none of names in the specified routing are part the BGM channel, all specified routing is added, else all supplied routing is removed or, in case of an error, the current routing of the BGM channel remains unchanged.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    STRING           channel;
    STRING           routing;
} OIP_ToggleBgmRouting;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ToggleBgmRouting.
channel	The BGM channel name as present in the PRAESENSA configuration.
routing	List of names of zone groups and/or zones. A comma separates each name in the routing list. No spaces are allowed before or after the separation commas in the string.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyBgmRouting

Related messages:
MESSAGETYPE_OIP_AddBgmRouting
MESSAGETYPE_OIP_RemoveBgmRouting
MESSAGETYPE_OIP_SetBgmRouting

4.30 MESSAGETYPE_OIP_SetBgmRouting

Purpose:
Sets the routing of a BGM channel. Note that the specified routing replaces the configured routing in the configuration of the PRAESENSA system.

Parameter structure:

```
Struct {
    COMMANDHEADER    header;
    STRING           channel;
    STRING           routing;
} OIP_SetBgmRouting;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_SetBgmRouting.
channel	The BGM channel name as present in the PRAESENSA configuration.

routing	List of names of zone groups and/or zones. A comma separates each name in the routing list. No spaces are allowed before or after the separation commas in the string.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyBgmRouting

Related messages:
MESSAGETYPE_OIP_AddBgmRouting
MESSAGETYPE_OIP_RemoveBgmRouting
MESSAGETYPE_OIP_ToggleBgmRouting

4.31 MESSAGETYPE_OIP_SetSubscriptionAlarm

Purpose:
Subscribes or unsubscribes to alarm notifications. Depending on the *alarmtype* parameter, it subscribes to faults or emergency alarms. Only when a subscription is set for the faults or emergency alarm, state notifications will be sent. When a subscription is set, the MESSAGETYPE_OIP_NotifyAlarm message is sent with the current state of the alarm.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    TOIAlarmType     alarmType;
    BOOLEAN          subscription;
} OIP_SetSubscriptionAlarm;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_SetSubscriptionAlarm.
alarmType	The alarm type to subscribe of unsubscribe, see §9.2.3.
subscription	Whether to subscribe or unsubscribe. `TRUE` = subscribe, `FALSE` = unsubscribe

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyAlarm

4.32 MESSAGETYPE_OIP_SetSubscriptionResources

Purpose:
Subscribes or unsubscribes to resource (read zone groups, zones or control outputs) state notifications of particular resources. Only when a subscription is set for a resource, resource state notifications are sent for that resource. When a subscription is set for a resource, the MESSAGETYPE_OIP_NotifyResources message is sent with the current state of that resource.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    STRING           resourceNames;
    BOOLEAN          subscription;
} OIP_SetSubscriptionResources;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_SetSubscriptionResources.
resourceNames	List of names of zone groups, zones and/or control outputs. A comma separates each name in the routing list. Resources already having the subscription state are ignored. No spaces are allowed before or after the separation commas in the string.
subscription	Whether to subscribe or unsubscribe. `TRUE` = subscribe, `FALSE` = unsubscribe.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyResources

4.33 MESSAGETYPE_OIP_SetSubscriptionResourceFaultState

Purpose:
Subscribes or unsubscribes to resource (read zone groups or zones) fault state notifications of particular resources for faults that affect the audio distribution of that zone or zone group. Only when a subscription is set for a resource, resource fault state notifications are sent for that resource. When a subscription is set for a resource, the MESSAGETYPE_OIP_NotifyResourceFaultState message is sent with the current state of that resource.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    STRING           resourceNames;
    BOOLEAN          subscription;
} OIP_SetSubscriptionResourceFaultState;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_SetSubscriptionResourceFaultState.
resourceNames	List of names of zone groups and/or zones. A comma separates each name in the routing list. Resources already having the subscription state are ignored. No spaces are allowed before or after the separation commas in the string. Subscription for control output resources is not allowed.
subscription	Whether to subscribe or unsubscribe. `TRUE` = subscribe, `FALSE` = unsubscribe.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyResourceFaultState

4.34 MESSAGETYPE_OIP_SetSubscriptionBgmRouting

Purpose:
Subscribes or unsubscribes to BGM routing notifications. Only when a subscription is set for a BGM channel, BGM routing notifications are sent for that BGM channel. When a subscription is set for a BGM channel, the MESSAGETYPE_OIP_NotifyBgmRouting message is sent with the routing of that BGM channel and with the *addition* parameter set to `TRUE`.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    STRING           channel;
    BOOLEAN          subscription;
} OIP_SetSubscriptionBgmRouting;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_SetSubscriptionBgmRouting.
channel	The BGM channel name as present in the PRAESENSA configuration.
subscription	Whether to subscribe or unsubscribe. `TRUE` = subscribe, `FALSE` = unsubscribe.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:

MESSAGETYPE_OIP_NotifyBgmRouting

4.35 MESSAGETYPE_OIP_SetSubscriptionEvents

Purpose:

Subscribes or unsubscribes to diagnostic event notifications. Only when a subscription is set for an event group, diagnostic event notifications are sent for that group. When a subscription is set for an event group, the MESSAGETYPE_OIP_NotifyDiagEvent message is sent with the diagnostic event of that group.

Parameter structure:

```
struct {
    COMMANDHEADER      header;
    TDiagEventGroup    eventGroup;
    BOOLEAN            subscription;
} OIP_SetSubscriptionEvents;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_SetSubscriptionEvents.
eventGroup	Group identification of the diagnostic events. The associated event-types for each group is represented in §9.3.2.
subscription	Whether to subscribe or unsubscribe. `TRUE` = subscribe, `FALSE` = unsubscribe.

Response message type:

MESSAGETYPE_OIP_Response

Update notifications:

MESSAGETYPE_OIP_NotifyDiagEvent

4.36 MESSAGETYPE_OIP_SetSubscriptionBgmVolume

Purpose:

Subscribes or unsubscribes to BGM volume notifications. Only when a subscription is set for zone, BGM volume notifications are sent for that zone. When a subscription is set for a zone, the MESSAGETYPE_OIP_NotifyBgmVolume message is sent with the current volume of that zone.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    STRING           zones;
    BOOLEAN          subscription;
} OIP_SetSubscriptionBgmVolume;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_SetSubscriptionBgmVolume.
zones	The zone names as present in the PRAESENSA configuration.
subscription	Whether to subscribe or unsubscribe. `TRUE` = subscribe, `FALSE` = unsubscribe.

Response message type:

MESSAGETYPE_OIP_Response

Update notifications:

MESSAGETYPE_OIP_NotifyBgmVolume

4.37 MESSAGETYPE_OIP_GetZoneNames

Purpose:

Retrieve the configured zone names from the PRAESENSA system. When the zone group parameter is empty all zone names are returned otherwise the zone names in that zone group are returned.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    STRING           zonegroup;
} OIP_GetZoneNames;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetZoneNames.
zoneGroup	The zone group to get the names of.

Response message type:

MESSAGETYPE_OIP_ResponseNames

4.38 MESSAGETYPE_OIP_GetZoneGroupNames

Purpose:

Retrieve the configured zone group names from the PRAESENSA system.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_GetZoneGroupNames;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetZoneGroupNames.

Response message type:

MESSAGETYPE_OIP_ResponseNames

4.39 MESSAGETYPE_OIP_GetMessageNames

Purpose:

Retrieve the configured message names from the PRAESENSA system. Note that the chimes on the PRAESENSA system are also messages, so the names of the chimes will be part of the response to the GetMessageNames.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_GetMessageNames;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetMessageNames.

Response message type:

MESSAGETYPE_OIP_ResponseNames

4.40 MESSAGETYPE_OIP_GetChimeNames

Purpose:

Retrieve the configured message names from the PRAESENSA system. Note that this is the same list as returned by MESSAGETYPE_OIP_GetMessageNames.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_GetChimeNames;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetChimeNames.

Response message type:

MESSAGETYPE_OIP_ResponseNames

4.41 MESSAGETYPE_OIP_GetAudioInputNames

Purpose:

Retrieve the configured audio input names from the PRAESENSA system.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_GetAudioInputNames;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetAudioInputNames.

Response message type:

MESSAGETYPE_OIP_ResponseNames

4.42 MESSAGETYPE_OIP_GetBgmChannelNames

Purpose:

Retrieve the configured BGM channel names from the PRAESENSA system.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_GetBgmChannelNames;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetBgmChannelNames.

Response message type:

MESSAGETYPE_OIP_ResponseNames

4.43 MESSAGETYPE_OIP_GetConfigId

Purpose:

Retrieve the configuration identifier from the PRAESENSA system. This is a number which is increased each time the configuration is saved.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_GetConfigId;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetConfigId.

Response message type:

MESSAGETYPE_OIP_ResponseConfigId

4.44 MESSAGETYPE_OIP_ActivateVirtualControlInput

Purpose:

Activate a control input. If the virtual control input is already active then activating it again will not have any effect.

Parameter structure:

```
struct {
    COMMANDHEADER          header;
    STRING                 virtualControlInput;
} OIP_ActivateVirtualControlInput;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ActivateVirtualControlInput.
virtualControlInput	Name of the virtual control input to activate.

Response message type:
MESSAGETYPE_OIP_Response.

Related messages:
MESSAGETYPE_OIP_DeactivateVirtualControlInput

4.45 MESSAGETYPE_OIP_DeactivateVirtualControlInput

Purpose:
Deactivate a virtual control input. If the virtual control input is already inactive then deactivating it again will not have any effect.

Parameter structure:

```
struct {
    COMMANDHEADER                           header;
    STRING                                  virtualControlInput;
    TOIVirtualControlInputDeactivation      deactivationType
} OIP_DeactivateVirtualControlInput;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_DeactivateVirtualControlInput.
virtualControlInput	Name of the virtual control input to deactivate.
deactivationType	Specifier how the associated action should be deactivated (see §9.2.15).

Response message type:
MESSAGETYPE_OIP_Response.

Related messages:
MESSAGETYPE_OIP_ActivateVirtualControlInput

4.46 MESSAGETYPE_OIP_SetSubscriptionUnitCount

Purpose:
Subscribes or unsubscribes to unit count notifications. Only when a subscription is set for the unit count, unit count updates will be sent. When a subscription is set, the MESSAGETYPE_OIP_NotifyUnitCount message is sent with the current number of connected units.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    BOOLEAN          subscription;
} OIP_SetSubscriptionUnitCount;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_SetSubscriptionUnitCount.
subscription	Whether to subscribe or unsubscribe. `TRUE` = subscribe, `FALSE` = unsubscribe.

Response message type:
MESSAGETYPE_OIP_Response

Update notifications:
MESSAGETYPE_OIP_NotifyUnitCount

4.47 MESSAGETYPE_OIP_SetSubscriptionVirtualControlInputs

Purpose:
Subscribes or unsubscribes to virtual control input state notifications. Only when a subscription is set for virtual control inputs, state notifications are sent for virtual control inputs. When a subscription is set, the MESSAGETYPE_OIP_NotifyVirtualControlInputs message is sent with the current state of the virtual control inputs.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
    STRING           virtualControlInputs;
    BOOLEAN          subscription;
} OIP_SetSubscriptionVirtualControlInputs;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_SetSubscriptionVirtualControlInputs.
virtualControlInputs	List of names of virtual control inputs. A comma separates each name in the routing list. Virtual control inputs already having the subscription state are ignored. No spaces are allowed before or after the separation commas in the string.
subscription	Whether to subscribe or unsubscribe. `TRUE` = subscribe, `FALSE` = unsubscribe.

Response message type:

MESSAGETYPE_OIP_Response

Update notifications:

MESSAGETYPE_OIP_NotifyVirtualControlInputs

4.48 MESSAGETYPE_OIP_GetVirtualControlInputNames

Purpose:

Retrieve the configured virtual control input names from the PRAESENSA system.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_GetVirtualControlInputNames;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetVirtualControlInputNames.

Response message type:

MESSAGETYPE_OIP_ResponseNames

4.49 MESSAGETYPE_OIP_GetConfiguredUnits

Purpose:

Retrieve the configured units from the PRAESENSA system. Only the units that are enabled are returned.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_GetConfiguredUnits;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetConfiguredUnits.

Response message type:

MESSAGETYPE_OIP_ResponseUnits

4.50 MESSAGETYPE_OIP_GetConnectedUnits

Purpose:

Retrieve the connected units from the PRAESENSA system. Only the units that are configured, enabled and connected with the correct firmware version (units that can be controlled) are returned.

Parameter structure:

```
struct {
    COMMANDHEADER    header;
} OIP_GetConnectedUnits;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetConnectedUnits.

Response message type:

MESSAGETYPE_OIP_ResponseUnits

5. RESPONSE MESSAGES

5.1 Introduction

The PRAESENSA system returns a response message after a command message has been executed. This section describes the response messages returned in case no protocol failures are detected (see §3.6). Section 5.1 describes the structure of the response messages. In specific cases, the default response structure is extended with additional information.

5.2 MESSAGETYPE_OIP_Response

Purpose:

Defines the general response of the commands that returned an error code and no additional information. It contains the basic information for all response messages.

Parameter structure:

```
struct {
    DWORD           messageType;
    UINT            length;
    UINT            reserved1;
    UINT            reserved2;
    DWORD           errorCode;
} RESPONSEHEADER;
```

Where:

messageType	The response message type, which is equal to MESSAGETYPE_OIP_Response.
length	The total length of the response structure
reserved1	Session sequence number. Currently the *reserved1* is not used and should be set to the value zero (0)
reserved2	Message sequence number. Currently the *reserved2* is not used and should be set to the value zero (0).
errorCode	The error code of the command this is a response for. For the possible error codes see 10.

NOTE:

The initial two elements described in section 3.4.1, are repeated in this structure.

Related messages:

Any command message not described in the sections §5.

5.3 MESSAGETYPE_OIP_ResponseGetNcoVersion

Purpose:

Responses to the command message MESSAGETYPE_OIP_GetNcoVersion.

Parameter structure:

```
struct {
    RESPONSEHEADER    header;
    STRING            version;
} OIP_ResponseGetNcoVersion;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ResponseGetNcoVersion.
version	Release of the system controller software. The release label has no defined format.

Related messages:

MESSAGETYPE_OIP_GetNcoVersion

5.4 MESSAGETYPE_OIP_ResponseGetProtocolVersion

Purpose:

Responses to the command message MESSAGETYPE_OIP_GetProtocolVersion.

Parameter structure:

```
struct {
    RESPONSEHEADER    header;
    STRING            version;
} OIP_ResponseGetProtocolVersion;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ResponseGetProtocolVersion.
version	Version of the Open Interface protocol in the format “M.m”. **Where:** M The major version number m The minor version number

Related messages:

MESSAGETYPE_OIP_GetProtocolVersion

5.5 MESSAGETYPE_OIP_ResponseCallId

Purpose:

Responses to the command message MESSAGETYPE_OIP_CreateCallEx2 and MESSAGETYPE_OIP_CreateCallEx3.

Parameter structure:

```
struct {
    RESPONSEHEADER    header;
    UINT              callId;
} OIP_ResponseCallId;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ResponseCallId.
callId	Unique identification of the call, which can be used in the call-handling commands.

Related messages:

MESSAGETYPE_OIP_CreateCallEx2
MESSAGETYPE_OIP_CreateCallEx3
MESSAGETYPE_OIP_StartCreatedCall
MESSAGETYPE_OIP_StopCall
MESSAGETYPE_OIP_AbortCall
MESSAGETYPE_OIP_AddToCall
MESSAGETYPE_OIP_RemoveFromCall

5.6 MESSAGETYPE_OIP_ResponseReportFault

Purpose:

Response to the command message MESSAGETYPE_OIP_ReportFault.

Parameter structure:

```
struct {
    RESPONSEHEADER    header;
    TOIEventId        eventId;
} OIP_ResponseReportFault;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ResponseReportFault.
eventId	Unique identification of the fault event, which can be used in the event handling commands.

Related messages:

MESSAGETYPE_OIP_ReportFault
MESSAGETYPE_OIP_AckFault
MESSAGETYPE_OIP_ResolveFault
MESSAGETYPE_OIP_ResetFault

5.7 MESSAGETYPE_OIP_ResponseNames

Purpose:

Responses to the command messages MESSAGETYPE_OIP_GetXXXNames.

Parameter structure:

```
struct {
    RESPONSEHEADER    header;
    STRING            names;
} OIP_ResponseNames;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ResponseNames.
names	The requested names of the items. A comma separates each name in the list.

Related messages:

MESSAGETYPE_OIP_GetZoneNames
MESSAGETYPE_OIP_GetZoneGroupNames
MESSAGETYPE_OIP_GetMessageNames
MESSAGETYPE_OIP_GetChimeNames
MESSAGETYPE_OIP_GetAudioInputNames
MESSAGETYPE_OIP_GetBgmChannelNames
MESSAGETYPE_OIP_GetVirtualControlInputNames

5.8 MESSAGETYPE_OIP_ResponseConfigId

Purpose:

Responses to the command message MESSAGETYPE_OIP_GetConfigId.

Parameter structure:

```
struct {
    RESPONSEHEADER    header;
    UINT              configId;
} OIP_ResponseGetConfigId;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_GetConfigId.
configId	Unique identification of the call, which can be used in the call-handling commands.

Related messages:

MESSAGETYPE_OIP_GetConfigId

5.9 MESSAGETYPE_OIP_ResponseUnits

Purpose:

Responses to the command message MESSAGETYPE_OIP_GetXXXUnits.

Parameter structure:

```
struct {
    RESPONSEHEADER    header;
    STRING            units;
} OIP_ResponseUnits;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_ResponseUnits
units	Comma (,) separated list of unit names with serial number, Formatted as name(serialnumber).

Related messages:

MESSAGETYPE_OIP_GetConfiguredUnits
MESSAGETYPE_OIP_GetConnectedUnits

6. NOTIFICATION MESSAGES

6.1 Introduction

The PRAESENSA system notifies you system about the changes of the states of various resources (e.g. calls, zones). Each notification message starts with a fixed number of fields, which are presented below in structure format.

```
struct {
    DWORD         messageType;
    UINT          length;
    UINT          reserved1;
    UINT          reserved2;
} NOTIFYHEADER;
```

Where:

messageType	The notification message type as documented in the sections below.
length	The total length of the notification structure.
reserved1	Session sequence number. Currently the *reserved1* is not used and should be set to the value zero (0)
reserved2	Message sequence number. Currently the *reserved2* is not used and should be set to the value zero (0).

NOTE:
The initial two elements described in section 3.4.1, are repeated in this structure.

6.2 MESSAGETYPE_OIP_NotifyCall

Purpose:

Sent when the state of a running call, started by this Open Interface connection changes. Note that this notification does not report state changes for calls started on Call-Stations or other Open Interface connections.

Parameter structure:

```
struct {
    NOTIFYHEADER      header;
    UINT              callId;
    TOICallState      callState;
} OIP_NotifyCall;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_NotifyCall.
callId	Unique identification of the call, which changed its state.
callState	The new state of the call. See §9.2.6 for the definitions of the call states.

Related messages:

MESSAGETYPE_OIP_StartCreatedCall
MESSAGETYPE_OIP_StopCall
MESSAGETYPE_OIP_AbortCall

6.3 MESSAGETYPE_OIP_NotifyAlarm

Purpose:

Sent when the state of an alarm changes and there is a subscription to the specific type of alarm.

Parameter structure:

```
struct {
    NOTIFYHEADER     header;
    TOIAlarmType     alarmType;
    TOIAlarmState    alarmState;
} OIP_NotifyAlarm;
```

Where:

header	Header of the message, where the *messageType* element is

	equal to MESSAGETYPE_OIP_NotifyAlarm.
alarmType	The type of alarm, which changed its state. See §9.2.3 for the different types.
alarmState	The new state of the alarm. See §9.2.4 for the definitions of the alarm states.

Related messages:

MESSAGETYPE_OIP_SetSubscriptionAlarm

6.4 MESSAGETYPE_OIP_NotifyResources

Purpose:

Sent when the state of resources (zone groups, zones, control outputs) change and there is a subscription to notifications of resources.

Parameter structure:

```
struct {
    NOTIFYHEADER          header;
    TOIResourceState      resourceState;
    UINT                  priority;
    UINT                  callId;
    STRING                resources;
} OIP_NotifyResources;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_NotifyResources.
resourceState	The new state of the resource. See §9.2.5 for the definitions of the resource states.
priority	The priority of the call using the resource when the state is OIRS_INUSE. Not used (no valid) when the resource become free (state OIRS_FREE).
callId	Identification of the call, which uses the resource. The value is OI_UNDEFINED_CALLID when the resource is freed.
resources	List of names of zone groups, zones and/or control outputs. A comma separates each name in the routing list.

Related messages:

MESSAGETYPE_OIP_SetSubscriptionResources

6.5 MESSAGETYPE_OIP_NotifyResourceFaultState

Purpose:

Sent when the fault state of resources (zone groups, zones) for faults that affect the audio distribution of that zone or zone group changes and there is a subscription to fault notifications of resources.

Parameter structure:

```
struct {
    NOTIFYHEADER          header;
    TOIResourceFaultState resourceFaultState;
    STRING                resources;
} OIP_NotifyResourceFaultState;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_NotifyResourceFaultState.
resourceFaultState	The new state of the resource. See §9.2.6 for the definitions of the resource fault states.
resources	List of names of zone groups, zones and/or control outputs. A comma separates each name in the routing list.

Related messages:

MESSAGETYPE_OIP_SetSubscriptionResourceFaultState

6.6 MESSAGETYPE_OIP_NotifyBgmRouting

Purpose:

Sent when the routing of a BGM channel changes and there is subscription to notifications of BGM channels.

Parameter structure:

```
struct {
    NOTIFYHEADER      header;
    BOOL              addition;
    STRING            channel;
    STRING            routing;
} OIP_ NotifyBgmRouting;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_NotifyBgmRouting.
addition	Whether the routing was added (`TRUE`) or removed (`FALSE`).
channel	The name of the BGM channel, which routing was changed.
routing	List of names of zone groups, zones and/or control outputs that were added or removed. A comma separates each name in the routing list.

Related messages:

MESSAGETYPE_OIP_SetSubscriptionBgmRouting
MESSAGETYPE_OIP_SetBgmRouting
MESSAGETYPE_OIP_AddBgmRouting
MESSAGETYPE_OIP_RemoveBgmRouting

6.7 MESSAGETYPE_OIP_NotifyEvent

Purpose:

Sent when a diagnostic event is added or updated and there is a subscription to notification of diagnostic events.

Parameter structure:

```
struct {
    NOTIFYHEADER     header;
    TOIActionType    action;
    DIAGEVENT        diagnosticEvent;
} OIP_NotifyDiagEvent;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_NotifyDiagEvent.
action	Indicates what happened with the diagnostic event. See §9.2.8 for the action definitions.
diagnosticEvent	Diagnostic event information. See chapter 7 for the descriptions of the diagnostic information.

Related messages:

MESSAGETYPE_OIP_SetSubscriptionEvents

6.8 MESSAGETYPE_OIP_NotifyBgmVolume

Purpose:

Sent when the volume of a BGM zone changes and there is subscription to notifications of BGM zones.

Parameter structure:

```
struct {
    NOTIFYHEADER      header;
    STRING            zone;
    INT               volume;
} OIP_ NotifyBgmVolume;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_NotifyBgmRouting.

zone	The name of the BGM zone, which volume was changed.
volume	The new volume of the zone.

Related messages:

MESSAGETYPE_OIP_SetSubscriptionBgmVolume
MESSAGETYPE_OIP_IncrementBgmVolume
MESSAGETYPE_OIP_DecrementBgmVolume
MESSAGETYPE_OIP_SetBgmVolume

6.9 MESSAGETYPE_OIP_NotifyUnitCount

Purpose:

Sent when the number of connected units has changed.

Parameter structure:

```
struct {
    NOTIFYHEADER     header;
    UINT             numberConnected;
} OIP_ NotifyUnitCount;
```

Where:

header	Header of the message, where the *messageType* element is equal to MESSAGETYPE_OIP_NotifyUnitCount.
numberConnected	The number of connected units.

Related messages:

MESSAGETYPE_OIP_SetSubscriptionUnitCount

6.10 MESSAGETYPE_OIP_NotifyVirtualControlInputState

Purpose:

Sent when the state of one or more virtual control inputs has changed state.

Parameter structure:

```
struct {
    NOTIFYHEADER                    header;
    STRING                          virtualControlInputs;
    TOIVirtualControlInputState     state;
} OIP_ NotifyVirtualControlInputState;
```

Where:

header	Header of the message, where the messageType element is equal to MESSAGETYPE_OIP_NotifyVirtualControlInputState.
virtualControlInputs	List of names of virtual control inputs of which the state has changed. A comma separates each name in the list.
state	The state of the virtual control inputs. See §9.2.16 for the definitions of the states.

Related messages:

MESSAGETYPE_OIP_SetSubscriptionVirtualControlInputs
MESSAGETYPE_OIP_activateVirtualControlInput
MESSAGETYPE_OIP_deactivateVirtualControlInput

7. DIAGNOSTIC EVENTS STRUCTURES

7.1 Introduction

The PRAESENSA system uses diagnostic event for reporting signals and faults that are detected within the system. The diagnostic events can be divided into three groups:

- General Events
  Events to signal user action or system changes. All generic events are without state, which means that they just notify the event.
- Call Events
  Signals the activity of calls. Call events are like general events, but they specifically report about calls.
- Fault Events
  Signals problems detected within the PRAESENSA system. Faults have states for the user and the equipment, reporting the fault event. Fault events influences the systems fault mode, reported by the message MESSAGETYPE_OIP_NotifyAlarm.

The diagnostic events are embedded in the MESSAGETYPE_OIP_NotifyEvent message, but since the event is variable in length, follows the complex structure rule as described in §3.4.3.2.

Each diagnostic event structure contains a fixed number of fields, which are described below.

```
struct {
    TDiagEventType     diagMessageType;
    UINT               length;
    TDiagEventGroup    diagEventGroup;
    TOIEventId         diagEventId;
    TDiagEventState    diagEventState;
    TIME               addTimeStamp;
    TIME               acknowledgeTimeStamp;
    TIME               resolveTimeStamp;
    TIME               resetTimeStamp;
    ORIGINATOR         addEventOriginator;
    ORIGINATOR         acknowledgeEventOriginator;
    ORIGINATOR         resolveEventOriginator;
    ORIGINATOR         resetEventOriginator;
} DIAGEVENTHEADER;
```

Where:

diagMessageType	The message type indicator for the diagnostic structure as defined in 9.4. In the sections below the various diagnostic event types are described.
length	The total length of the diagnostic event information (including the *diagMessageType , length* and the additional information as described for a specific diagnostic event type)
diagEventGroup	The group to which the event belongs. See §9.3.2 for the diagnostic group definitions.
diagEventId	The identification of the event as generated by the PRAESENSA system.
diagEventState	The state of the event.
addTimeStamp	Time of creation (add to the system) of the diagnostic event.
acknowledgeTimeStamp	Time of acknowledgement by a user of the diagnostic event. On creation filled with value zero.
resolveTimeStamp	Time of resolving the problem by the event-creator of the diagnostic event. On creation filled with value zero.
resetTimeStamp	Time of reset by a user of the diagnostic event. On creation filled with value zero.
addEventOriginator	The originator that created (add to the system) the event.
acknowledgeEventOriginator	The originator that acknowledged the event, filled when

	acknowledged. On creation filled with value structure OIEOT_NoEventOriginator.
resolveEventOriginator	The originator that resolved the event, filled when resolved. On creation filled with value structure OIEOT_NoEventOriginator.
resetEventOriginator	The originator that reset the event, filled when reset. On creation filled with value structure OIEOT_NoEventOriginator.

Note: the event originator information is described in §7.4.13.

7.2 General Diagnostic Events

This section describes the general diagnostic event types. For each diagnostic event is either the structure defined, or a reference to the structure definition.

Since a general diagnostic event is stateless, several elements in the `DIAGEVENTHEADER` structure have default values:

- The *diagEventState* is always set to the value DES_NEW (See §9.3.1)
- The time stamps for Acknowledge, Resolve and Reset are set to no time (value 0).
- The originators for Acknowledge, Resolve and Reset are set to the type OIEOT_NoEventOriginator

7.2.1 DET_EvacAcknowledge

Purpose:

This diagnostic event indicates that the system emergency state is acknowledged.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_EvacAcknowledge.

7.2.2 DET_EvacReset

Purpose:

This diagnostic event indicates that the system emergency state is reset.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_EvacReset.

7.2.3 DET_EvacSet

Purpose:

This diagnostic event indicates that the system emergency state is set (activated).

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_EvacSet.

7.2.4 DET_UnitConnect

Purpose:

This diagnostic event indicates that a unit has connected to or disconnected from the PRAESENSA system.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_UnitConnect.

7.2.5 DET_SCStartup

Purpose:

This diagnostic event indicates that the PRAESENSA system has started.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_SCStartup.

7.2.6 DET_OpenInterfaceConnect

Purpose:

This diagnostic event indicates that a remote system has connected to the PRAESENSA system using the open interface.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_OpenInterfaceConnect.

7.2.7 DET_OpenInterfaceDisconnect

Purpose:

This diagnostic event indicates that a remote system has disconnected from the PRAESENSA system using the open interface.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_OpenInterfaceDisconnect.

7.2.8 DET_OpenInterfaceConnectFailed

Purpose:

This diagnostic event indicates that a remote system has attempted to connect to the PRAESENSA system using the open interface but failed.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_OpenInterfaceConnectFailed.

7.2.9 DET_CallLoggingSuspended

Purpose:

This diagnostic event indicates that call logging has been suspended because of a logging queue overflow.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_CallLoggingSuspended.

7.2.10 DET_CallLoggingResumed

Purpose:

This diagnostic event indicates that call logging has been resumed.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_CallLoggingResumed.

7.2.11 DET_UserLogIn

Purpose:

This diagnostic event Indicates that a user has logged in..

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_UserLogIn.

7.2.12 DET_UserLogOut

Purpose:

This diagnostic event indicates that a user has logged out..

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_UserLogOut.

7.2.13 DET_UserLogInFailed

Purpose:

This diagnostic event indicates that a login attempt has failed.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_UserLogInFailed.

7.2.14 DET_BackupPowerModeStart

Purpose:

This diagnostic event indicates that backup power mode has started.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_BackupPowerModeStart. This event is only generated when backup power mode (in the system settings) has been configured **not** to generate a fault event.

7.2.15 DET_BackupPowerModeEnd

Purpose:

This diagnostic event indicates that backup power mode has ended.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_BackupPowerModeEnd. This event is only generated when backup power mode (in the system settings) has been configured **not** to generate a fault event.

7.2.16 DET_ConfigurationRestored

Purpose:

This diagnostic event Indicates that the configuration on the system controller has been restored from a backup.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    BOOLEAN            configurationRestored;
    BOOLEAN            securityConfigurationRestored;
    BOOLEAN            messagesRestored;
} ConfigurationRestoredDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_BoosterSpareSwitch.
configurationRestored	Whether the configuration is restored.
securityConfigurationRestored	Whether the security configuration is restored.
messagesRestored	Whether the messages are restored.

7.2.17 DET_DemoteToBackup

Purpose:

This diagnostic event indicates that the current duty controller in a redundant system detected a critical fault and demoted itself to backup.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_DemoteToBackup.

7.3 Call Diagnostic Events

This section describes the call diagnostic event types. For each diagnostic event either the structure is defined, or a reference to the structure definition.

Since a call diagnostic event is stateless, the same default values are used as described in §7.2.

7.3.1 DET_CallStartDiagEventV2

Purpose:

This diagnostic event indicates the start of a call in the PRAESENSA system.

Parameter structure:

```
struct {
    DIAGEVENTHEADER       header;
    UINT                  callId;
    STRING                audioInput;
    STRING                endChime;
    BOOLEAN               liveSpeech;
    STRING                messageNames;
    STRING                outputNames;
    UINT                  priority;
    STRING                startChime;
    UINT                  messageRepeat;
    STRING                macroName;
    UINT                  originalCallId;
    TOICallOutputHandling outputHandling;
    TOICallTiming         callTiming;
    UINT                  reserved;
} CallStartDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_CallStartDiagEventV2.
audioInput	The names of the audio input used in this call.
endChime	The names of the end chimes used in this call.
liveSpeech	Whether or not this call has live speech.
messageNames	List of names of prerecorded messages used in this call. A comma separates each name in the list.
outputNames	List of names of zone groups, zones and/or control outputs used in the call. A comma separates each name in the routing list.
Priority	The priority of the call. See §4.5 for the value description of the priority.
startChime	The names of the start chimes used in this call.
messageRepeat	The repeat count of the messages in the call. See §4.5 for the value description of the repeat count.
callId	Identification of the call.
macroName	The name of the macro used in this call.
originalCallId	Identification of the original call in case of a replay.
outputHandling	Whether the call is ‘partial’, ‘non-partial’ or ‘stacked’. Partial calls are calls that proceed even in case not all required zones are available. Stacked calls are calls that extend partial calls with replays to previously unavailable zones.
callTiming	Whether the call should start ‘immediate’, ‘time-shifted’ or ‘pre-monitored’.
reserved	Parameter only used for internal processing.

7.3.2 DET_CallEndDiagEventV2

Purpose:

This diagnostic event indicates the end (or abort) of a call in the PRAESENSA system.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_CallStartDiagEventV2.

Parameter structure:

```
struct {
    DIAGEVENTHEADER      header;
    UINT                 callId;
    TOICallState         callStateCompleted;
    BOOLEAN              callAborted;
    TOICallStopReason    callStopReason;
    UINT                 reserved;
} CallEndDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_CallChangeResourceDiagEventV2.
callId	Identification of the call.
callStateCompleted	The last completed call state the moment the call is stopped or aborted. See §9.2.7 for the definitions of the call states.
callAborted	Whether a call was aborted. TRUE = call is aborted, FALSE = the call is stopped.
callStopReason	Why the call was stopped or aborted. See §9.2.8 for the definitions of the call stop reasons.
reserved	Parameter only used for internal processing.

7.3.3 DET_CallChangeResourceDiagEventV2

Purpose:

This diagnostic event indicates a change in routing of a running call. The diagnostic event indicates whether zone groups, zones and/or control outputs are added to the routing or removed from the routing.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               callId;
    STRING             removedResourceNames;
    STRING             addedResourceNames;
} CallChangeResourceDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_CallChangeResourceDiagEvent.
callId	Identification of the call.
removedResourceNames	List of names of zone groups, zones and/or control outputs removed from the call. A comma separates each name in the routing list.
addedResourceNames	List of names of zone groups, zones and/or control outputs added to the call. A comma separates each name in the routing list.

7.3.4 DET_CallTimeoutDiagEventV2

Purpose:

This diagnostic event indicates that a stacked call has reached its time-out point and implies that the call has been unable to reach all required zones. The diagnostic event provides the unreached zones.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               callId;
    STRING             unreachedResourceNames;
} CallTimeoutDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_CallTimeoutDiagEvent.
callId	Identification of the call.
unreachedResourcesNames	List of names of zones that were not reached during the extended call.

7.3.5 DET_CallRestartDiagEvent

Purpose:

This diagnostic event indicates the restart of a call in the PRAESENSA system. The diagnostic event is only logged when the call was reset earlier (see §7.3.6).

Parameter structure:

```
struct {
    DIAGEVENTHEADER       header;
    UINT                  callId;
    STRING                audioInput;
    STRING                endChime;
    BOOLEAN               liveSpeech;
    STRING                messageNames;
    STRING                outputNames;
    UINT                  priority;
    STRING                startChime;
    UINT                  messageRepeat;
    STRING                macroName;
    UINT                  originalCallId;
    TOICallOutputHandling outputHandling;
    TOICallTiming         callTiming;
    UINT                  reserved;
} CallRestartDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_CallRestartDiagEvent.
audioInput	The names of the audio input used in this call.
endChime	The names of the end chimes used in this call.
liveSpeech	Whether or not this call has live speech.
messageNames	List of names of prerecorded messages used in this call. A comma separates each name in the list.
outputNames	List of names of zone groups, zones and/or control outputs used in the call. A comma separates each name in the routing list.
Priority	The priority of the call. See §4.5 for the value description of the priority.
startChime	The names of the start chimes used in this call.
messageRepeat	The repeat count of the messages in the call. See §4.5 for the value description of the repeat count.
callId	Identification of the call.
macroName	The name of the macro used in this call.
originalCallId	Identification of the original call in case of a replay.
outputHandling	Whether the call is ‘partial’, ‘non-partial’ or ‘stacked’. Partial calls are calls that proceed even in case not all required zones are available. Stacked calls are calls that extend partial calls with replays to previously unavailable zones.
callTiming	Whether the call should start ‘immediate’, ‘time-shifted’ or ‘pre-monitored’.
reserved	Parameter only used for internal processing.

7.3.6 DET_CallResetDiagEvent

Purpose:

This diagnostic event indicates the reset of a call in the PRAESENSA system. A call can only be reset (and restarted) if the ‘Continue call’ setting in the Call Macro is set to ‘After interruption’. If a call is reset, the call state is set to OICS_IDLE (see §9.2.7) and the call will be restarted.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_CallResetDiagEvent.

Parameter structure:

```
struct {
    DIAGEVENTHEADER      header;
    UINT                 callId;
    TOICallState         callStateCompleted;
    TOICallResetReason   callResetReason;
    UINT                 reserved;
} CallResetDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_CallResetDiagEvent.
callId	Identification of the call.
callStateCompleted	The active call state the moment the call is reset. See §9.2.7 for the definitions of the call states.
callResetReason	Why the call was reset. See §**Error! Reference source not found.**9.2.8 for the definitions of the call reset reason.
reserved	Parameter only used for internal processing.

7.4 Fault Diagnostic Events

This section describes the fault diagnostic event types. For each diagnostic event either the structure is defined, or a reference to the structure definition.

The creation of a fault within a fault-less system changes the system to the fault mode. This indicates that the PRAESENSA system requires maintenance. The maintenance engineer acknowledges the faults and takes appropriate action to repair the faults. When the system detects that the faults are resolved, the fault-diagnostic events resolve their fault. Finally, the maintenance engineer should reset the fault to bring the system in normal operation mode.

Each fault diagnostic event passes several states, which are all notified. The link between related faults is controlled by the *diagEventId* element in the header of the diagnostic event (see structure in section 7).

7.4.1 DET_AudioPathSupervision

Purpose:

This diagnostic event indicates that an audio-path failure is detected.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_AudioPathSupervision.

7.4.2 DET_MicrophoneSupervision

Purpose:

This diagnostic event indicates that a microphone failure is detected. Note that this diagnostic event only applies to a Call Station.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_MicrophoneSupervision

7.4.3 DET_SystemInputContact

Purpose:

This diagnostic event indicates that a system input contact failure is detected.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_SystemInputContact.

7.4.4 DET_CallStationExtension

Purpose:

This diagnostic event indicates that a mismatch between the number of configured call station extensions and the number of detected call station extensions

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               numberConfigured;
    UINT               numberDetected;
} CallStationExtensionDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_CallStationExtension.
numberConfigured	The number of extensions as configured in the PRAESENSA system configuration
numberDetected	The number of extensions as reported by the call station.

7.4.5 DET_ConfigurationFile

Purpose:

This diagnostic event indicates that a missing or corrupt configuration file is detected.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_ConfigurationFile.

7.4.6 DET_ConfigurationVersion

Purpose:

This diagnostic event indicates that a mismatch between the configuration file version and the required configuration file version is detected. The configuration file requires conversion.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    STRING             expected;
    STRING             loaded;
} ConfigurationVersionDiagEvent;
```

Where:

Header	Header of the event, where the *diagMessageType* element is equal to DET_ConfigurationVersion.
expected	String containing the expected configuration file version
Loaded	String containing the loaded (opened) configuration file version

7.4.7 DET_IllegalConfiguration

Purpose:

This diagnostic event indicates an inconsistency within the active configuration file.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               errorCode;
} IllegalConfigurationDiagEvent;
```

Where:

Header	Header of the event, where the *diagMessageType* element is equal to DET_IllegalConfiguration.
errorCode	The code of the illegal configuration error. Not used at the moment, currently filled with the value ‘0’.

7.4.8 DET_PrerecordedMessagesNames

Purpose:

This diagnostic event indicates that a mismatch is detected between the configured (and used) prerecorded message-names and the detected prerecorded message-names in the PRAESENSA system.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    STRING             missingMessages;
} PrerecordedMessagesNamesDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_PrerecordedMessagesNames.
missingMessages	List of names of prerecorded messages not found in the PRAESENSA system, but used in the configuration. A comma separates each name in the list.

7.4.9 DET_PrerecordedMessagesCorrupt

Purpose:

This diagnostic event indicates that one or more prerecorded messages in the PRAESENSA system is corrupt and cannot be used.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    STRING             corruptMessages;
} PrerecordedMessagesCorruptDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_PrerecordedMessagesCorrupt.
corruptMessages	List of names of corrupt prerecorded messages in the PRAESENSA system. A comma separates each name in the list.

7.4.10 DET_UnitMissing

Purpose:

This diagnostic event indicates a missing unit, which was configured in the PRAESENSA system configuration.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_UnitMissing.

7.4.11 DET_UnitReset

Purpose:

This diagnostic event indicates that a restart of a unit is detected.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    STRING             chipType;
} UnitResetDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_UnitReset.
chipType	The type of the processor that caused is restarted.

7.4.12 DET_UserInjectedFault

Purpose:

This diagnostic event indicates that a fault is injected by a user or a remote system. Note that this diagnostic event message can be triggered by the MESSAGETYPE_OIP_ReportFault as well as by a configured control-input of the PRAESENSA system.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    STRING             errorDescription;
} UserInjectedFaultDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_UserInjectedFault.
errorDescription	A textual description of the error.

Related messages:

MESSAGETYPE_OIP_ReportFault

7.4.13 DET_NoFaults

Purpose:

A diagnostic event of this type does not represent an actual fault, but is used to indicate that there are no fault events present in the logging of the system controller. This event is always sent in a message with the *TOIActionType* equal to OIACT_EXISTING_LAST (See §9.2.9).

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_NoFaults and the *diagEventId* is equal to zero.

7.4.14 DET_ZoneLineFault

Purpose:

This diagnostic event indicates that a Zone Line Fault that is injected by a remote system by triggering configured control input.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    STRING             zoneNames;
} ZoneLineFaultDiagEvent;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_ZoneLineFault.
zoneNames	Zone names which are configured to the input contact that are reported.

7.4.15 DET_NetworkChangeDiagEvent

Purpose:

This diagnostic event indicates that there was a change in the network (broken links between devices). This event is only reported if network supervision is enabled (see [UG_PRAESENSA]).

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    BYTE               nrNetworkChanges;
    TNetworkChangeData networkChanges[];
} NetworkChangeDiagEvent;
```

Where TNetworkChangeData is defined as:

```
struct {
    STRING          localPortId;
    STRING          localSystemName;
    STRING          remotePortId;
    STRING          remoteSystemName;
} TNetworkChangeData
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_NetworkChangeDiagEvent.
nrNetworkChanges	The number of changes present in the network changes array element. Only this amount of array elements is transmitted.
networkChanges []	Array holding the network changes information. The actual length of the array is defined in the *nrNetworkChanges* element. The structure of each array element is described

below.

localPortId	The port ID of the local system.
localSystemName	The name of the local system as configured in the PRAESENSA system.
remotePortId	The port ID of the remote system.
remoteSystemName	The name of the remote system as configured in the PRAESENSA system.

7.4.16 DET_IncompatibleFirmware

Purpose:

This diagnostic event indicates that a device contains incompatible firmware and cannot be used in the PRAESENSA system.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    STRING             current;
    STRING             expected;
} OverheatFault;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_IncompatibleFirmware.
current	The current firmware in the device.
expected	The expected firmware the device should contain.

7.4.17 DET_Amp48VAFault

Purpose:

This diagnostic event indicates the loss of 48V A supply for the amplifier. Severity is high if DET_Amp48VBFault is also reported.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} Amp48VAFault;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_Amp48VAFault.
severity	Severity of the fault. LOW = 0, HIGH = 1.

Related events:

DET_Amp48VBFault

7.4.18 DET_Amp48VBFault

Purpose:

This diagnostic event indicates the loss 48V B supply. Severity is high if DET_Amp48VAFault is also reported.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} Amp48VAFault;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_Amp48VBFault.
severity	Severity of the fault. LOW = 0, HIGH = 1.

Related events:

DET_Amp48VAFault

7.4.19 DET_AmpChannelFault

Purpose:

This diagnostic event indicates a channel fault internally in the amplifier. If not used already, the spare channel takes over the functionality of the channel. Severity is high if the spare channel is already in use.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} Amp48VAFault;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_AmpChannelFault.
severity	Severity of the fault. LOW = 0, HIGH = 1.

7.4.20 DET_AmpShortCircuitLineAFault

Purpose:

This diagnostic event indicates for the amplifier channel the hardware short detection is triggered or the output voltage is too low due to a short on line A.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} AmpShortCircuitLineAFault;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_AmpShortCircuitLineAFault.
severity	Severity of the fault. LOW = 0, HIGH = 1.

7.4.21 DET_AmpShortCircuitLineBFault

Purpose:

This diagnostic event indicates for the amplifier channel the hardware short detection is triggered or the output voltage is too low due to a short on line B.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} AmpShortCircuitLineBFault;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_AmpShortCircuitLineBFault.
severity	Severity of the fault. LOW = 0, HIGH = 1.

7.4.22 DET_AmpAcc18VFault

Purpose:

This diagnostic event indicates failure of the amplifier lifeline power supply. The severity is not used.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} AmpAcc18VFault;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_AmpAcc18VFault.
severity	Severity of the fault. LOW = 0, HIGH = 1.

7.4.23 DET_AmpSpareInternalFault

Purpose:

This diagnostic event indicates an internal failure in the amplifier spare channel and can no longer be used. Severity is always high.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} AmpSpareInternalFault;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_AmpSpareInternalFault.
severity	Severity of the fault. LOW = 0, HIGH = 1.

7.4.24 DET_AmpChannelOverloadFault

Purpose:

This diagnostic event indicates for the amplifier channel an output overload has occurred.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} AmpChannelOverloadFault;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_AmpChannelOverloadFault.
severity	Severity of the fault. LOW = 0, HIGH = 1.

7.4.25 DET_EolFailureLineAFault

Purpose:

This diagnostic event indicates that the end-of-line device for the amplifier channel on line A is disconnected (the end-of-line pilot tone is not present).

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} EolFailureLineAFault;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_EolFailureLineAFault.
severity	Severity of the fault. LOW = 0, HIGH = 1.

7.4.26 DET_EolFailureLineBFault

Purpose:

This diagnostic event indicates that the end-of-line device for the amplifier channel on line B is disconnected (the end-of-line pilot tone is not present).

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} EolFailureLineBFault;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_EolFailureLineBFault.
severity	Severity of the fault. LOW = 0, HIGH = 1.

7.4.27 DET_GroundShortFault

Purpose:

This diagnostic event indicates that a ground fault is signaled by the amplifier hardware.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_GroundShortFault.

7.4.28 DET_OverheatFault

Purpose:

This diagnostic event indicates that amplifier hardware is overheated. All channels are disabled and severity is always high.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} OverheatFault;
```

Where:

header	Header of the event, where the *diagMessageType* element is equal to DET_OverheatFault
severity	Severity of the fault. LOW = 0, HIGH = 1.

7.4.29 DET_PowerMainsSupplyFault

Purpose:

This diagnostic event indicates the loss of mains power for a Multifunction Power Supply

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_PowerMainsSupplyFault.

7.4.30 DET_PowerBackupSupplyFault

Purpose:

This diagnostic event indicates the loss of backup power supply for a Multifunction Power Supply

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_PowerBackupSupplyFault.

7.4.31 DET_MainsAbsentPSU1Fault

Purpose:

This diagnostic event indicates absence of the output 1 mains power. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_MainsAbsentPSU1Fault.

7.4.32 DET_MainsAbsentPSU2Fault

Purpose:

This diagnostic event indicates absence of the output 2 mains power. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_MainsAbsentPSU2Fault.

7.4.33 DET_MainsAbsentPSU3Fault

Purpose:

This diagnostic event indicates absence of the output 3 mains power. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_MainsAbsentPSU3Fault.

7.4.34 DET_BackupAbsentPSU1Fault

Purpose:

This diagnostic event indicates absence of the output 1 12V DC backup power. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_BackupAbsentPSU1Fault.

7.4.35 DET_BackupAbsentPSU2Fault

Purpose:

This diagnostic event indicates absence of the output 2 12V DC backup power. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_BackupAbsentPSU2Fault.

7.4.36 DET_BackupAbsentPSU3Fault

Purpose:

This diagnostic event indicates absence of the output 3 12V DC backup power. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_BackupAbsentPSU3Fault.

7.4.37 DET_DcOut1PSU1Fault

Purpose:

This diagnostic event indicates a missing 48V DC output for output 1A. The numbers match the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_DcOut1PSU1Fault.

7.4.38 DET_DcOut2PSU1Fault

Purpose:

This diagnostic event indicates a missing 48V DC output for output 1B. The numbers match the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_DcOut2PSU1Fault.

7.4.39 DET_DcOut1PSU2Fault

Purpose:

This diagnostic event indicates a missing 48V DC output for output 2A. The numbers match the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_DcOut1PSU2Fault.

7.4.40 DET_DcOut2PSU2Fault

Purpose:

This diagnostic event indicates a missing 48V DC output for output 2B. The numbers match the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_DcOut2PSU2Fault.

7.4.41 DET_DcOut1PSU3Fault

Purpose:

This diagnostic event indicates a missing 48V DC output for output 3A. The numbers match the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_DcOut1PSU3Fault.

7.4.42 DET_DcOut2PSU3Fault

Purpose:

This diagnostic event indicates a missing 48V DC output for output 3B. The numbers match the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_DcOut2PSU3Fault.

7.4.43 DET_AudioLifelinePSU1Fault

Purpose:

This diagnostic event indicates a wiring problem in the ACC connector with the lifeline analog audio signal for output 1. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_AudioLifelinePSU1Fault.

7.4.44 DET_AudioLifelinePSU2Fault

Purpose:

This diagnostic event indicates a wiring problem in the ACC connector with the lifeline analog audio signal for output 2. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_AudioLifelinePSU2Fault.

7.4.45 DET_AudioLifelinePSU3Fault

Purpose:

This diagnostic event indicates a wiring problem in the ACC connector with the lifeline analog audio signal for output 3. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_AudioLifelinePSU3Fault.

7.4.46 DET_AccSupplyPSU1Fault

Purpose:

This diagnostic event indicates a missing 10 to 18V at the ACC connector for output 1. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_AccSupplyPSU1Fault.

7.4.47 DET_AccSupplyPSU2Fault

Purpose:

This diagnostic event indicates a missing 10 to 18V at the ACC connector for output 2. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_AccSupplyPSU2Fault.

7.4.48 DET_AccSupplyPSU3Fault

Purpose:

This diagnostic event indicates a missing 10 to 18V at the ACC connector for output 3. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_AccSupplyPSU3Fault.

7.4.49 DET_Fan1Fault

Purpose:

This diagnostic event indicates that fan 1 in the Multifunction Power Supply is broken.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_Fan1Fault.

7.4.50 DET_Fan2Fault

Purpose:

This diagnostic event indicates that fan 2 in the Multifunction Power Supply is broken.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_Fan2Fault.

7.4.51 DET_DcAux1Fault

Purpose:

This diagnostic event indicates the absence of 24V DC aux 1 voltage for the Multifunction Power Supply. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_DcAux1Fault.

7.4.52 DET_DcAux2Fault

Purpose:

This diagnostic event indicates the absence of 24V DC aux 2 voltage for the Multifunction Power Supply. The number matches the screening at the back-panel of the device.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_DcAux2Fault.

7.4.53 DET_BatteryShortFault

Purpose:

This diagnostic event indicates a short in the external battery for the Multifunction Power Supply.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_BatteryShortFault.

7.4.54 DET_BatteryRiFault

Purpose:

This diagnostic event indicates a Ri fault for the connected battery of the Multifunction Power Supply. Depending on the configured battery capacity in the PRAESENSA system a fault is reported.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_BatteryRiFault.

7.4.55 DET_BatteryOverheatFault

Purpose:

This diagnostic event indicates the temperature of the connected battery of the Multifunction Power Supply is not in correct working range

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_BatteryOverheatFault.

7.4.56 DET_BatteryFloatChargeFault

Purpose:

This diagnostic event indicates that the battery of the Multifunction Power Supply is most likely broken. The charger enters a float state when the State of Charge (SoC) is 100%. In this state a low charge current is expected just to component the self-discharge of the battery. When the charge current is very high the battery is probably broken and therefore the fault is reported. The charger is suspended for safety reasons.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_BatteryFloatChargeFault.

7.4.57 DET_MainsAbsentChargerFault

Purpose:

This diagnostic event indicates that the mains converter for the charger is defect which prevents charging the battery correctly.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_MainsAbsentChargerFault.

7.4.58 DET_PoESupplyFault

Purpose:

This diagnostic event indicates that a mismatch is detected the number of Power over Ethernet connections to the call station and the number of expected Power over Ethernet inputs configured in the PRAESENSA system.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the DIAGEVENTHEADER, wherein the *diagMessageType* is equal to DET_PoESupplyFault.

7.4.59 DET_PowerSupplyAFault

Purpose:

This diagnostic event indicates that the power supply input A level on the system controller is not within range. The fault is only reported if the power supply input is configured to be supervised in the PRAESENSA system.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_PowerSupplyAFault.

7.4.60 DET_PowerSupplyBFault

Purpose:

This diagnostic event indicates that the power supply input B level on the system controller is not within range. The fault is only reported if the power supply input is configured to be supervised in the PRAESENSA system.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_PowerSupplyBFault.

7.4.61 DET_ExternalPowerFault

Purpose:

This diagnostic event indicates that the PRAESENSA system is now in backup power mode. This event is only generated when backup power mode (in the system settings) has been configured to generate a fault event.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_ExternalPowerFault.

7.4.62 DET_ChargerSupplyVoltageTooLowFault

Purpose:

This diagnostic event indicates that the charger supply voltage is too low which prevents charging the battery correctly.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_ChargerSupplyVoltageTooLowFault.

7.4.63 DET_BatteryOvervoltageFault

Purpose:

This diagnostic event indicates that the internal charger is defect and is switched off for safety reasons.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_BatteryOvervoltageFault.

7.4.64 DET_BatteryUndervoltageFault

Purpose:

This diagnostic event indicates that there is an undervoltage situation when mains is absent. The battery is too empty to operate on.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_BatteryUndervoltageFault.

7.4.65 DET_MediaClockFault

Purpose:

This diagnostic event indicates there are one or more devices that failed to lock to PTP for a longer period of time.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_MediaClockFault.

7.4.66 DET_ChargerFault

Purpose:

This diagnostics event indicates an internal charger fault which prevents charging the battery correctly.

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_ChargerFault.

7.4.67 DET_Amp20VFault

Purpose:

This diagnostic event indicates the failure of the power convertor for the controller section of the amplifier.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} Amp20VFault;
```

Where:

Header	Header of the event, where the *diagMessageType* element is equal to DET_Amp20VFault.
Severity	Severity of the fault. LOW = 0, HIGH = 1.

Related events:

DET_AmpPsuFault

7.4.68 DET_AmpPsuFault

Purpose:

This diagnostic event indicates the failure of the power convertor for the audio section of the amplifier.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
    UINT               severity;
} AmpPsuFault;
```

Where:

Header	Header of the event, where the *diagMessageType* element is equal to DET_AmpPsuFault.
Severity	Severity of the fault. LOW = 0, HIGH = 1.

Related events:

DET_Amp20VFault

7.4.69 DET_NetworkLatencyFault

Purpose:

This diagnostic event indicates that an audio flow gets interrupted by network delay and network jitter.

Parameter structure:

```
struct {
    DIAGEVENTHEADER    header;
```

```
    UINT              severity;
} NetworkLatencyFault;
```

Where:

Header	Header of the event, where the *diagMessageType* element is equal to DET_NetworkLatencyFault.
Severity	Severity of the fault. LOW = 0, HIGH = 1.

7.4.70 DET_SynchronizationFault

Purpose:

This diagnostic event indicates that the synchronization between a backup controller and a master controller in a redundant system failed..

Parameter structure:

The Diagnostic Event structure contains only the information as described in the `DIAGEVENTHEADER`, wherein the *diagMessageType* is equal to DET_SynchronzationFault.

8. EVENT ORIGINATOR STRUCTURES

8.1 Introduction

Each diagnostic event that is sent contains originators to indicate who or which device has triggered the event. The group of originators described the structures for each kind of originator.

Each originator structure contains a fixed number of fields, which are described below.

```
Struct {
    DWORD              originatorType;
    UINT               length;
} ORIGINATORHEADER;
```

Where:

originatorType	The originator type indicator for the originator structure as defined §9.5. In the sections below the various diagnostic event types are described.
length	The total length of the originator information (including the *originatorType, Length* and the additional information as described for a specific diagnostic event type)

8.2 OIEOT_NoEventOriginator

Purpose:

This originator represents no or an unknown originator. There is no information available about the originator. During the creation of a diagnostic event message, only the *addEventOriginater* element is filled with an originator. All other originator elements of the structure are filled with this originator type.

Parameter structure:

The Originator structure contains only the information as described in the `ORIGINATORHEADER`, wherein the *originatorType* is equal to OIEOT_NoEventOriginator.

Note that since this originator type does not add additional information, the length parameter in the `ORIGINATORHEADER` only holds the length of the `ORIGINATORHEADER`.

8.3 OIEOT_UnitEventOriginator

Purpose:

This originator represents a unit connected to the PRAESENSA system.

Parameter structure:

```
struct {
    ORIGINATORHEADER    header;
    STRING              unitName;
} UnitOriginator;
```

Where:

header	The originator header, where the *originatorType* element is equal to OIEOT_UnitEventOriginator.
unitName	The name of the originator unit as configured in the PRAESENSA system configuration.

8.4 OIEOT_OpenInterfaceEventOriginator

Purpose:

This originator represents an open interface connection and its connection name.

Parameter structure:

```
struct {
    ORIGINATORHEADER    header;
    STRING              tcpIpDeviceName;
    DWORD               ipAddress;
    WORD                portNumber;
    STRING              userName;
} OpenInterfaceOriginator;
```

Where:

header	The originator header, where the *originatorType* element is

	equal to OIEOT_OpenInterfaceEventOriginator.
tcpIpDeviceName	The name of the TCP/IP device. Currently this name is not (yet) filled (empty string).
ipAddress	The IP address of the originator open interface connection. Note that this IP address is transmitted as DWORD (LSB ordering) and not as an IP-address. The ordering of the bytes is different. Only IPv4 is supported in the PRAESENSA system.
portNumber	The TCP-port number of the open interface connection.
userName	The login user name of the open interface connection.

8.5 OIEOT_ControlInputEventOriginator

Purpose:

This originator represents a binary control input, located on a unit.

Parameter structure:

```
struct {
    UnitOriginator      unitHeader;
    STRING              inputContactName;
} ControlInputOriginator;
```

Where:

unitHeader	The unit-originator header (See §8.3), where the *originatorType* element is equal to OIEOT_ControlInputEventOriginator.
inputContactName	The name of the input contact as configured in the PRAESENSA system configuration.

8.6 OIEOT_AudioOutputEventOriginator

Purpose:

This originator represents an audio output, located on a unit.

Parameter structure:

```
struct {
    UnitOriginator      unitHeader;
    STRING              audioOutputName;
} AudioOutputEventOriginator;
```

Where:

unitHeader	The unit-originator header (See §8.3), where the *originatorType* element is equal to OIEOT_AudioOutputEventOriginator.
audioOutputName	The name of the audio output as configured in the PRAESENSA system configuration.

8.7 OIEOT_AudioInputEventOriginator

Purpose:

This originator represents an audio input, located on a unit.

Parameter structure:

```
struct {
    UnitOriginator      unitHeader;
    STRING              audioInputName;
} AudioInputEventOriginator;
```

Where:

unitHeader	The unit-originator header (See §8.3), where the *originatorType* element is equal to OIEOT_AudioInputEventOriginator.
audioInputName	The name of the audio input as configured in the PRAESENSA system configuration.

8.8 OIEOT_UserEventOriginator

Purpose:

This originator represents user action performed on the system.

Parameter structure:

```
struct {
    UnitOriginator      unitHeader;
    STRING              userId;
} UserEventOriginator;
```

Where:

unitHeader	The unit-originator header (See §8.3), where the *originatorType* element is equal to OIEOT_UserEventOriginator.
userId	The user ID which is logged in.

8.9 OIEOT_NetworkEventOriginator

Purpose:

This originator represents network action performed on the system

Parameter structure:

```
struct {
    UnitOriginator      unitHeader;
    DWORD               ipAddress;
    DWORD               portNumber;
    STRING              userName;
} NetworkEventOriginator;
```

Where:

unitHeader	The unit-originator header (See §8.3), where the *originatorType* element is equal to OIEOT_NetworkEventOriginator.
ipAddress	The IP address of the network connection. Note that this IP address is transmitted as DWORD (LSB ordering) and not as an IP-address. The ordering of the bytes is different. Only IPv4 is supported in the PRAESENSA system.
portNumber	The TCP-port number of the connection
userName	The user name of the originator network connection

9. OIP CONSTANT VALUES

In this document some constants are used. In this section all constants will be connected to their values and to their reference type. Note that these constants are only used within the Open Interface protocol and not in the diagnostic events and event originators.

9.1 Protocol Constants

Related to the protocol, there are several constants. This section summary describes the constants to be used to handle the protocol.

Constant Meaning	Value
Port Number for the unsecure connection	9401
Port Number for the secure connection	9403
Transmit timeout for transmission heartbeat message	5 seconds
Check timeout to verify whether a message is received (reset after each message reception)	15 seconds
Maximum command response time	10 seconds
Minimum message size (message-type + length)	8 bytes
Maximum message size	128 Kbytes
Maximum string size	64 Kbytes

9.2 General Constants

9.2.1 TOIEventId

The event Identification represents a diagnostic event as generated by the PRAESENSA system. The type is mapped upon a UINT basic type as described in §3.4.3.1. In case the command results in an error, a special value is returned, described in the table below.

Constant name	Value
OI_UNDEFINED_EVENTID	0xFFFFFFFF

9.2.2 TOICallId

The call Identification represents a running call in the PRAESENSA system and is generated by the PRAESENSA system. The type is mapped upon a UINT basic type as described in §3.4.3.1. In case the command result in an error, a special value is returned, described in the table below.

Constant name	Value
OI_UNDEFINED_CALLID	0xFFFFFFFF

9.2.3 TOIAlarmType

The system wide alarms as used within the PRAESENSA system are represented by the alarm-type. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below.

Constant name	Value
OIAT_EVAC	0x00000000
OIAT_FAULT	0x00000001

9.2.4 TOIAlarmState

The alarm states as used within the PRAESENSA system are represented by the Alarm-state type. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below.

Constant name	Value
OIAS_ACTIVE	0x00000000
OIAS_ACKNOWLEDGED	0x00000001
OIAS_INACTIVE	0x00000002

9.2.5 TOIResourceState

The resource states as used within the PRAESENSA system are represented by the resource-state type. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below.

Constant name	value
OIRS_FREE	0x00000000
OIRS_INUSE	0x00000001

9.2.6 TOIResourceFaultState

The resource fault states as used within the PRAESENSA system are represented by the resource fault state type. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below.

Constant name	value
OIRS_OK Indicates that no fault is present for the resource that affects the audio distribution of that resource.	0x00000000
OIRS_FAULT Indicates that a fault is present for the resource that affects the audio distribution of that resource.	0x00000001

9.2.7 TOICallState

The call states as used within the PRAESENSA system are represented by the Call-state type. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below.

Constant name	value
OICS_START The call is in preparation.	0x00000000
OICS_STARTCHIME The call is processing the start-chime.	0x00000001
OICS_MESSAGES The call is processing the prerecorded messages (including repeats).	0x00000002
OICS_LIVESPEECH The call is in the live speech state. The audio input passed during the start of the call is active.	0x00000003
OICS_ENDCHIME The call is processing the end-chime.	0x00000004
OICS_END Final state of the call. The associated call identification is not valid any more.	0x00000005
OICS_ABORT Final state of the call. The associated call identification is not valid any more.	0x00000006
OICS_IDLE The call is identified, but the processing needs to be started (no resources are associated with the call yet).	0x00000007
OICS_REPLAY Indicates that the mentioned call is waiting for available resources or/and replaying a previously recorded call.	0x00000008

9.2.8 TOICallStopReason

The reason for an aborted call to stop as used within the PRAESENSA system is represented by the stopReason type. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below.

Constant name	value
OICSR_ORIGINATOR The call was stopped by the originator.	0x00000000
OICSR_RESOURCE_LOST The call was stopped due to lost or overruled resources.	0x00000001

Constant name	value
OICSR_SYSTEM The call was stopped by the system.	0x00000002
OICSR_STOPCOMMAND The call was stopped by a stop command.	0x00000003
OICSR_UNKNOWN The call was stopped by an undefined reason.	0x00000004

9.2.9 TOICallResetReason

The reason for a call to reset as used within the PRAESENSA system is represented by the resetReason type. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below.

Constant name	value
OICRR_RESOURCE_LOST The call was reset due to lost or overruled resources.	0x00000000
OICRR_SYSTEM The call was reset by the system.	0x00000001
OICRR_UNKNOWN The call was reset by an undefined reason.	0x00000002

9.2.10 TOIActionType

The action type describes the action performed on the specified diagnostic event. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below.

Constant name	Value
OIACT_NEW The specified diagnostic event is added to the event storage in the PRAESENSA system.	0x00000000
OIACT_ACKNOWLEDGED The specified diagnostic (fault) event is acknowledged.	0x00000001
OIACT_RESOLVED The specified diagnostic (fault) event is resolved.	0x00000002
OIACT_RESET The specified diagnostic (fault) event is reset.	0x00000003
OIACT_UPDATED The specified diagnostic (fault) event is updated. This means that additional information is added to the diagnostic event (e.g. The number of WLS2-Boards with failures is extending).	0x00000004
OIACT_REMOVED The specified diagnostic event is removed from the event storage in the PRAESENSA system.	0x00000005
OIACT_EXISTING The specified diagnostic event is already present in the storage. This action type is passed for each diagnostic event already in the storage after subscription for the events (See §4.35).	0x00000006
OIACT_EXISTING_LAST The specified diagnostic event is already present in the storage and it is the last present event sent, or there are actually no fault events present in the storage, in which case the specified diagnostic event is of type DET_NoFaults.	0x00000007

9.2.11 TOICallOutputHandling

Describes how calls behave on routing availability. The type is mapped upon a UINT basic type as described in §3.4.3.1.

Constant name	Value
OICOH_PARTIAL Partial calls are calls that proceed even in case not all required zones are available.	0x00000000
OICOH_NON_PARTIAL Non-partial calls are calls that require the entire routing to be available at the start of the call and during the call. When during the call a part of the routing becomes unavailable, the call is aborted.	0x00000001
OICOH_STACKED Stacked calls are calls that extend partial calls with replays to previously unavailable zones.	0x00000002

9.2.12 TOICallStackingMode

Describes when recorded calls replay. A stacked call or a stacked call waits for each zone to become available for replay. The type is mapped upon a UINT basic type as described in §3.4.3.1

Constant name	Value
OICSM_WAIT_FOR_ALL Wait with replay for all zones to become available	0x00000000
OICSM_WAIT_FOR_EACH Start a replay for each zone to become available	0x00000001

9.2.13 TOICallTiming

Indicates the way the call must be handled. The type is mapped upon a UINT basic type as described in §3.4.3.1

Constant name	Value
OICTM_IMMEDIATE Broadcast to the selected zones and zone groups when the call is started.	0x00000000
OICTM_TIME_SHIFTED Broadcast to the selected zones and zone groups when the original call is finished to prevent audio feedback during live speech.	0x00000001
OICTM_MONITORED Broadcast when the call is not cancelled within 2 seconds after the monitoring phase has finished.	0x00000002

9.2.14 TOICallStackingTimeout

Defines the limit of time for stacked call broadcasting. The type is mapped upon a UINT basic type as described in §3.4.3.1

Constant name	Value
OICST_INFINITE Wait infinitely for zones to become available for broadcasting.	0xFFFFFFFF

9.2.15 TOIVirtualControlInputDeactivation

Defines the behavior of the running action when deactivating a virtual control input. The type is mapped upon a UINT basic type as described in §3.4.3.1

Constant name	Value
OIVCI_STOP	0x00000000

Constant name	Value
Stop the running action gracefully.	
OIVCI_ABORT Abort the running action immediately.	0x00000001

9.2.16 TOIVirtualControlInputState

Defines the values returned when the state of virtual control inputs change. The type is mapped upon a UINT basic type as described in §3.4.3.1

Constant name	Value
OIVCIS_INACTIVE Indicates that the virtual control input is in the inactive state (associated action not running).	0x00000000
OIVCIS_ACTIVE Indicates that the virtual control input is in the active state (associated action running).	0x00000001

9.3 Diagnostic Constant values

9.3.1 TDiagEventState

The diagnostic event states as used within the PRAESENSA system are represented by the Diagnostic-Event-state type. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below.

Constant name	value
DES_NEW	0x00000000
DES_ACKNOWLEDGED	0x00000001
DES_RESOLVED	0x00000002
DES_RESET	0x00000003

9.3.2 TDiagEventGroup

The diagnostic event groups as used within the PRAESENSA system are represented by the Diagnostic-event-group type. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below.

Constant name	value
DEG_CallEventGroup	0x00000000
DEG_GeneralEventGroup	0x00000001
DEG_FaultEventGroup	0x00000002

9.3.3 TDiagEventType

The diagnostic event types as used within the PRAESENSA system are represented by the diagnostic-event type. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below. In the event that a value of TDiagEventType is received that is not in this table, a new version of PRAESENSA is probably installed on the system controller.

Constant name	Value
DET_CallChangeResourceV2	0x00467105
DET_CallEndV2	0x00467106
DET_CallStartV2	0x00467107
DET_CallTimeoutV2	0x00467108
DET_CallRestart	0x00467109
DET_CallReset	0x0046710B
DET_EvacAcknowledge	0x00467204
DET_EvacReset	0x00467205
DET_EvacSet	0x00467206
DET_SCStartup	0x00467209
DET_OpenInterfaceConnect	0x0046720A
DET_OpenInterfaceDisconnect	0x0046720B
DET_UnitConnect	0x0046720E
DET_CallLoggingSuspended	0x0046720F

Constant name	Value
DET_CallLoggingResumed	0x00467210
DET_UserLogIn	0x00467213
DET_UserLogOut	0x00467214
DET_UserLogInFailed	0x00467215
DET_OpenInterfaceConnectFailed	0x00467216
DET_BackupPowerModeStart	0x00467217
DET_BackupPowerModeEnd	0x00467218
DET_ConfigurationRestored	0x00467219
DET_AudioPathSupervision	0x00467308
DET_CallStationExtension	0x0046730A
DET_ConfigurationFile	0x0046730D
DET_ConfigurationVersion	0x0046730E
DET_IllegalConfiguration	0x00467312
DET_MicrophoneSupervision	0x00467315
DET_PrerecordedMessagesNames	0x00467319
DET_SystemInputContact	0x0046731B
DET_UnitMissing	0x0046731C
DET_UserInjectedFault	0x00467320
DET_NoFaults	0x00467334
DET_ZoneLineFault	0x00467335
DET_PrerecordedMessagesCorrupt	0x00467337
DET_NetworkChangeDiagEvent	0x00467339
DET_DemoteToBackup	0x0046733A
DET_Amp48VAFault	0x00467400
DET_Amp48VBFault	0x00467401
DET_AmpChannelFault	0x00467402
DET_AmpShortCircuitLineAFault	0x00467405
DET_AmpShortCircuitLineBFault	0x00467406
DET_EolFailureLineAFault	0x00467407
DET_EolFailureLineBFault	0x00467408
DET_Fan1Fault	0x00467409
DET_Fan2Fault	0x0046740a
DET_GroundShortFault	0x0046740b
DET_OverheatFault	0x0046740c
DET_UnitResetFault	0x0046740d
DET_IncompatibleFirmware	0x0046740e
DET_PoESupplyFault	0x0046740f
DET_PowerSupplyAFault	0x00467410
DET_PowerSupplyBFault	0x00467411
DET_ExternalPowerFault	0x00467412
DET_DcAux1Fault	0x00467413
DET_DcAux2Fault	0x00467414
DET_BatteryShortFault	0x00467415
DET_BatteryRiFault	0x00467416
DET_BatteryOverheatFault	0x00467417
DET_BatteryFloatChargeFault	0x00467418
DET_MainsAbsentChargerFault	0x00467419
DET_MainsAbsentPSU1Fault	0x0046741a
DET_BackupAbsentPSU1Fault	0x0046741b
DET_DcOut1PSU1Fault	0x0046741c
DET_DcOut2PSU1Fault	0x0046741d
DET_AudioLifelinePSU1Fault	0x0046741e
DET_AccSupplyPSU1Fault	0x0046741f
DET_MainsAbsentPSU2Fault	0x00467420
DET_BackupAbsentPSU2Fault	0x00467421
DET_DcOut1PSU2Fault	0x00467422
DET_DcOut2PSU2Fault	0x00467423
DET_AudioLifelinePSU2Fault	0x00467424
DET_AccSupplyPSU2Fault	0x00467425

Constant name	Value
DET_MainsAbsentPSU3Fault	0x00467426
DET_BackupAbsentPSU3Fault	0x00467427
DET_DcOut1PSU3Fault	0x00467428
DET_DcOut2PSU3Fault	0x00467429
DET_AudioLifelinePSU3Fault	0x0046742a
DET_AccSupplyPSU3Fault	0x0046742b
DET_AmpAcc18VFault	0x0046742c
DET_AmpSpareInternalFault	0x0046742d
DET_AmpChannelOverloadFault	0x0046742e
DET_PowerMainsSupplyFault	0x0046742f
DET_PowerBackupSupplyFault	0x00467430
DET_ChargerSupplyVoltageTooLowFault	0x00467431
DET_BatteryOvervoltageFault	0x00467432
DET_BatteryUndervoltageFault	0x00467433
DET_MediaClockFault	0x00467434
DET_ChargerFault	0x00467435
DET_Amp20VFault	0x00467436
DET_AmpPsuFault	0x00467437
DET_NetworkLatencyFault	0x00467438
DET_SynchronizationFault	0x00467439

9.4 Message Types

The message types (command, response and notification messages) as used within the PRAESENSA system are represented by the Message type. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below.

Constant name	Value
MESSAGETYPE_OIP_Login	0x00447002
MESSAGETYPE_OIP_StopCall	0x00447004
MESSAGETYPE_OIP_AbortCall	0x00447005
MESSAGETYPE_OIP_AddToCall	0x00447006
MESSAGETYPE_OIP_RemoveFromCall	0x00447007
MESSAGETYPE_OIP_AckAllFaults	0x00447008
MESSAGETYPE_OIP_ResetAllFaults	0x00447009
MESSAGETYPE_OIP_AckEvacAlarm	0x0044700a
MESSAGETYPE_OIP_SetSubscriptionAlarm	0x0044700d
MESSAGETYPE_OIP_SetSubscriptionResources	0x0044700e
MESSAGETYPE_OIP_GetNcoVersion	0x0044700f
MESSAGETYPE_OIP_IncrementBgmVolume	0x00447010
MESSAGETYPE_OIP_DecrementBgmVolume	0x00447011
MESSAGETYPE_OIP_SetBgmVolume	0x00447012
MESSAGETYPE_OIP_AddBgmRouting	0x00447013
MESSAGETYPE_OIP_RemoveBgmRouting	0x00447014
MESSAGETYPE_OIP_SetBgmRouting	0x00447015
MESSAGETYPE_OIP_SetSubscriptionBgmRouting	0x00447016
MESSAGETYPE_OIP_ReportFault	0x00447017
MESSAGETYPE_OIP_ResolveFault	0x00447018
MESSAGETYPE_OIP_AckFault	0x00447019
MESSAGETYPE_OIP_ResetFault	0x0044701a
MESSAGETYPE_OIP_SetSubscriptionEvents	0x0044701b
MESSAGETYPE_OIP_Response	0x0044701c
MESSAGETYPE_OIP_ResponseCallId	0x0044701d
MESSAGETYPE_OIP_ResponseGetNcoVersion	0x0044701e
MESSAGETYPE_OIP_ResponseReportFault	0x0044701f
MESSAGETYPE_OIP_ResponseProtocolError	0x00447020
MESSAGETYPE_OIP_NotifyAlarm	0x00447022
MESSAGETYPE_OIP_NotifyCall	0x00447023
MESSAGETYPE_OIP_NotifyResources	0x00447024
MESSAGETYPE_OIP_NotifyBgmRouting	0x00447025

Constant name	Value
MESSAGETYPE_OIP_NotifyDiagEvent	0x00447026
MESSAGETYPE_OIP_KeepAlive	0x00447027
MESSAGETYPE_OIP_StartCreatedCall	0x00447029
MESSAGETYPE_OIP_GetZoneNames	0x0044702a
MESSAGETYPE_OIP_GetZoneGroupNames	0x0044702b
MESSAGETYPE_OIP_GetMessageNames	0x0044702c
MESSAGETYPE_OIP_GetChimeNames	0x0044702d
MESSAGETYPE_OIP_GetAudioInputNames	0x0044702e
MESSAGETYPE_OIP_GetBgmChannelNames	0x0044702f
MESSAGETYPE_OIP_GetConfigId	0x00447030
MESSAGETYPE_OIP_SetSubscriptionBgmVolume	0x00447031
MESSAGETYPE_OIP_ResponseConfigId	0x00447032
MESSAGETYPE_OIP_ResponseNames	0x00447033
MESSAGETYPE_OIP_NotifyBgmVolume	0x00447034
MESSAGETYPE_OIP_IncrementBgmChannelVolume	0x00447035
MESSAGETYPE_OIP_DecrementBgmChannelVolume	0x00447036
MESSAGETYPE_OIP_CancelAll	0x00447038
MESSAGETYPE_OIP_CancelLast	0x00447039
MESSAGETYPE_OIP_ToggleBgmRouting	0x0044703A
MESSAGETYPE_OIP_ResetEvacAlarmEx	0x0044703B
MESSAGETYPE_OIP_SetSubscriptionResourceFaultState	0x0044703C
MESSAGETYPE_OIP_NotifyResourceFaultState	0x0044703D
MESSAGETYPE_OIP_ActivateVirtualControlInput	0x0044703F
MESSAGETYPE_OIP_DeactivateVirtualControlInput	0x00447040
MESSAGETYPE_OIP_SetSubscriptionUnitCount	0x00447041
MESSAGETYPE_OIP_SetSubscriptionVirtualControlInputs	0x00447042
MESSAGETYPE_OIP_GetVirtualControlInputNames	0x00447043
MESSAGETYPE_OIP_NotifyUnitCount	0x00447044
MESSAGETYPE_OIP_NotifyVirtualControlInputState	0x00447045
MESSAGETYPE_OIP_GetConfiguredUnits	0x00447046
MESSAGETYPE_OIP_GetConnectedUnits	0x00447047
MESSAGETYPE_OIP_ResponseUnits	0x00447048
MESSAGETYPE_OIP_CreateCallEx3	0x00447049
MESSAGETYPE_OIP_GetProtocolVersion	0x0044704A
MESSAGETYPE_OIP_ResponseGetProtocolVersion	0x0044704B

9.5 Event originator Message Types

The originator message types as used within the PRAESENSA system are represented by the originator-message type. The type is mapped upon a UINT basic type as described in §3.4.3.1. The valid values used within this type are described in the table below.

Constant name	value
OIEOT_NoEventOriginator	0x00477002
OIEOT_UnitEventOriginator	0x00477003
OIEOT_OpenInterfaceEventOriginator	0x00477004
OIEOT_ControlInputEventOriginator	0x00477005
OIEOT_AudioOutputEventOriginator	0x00477006
OIEOT_AudioInputEventOriginator	0x00477007
OIEOT_UserEventOriginator	0x00477009
OIEOT_NetworkEventOriginator	0x0047700A

10. Error Codes

Responses returned upon a remote function request contain an error field. In this section an overview is given of the possible errors and their hexadecimal values.

Remote Function Services Error code	Value
ERROR_OK The command message is executed successfully.	0X00000000
ERROR_INVALID_PARAMETERS If one of the parameters is wrong, this error code is returned.	0X0044E000
ERROR_INTERNAL The PRAESENSA system cannot fulfill the command due to an internal error.	0X0044E001
ERROR_INVALID_MESSAGE_LENGTH The overall message length of the data is too small (below 8 bytes) or too large (above 128 Kbytes).	0X0044E002
ERROR_UNEXPECTED_COMMAND_TYPE The message cannot be used, since the message-type is not a known command by the PRAESENSA system.	0X0044E003
ERROR_TOO_MUCH_UNMARSHAL_DATA Parsing of the message was done, but conform the length information, there is still data left in the message. The message length does not match the content. The message is not accepted.	0X0044E004
ERROR_MUST_LOGIN_FIRST Command received before the user is logged in.	0X0044E005
ERROR_INVALID_MESSAGETYPE The message cannot be used, since the message-type is not known by the PRAESENSA system.	0x0044E006
ERROR_STRING_TOO_LONG The length of a string is too long (above 64 Kbytes). Related to a string in a message, but message length within boundaries.	0X0044E007
ERROR_UNEXPECTED_END Parsing of the message goes beyond the end of the message. Sum of the element lengths greater than the message length.	0X0044E008
ERROR_CALL_NO_LONGER_EXISTS The given callId belongs to a call that, even though it was created (but not yet started via the Open Interface), no longer exists. Successive use of this callId will result in ERROR_INVALID_PARAMETERS.	0X0044E009

PART 2 - OPEN INTERFACE LIBRARY

This part 2 of the Open Interface programming instructions describes the Open Interface Library of the PRAESENSA system.

11. INTRODUCTION

11.1 Purpose

The purpose of part 2 of this document is to describe the usage of the PRAESENSA Open Interface protocol version V10.0 based on a C# and .NET Framework implementation.

11.2 Scope

Part 2 of this document describes the use of the open interface in combination with C# and .NET Framework. To understand this document, knowledge is expected on the following issues:

- The C# programming language and its development environment.
- The PRAESENSA system and its installation.

This document is intended for users, who want to use the PRAESENSA Open Interface into their application.

The user of this document cannot derive any rights from this document regarding the programming interface. Extensions and improvements on the Open Interface can be implemented when new versions of PRAESENSA are introduced.

11.3 Definitions, Acronyms and Abbreviations

DNS	Domain Name System
OI	Open Interface
PA	Public address

11.4 References

The reference that must be used for this document is:

- The PRAESENSA configuration manual.
- Part 1 of this document > - Open Interface protocol.

11.5 Summary

Chapter 12, "**APPLICATION CONTROL OVERVIEW**" describes the principles of controlling the PRAESENSA application using the Open Interface.

Chapter 13, "**INTERFACE DEFINITION**" describes the constants, methods and events present on the Open Interface.

Chapter 14 provides a Visual Basic example where most of the functions described are used.

12. APPLICATION CONTROL OVERVIEW

12.1 Principle

The PRAESENSA system is a public address system to perform calls to various areas in a building. Each area, called a zone, is reached by means of one or more amplifiers and is given a name. Multiple areas (zones) can be groups into a zone-group.

Special calls are identified within the PRAESENSA system as emergency calls. These calls can be triggered by e.g. a fire alarm system. These emergency alarm calls contain mostly repeated pre-recorded messages and put the system into a special (emergency) state. The system remains in this state until an operator acknowledges and resets the emergency state. Beside the normal calls, the PRAESENSA system monitors itself and reports any faults found in the system.

To perform a PA call, the following main information needs to be passed to the PRAESENSA system:

- The routing, a collection of zone names and/or zone Group names.
- The priority of the call.
- [Optional] A starting chime name to trigger the listeners that a call is starting.
- [Optional] A set of pre-recorded messages to be played.
- [Optional] A live speech section, where the operator can do his/her spoken message. The microphone is identified by means of the name of an audio input.
- [Optional] An ending chime to notify the termination of the call.

Note that most of the inputs are optional, but at least one of the optional elements must be defined to trigger a valid call.

Upon subscription for diagnostic events, the system first sent all available events as present in the PRAESENSA system, followed by the new and updated events.

12.1.1 Limitations

Some of the type definitions described in section 13.2 are currently not supported and should not be used.

- Only partial calls are supported. Setting the output handling to anything other than OICOH_PARTIAL when creating a call will result in a parameter error.
- Only immediate calls are supported. Setting the call timing to anything other than OICTM_IMMEDIATE when creating a call will result in a parameter error.
- Call stacking is not (yet) supported.

12.2 Referencing the library

Before the library can be used within your C# application, you need to add a reference to the library. This can be done in the Visual Studio development environment, using the Project→Add References… menu entry.

In the Add Reference dialog, select Assemblies, Extensions and then select `OpenInterfaceNetLibrary` (if present). When not present, select the browse button and select the file `OpenInterfaceNetLibrary.dll` in the installed location.

NOTE: The `OpenInterfaceNetLibrary` DLL targets .NET Framework 4.5.

12.3 Library usage in C#

After the reference addition, C# knows the methods and events of the PRAESENSA open interface. The application can call the method on the library and the PRAESENSA system will send events to the application.

Before the library can be used the following directive should be added.

```
using Bosch.PRAESENSA.OpenInterface;
```

To use the library, a client object needs to be constructed.

```
OpenInterfaceNetClient client = new OpenInterfaceNetClient();
```

12.4 Catching errors

Problems detected during a call to the PRAESENSA system will be reported by means of `TOIErrorCode` return codes. If no errors are reported, the functions return `TOIErrorCode.OIERROR_OK`.

The following code sample shows the use of a `TOIErrorCode` code to catch failures.

```
string ip = "192.168.53.100";
string username = "user";
string password = "password";

TOIErrorCode ec = client.Connect(ip, username, password);
if (ec == TOIErrorCode.OIERROR_OK)
{
    // Continue as usual
}
else
{
    // Handle and/or report error
}
```

Explanation about error codes can be found in section 13.2.2.

13. INTERFACE DEFINITION

13.1 Introduction

This chapter describes the various remote methods available on the PRAESENSA open interface.

13.1.1 Method and Event explanation

The descriptions of the methods and the events contain a brief function explanation and the declaration of the method/event. Further the following items can be present, depending on the content of the method/event:

- **Parameters:**
  A description of the parameters to be passed to the interface method.
- **Return value:**
  A description of the return value returned by the interface method.
- **Related event types:**
  A list of types, whereby the described function is operational. When called for other type the Open Interface shall generate an exception.
- **Error codes:**
  A list of error codes, which can be thrown during the execution of the interface method. See §13.2.2 for a description of the error codes.

13.2 Enumeration type definitions

Within the library various enumeration types and constants are defined to prevent the use of magic (non-explaining) numbers.

13.2.1 OpenInterfaceConstants

UNDEFINED_CALLID = UINT_MAX:
: Standard indication for a call identifier to which no call is associated.

13.2.2 TIOErrorCode

The `TIOErrorCode` type represents the error values, which can be returned by the Open Interface functions. The error values have the following meaning:

OIERROR_OK:
: The Open Interface function has successfully executed.

OIERROR_ALREADY_LOGGED_IN:
: The Open Interface is already logged in to a PRAESENSA system. Disconnect from the PRAESENSA system and try again.

OIERROR_BAD_CREDENTIALS:
: The Open Interface could not complete the connection, because the username and/or the password is incorrect.

OIERROR_INTERNAL_ERROR:
: The PRAESENSA system detected an internal error during the processing of the command. Check the PRAESENSA System configuration. If persistent, contact PRAESENSA customer services.

OIERROR_INVALID_PARAMETERS:
: Indications that one or more parameters passed to the method do not match the configured names present in the connected PRAESENSA system or that a passed value is out of range. Strings are considered to be invalid when their length exceeds 15000 characters.

OIERROR_NO_CONNECTION:
: The Open Interface connection to the PRAESENSA system is not established.

OIERROR_NOT_REGISTERED:
A command was received via the Open Interface before the user is logged in. Call the `Connect` method first and try again.

OIERROR_UNABLE_TO_MAKE_CONNECTION:
The Open Interface could not complete the connection, due to problems of the link to the PRAESENSA system.

OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER:
The PRAESENSA system controller does not support the function called. In general this means that there is a protocol version mismatch. The added functions to the open interface between the two versions cannot be executed.

OIERROR_CALL_NO_LONGER_EXISTS:

The given callId belongs to a call that, even though it was created (but not yet started via the Open Interface), no longer exists. Successive use of this callId will result in OIERROR_INVALID_PARAMETERS.

OIERROR_NO_RESPONSE_RECEIVED:

No answer was received after 5000 ms. Check the connection and try again.

OIERROR_PROTOCOL_ERROR_UNEXPECTED_END:

Parsing of the response goes beyond the end of the message. Sum of the element lengths is greater than the message length. In general this means that there is a protocol version mismatch.

13.2.3 TOIAlarmType

The `TOIAlarmType` type defines the type of alarm

OIAT_EVAC:
Indicates that the alarm is of type evac.

OIAS_FAULT:
Indicates that the alarm is of type fault.

13.2.4 TOIAlarmState

The `TOIAlarmState` type defines the values returned when an alarm occurs.

OIAS_ACTIVE:
Indicates that the alarm state is active.

OIAS_ACKNOWLEDGED:
Indicates that an alarm situation is present and that the alarm state has been acknowledged

OIAS_INACTIVE:
Indicates that no alarm situation is present.

13.2.5 TOICallPriority

The `TOICallPriority` type gives the various sub-ranges for the call priority. The actual value of the call priority depends whether the call is a background music call, a normal call or an emergency call. For each sub-range the minimum and maximum value is given as constant. Calls with higher priority proceeds / overrules calls with lower priority.

OI_MIN_PRIORITY_BGM = 0:
Represents the minimum background music priority value.

OI_MAX_PRIORITY_BGM = 31:
Represents the maximum background music priority value.

OI_MIN_PRIORITY_CALL = 32:
Represents the minimum normal call priority value.

OI_MAX_PRIORITY_CALL = 223:
Represents the maximum normal call priority value.

OI_MIN_PRIORITY_ALARM = 224:
Represents the minimum emergency call priority value.

OI_MAX_PRIORITY_ALARM = 255:
Represents the maximum emergency call priority value.

13.2.6 TOICallState

The `TOICallState` type defines the values returned when the state of a running call changes. Together with the call states, a callId is passed, which identifies the associated call.

OICS_START:
Indicates that the mentioned call has started.

OICS_STARTCHIME:
Indicates that the mentioned call is busy with its starting chime.

OICS_MESSAGES:
Indicates that the mentioned call is busy playing the specified messages for the call.

OICS_LIVESPEECH:
Indicates that the mentioned call is in the live speech phase. The operator of the call can now speak.

OICS_ENDCHIME:
Indicates that the mentioned call is busy with its ending chime.

OICS_END:
Indicates that the mentioned call has ended. The callId is no longer valid after this notification.

OICS_ABORT:
Indicates that the mentioned call has been aborted by either the user or another call started with a higher priority. The callId is after this notification no longer valid.

OICS_IDLE:
Indicates that the mentioned call is known by the system, but not (yet) operational. Note that a call can become idle when the call loses all his resources (BGM call).

OICS_REPLAY:

Indicates that the mentioned call is waiting for available resources and/or replaying the recorded call

13.2.7 TOICallStopReason

The `TOICallStopReason` type defines possible stop and abort reasons for a stopped call. This type is returned as a property by the StopReason getter supplied in the DET_CallEndDiagEventV2 event type The getter Aborted indicates whether the call is stopped or an aborted call. When a call ends naturally, the value will be OICSR_ORIGINATOR.

OICSR_ORIGINATOR:
Indicates that the call was ended by the originator.

OICSR_RESOURCE_LOST:

Indicates that resource(s) used by the ended call were lost or overruled.

OICSR_SYSTEM:

Indicates that the ended call was stopped by the system.

OICSR_STOPCOMMAND:

Indicates that the ended call was stopped by a stop command.

OICSR_UNKNOWN:

Indicates that the aborted call was stopped for an undefined reason.

13.2.8 TOICallResetReason

The `TOICallResetReason` type defines possible reasons for a reset call. This type is returned as a property by the ResetReason getter supplied in the DET_CallResetDiagEvent event type

OICRR_RESOURCE_LOST:

Indicates that resource(s) used by the reset call were lost or overruled.

OICRR_SYSTEM:

Indicates that the call was reset by the system.

OICRR_UNKNOWN:

Indicates that the call was reset for an undefined reason.

13.2.9 TOIResourceState

The `TOIResourceState` type defines the values returned when the state of resources (read zone groups, zones or control outputs) present in the PRAESENSA system changes.

OIRS_FREE:
Indicates that the resource is free to be used in a call.

OIRS_INUSE:
Indicates that the resource is in use by a running call.

13.2.10 TOIResourceFaultState

The `TOIResourceFaultState` type defines the values returned for the fault state when the state of resources (read zone groups or zones) present in the PRAESENSA system changes.

OIRS_OK
Indicates that no fault is present for the resource that affects the audio distribution of that resource.

OIRS_FAULT:
Indicates that a fault is present for the resource that affects the audio distribution of that resource.

13.2.11 TOIVirtualControlInputDeactivation

The `TOIVirtualControlInputDeactivation` type defines the behavior of the running action when deactivating a virtual control input.

OIVCI_STOP:
Stop the running action gracefully.

OIVCI_ABORT:
Abort the running action immediately.

13.2.12 TOIVirtualControlInputState

The `TOIVirtualControlInputState` type defines the values returned when the state of virtual control inputs change.

OIVCIS_ACTIVE:
Indicates that the control input is in the active state (associated action running). During the time the action is aborting (gracefully) the control input remains in the active state until the action has completed.

OIVCIS_INACTIVE:
Indicates that the control input is in the inactive state (associated action not running).

13.2.13 TOIDiagEventType

The `TOIDiagEventType` type defines the type of event passed through the open interface. It identifies the events and the associated members for that event.

Note that newer versions of PRAESENSA will most likely send newer (other) types. The application should check and report this so it can be adapted to the new situation.

13.2.13.1 Call Diagnostic Event-Group Event-types

OIDET_CallStartV2:
Indicates that the diagnostic event represents the start of a call in the PRAESENSA system.

OIDET_CallEndV2:
Indicates that the diagnostic event represents the end (or abort) of a call in the PRAESENSA system.

OIDET_CallChangeResourceV2:
Indicates that the diagnostic event represents a change in routing of a running call. The diagnostic event indicates whether zones are added to the routing or removed from the routing.

OIDET_CallTimeoutV2:
This diagnostic event indicates that a stacked call has reached its time-out point and implies that the call has been unable to reach all required zones. The diagnostic event provides the unreached zones.

OIDET_CALLRESTART:

Indicates that the diagnostic events represents the restart of a call in the PRAESENSA system.

OIDET_CALLRESET

Indicates that the diagnostic events represents a reset of a call in the PRAESENSA system. A reset indicates that the call will be restarted.

13.2.13.2 General Diagnostic Event-Group Event-types

OIDET_EvacAcknowledge:
Indicates that the diagnostic event represents that the system emergency state is acknowledged.

OIDET_EvacReset:
Indicates that the diagnostic event represents that the system emergency state is reset.

OIDET_EvacSet:
Indicates that the diagnostic event represents that the system emergency state is set (activated).

OIDET_UnitConnect:
Indicates that the diagnostic event represents that a unit has connected to or disconnected from the PRAESENSA system.

OIDET_SCStartup:
Indicates that the diagnostic event represents that the PRAESENSA system has started.

OIDET_OpenInterfaceConnect:
Indicates that the diagnostic event represents that a remote system has connected to the PRAESENSA system using the open interface.

OIDET_OpenInterfaceDisconnect:
Indicates that the diagnostic event represents that a remote system has disconnected from the PRAESENSA system using the open interface.

OIDET_OpenInterfaceConnectFailed:
Indicates that the diagnostic event represents that a remote system has attempted to connect to the PRAESENSA system using the open interface but failed.

OIDET_CallLoggingSuspended:
Indicates that call logging has been suspended because of a logging queue overflow.

OIDET_CallLoggingResumed:
Indicates that call logging has been resumed.

OIDET_UserLogIn:
Indicates that the diagnostic event represents that a user has logged in..

OIDET_UserLogOut:
Indicates that the diagnostic event represents that a user has logged out..

OIDET_UserLogInFailed:
Indicates that the diagnostic event represents that a login attempt has failed.

OIDET_BackupPowerModeStart:
Indicates that the backup power mode has started. This event is only generated when backup power mode (in the system settings) has been configured **not** to generate a fault event.

OIDET_BackupPowerModeEnd:
Indicates that the backup power mode has ended. This event is only generated when backup power mode (in the system settings) has been configured **not** to generate a fault event.

OIDET_ConfigurationRestored:
Indicates that the backup has been restored. It also indicates which parts of the configuration are restored (configuration, security settings, messages).

OIDET_DemoteToBackup:
Indicates that the current duty controller in a redundant system detected a critical fault and demoted itself to backup.

13.2.13.3 Fault Diagnostic Event-Group Event-types

Amplifier specific faults contain the Severity property (see chapter §13.5.4 for a description of the fault event classes). The severity can either be high or low. If the severity is high, the fault aggregates to a zone fault in the PRAESENSA system. This indicates that audio routing is not possible for (a part of) that zone.

OIDET_AudioPathSupervision:
Indicates that the diagnostic event represents detection of an audio-path failure.

OIDET_MicrophoneSupervision:
Indicates that the diagnostic event represents detection of microphone failure. Note that this diagnostic event only applies to a call station.

OIDET_SystemInputContact:
Indicates that the diagnostic event represents detection of a system input contact failure.

OIDET_CallStationExtension:
Indicates that the diagnostic event represents a mismatch between the number of configured call station extensions and the number of detected call station extensions.

OIDET_ConfigurationFile:
Indicates that the diagnostic event represents detection of a missing or corrupt configuration file.

OIDET_ConfigurationVersion:
Indicates that the diagnostic event represents a mismatch between the configuration file version and the required configuration file version. The configuration file requires conversion.

OIDET_IllegalConfiguration:
Indicates that the diagnostic event represents an inconsistency within the active configuration file: internal references between configuration items could not be verified.

OIDET_PrerecordedMessagesNames:
Indicates that the diagnostic event represents a mismatch between the configured (and used) prerecorded message-names and the detected prerecorded message-names.

OIDET_PrerecordedMessagesCorrupt:
Indicates that the diagnostic event represents one or more prerecorded messages in the PRAESENSA system is corrupt and cannot be used.

OIDET_UnitMissing:
Indicates that the diagnostic event represents a missing configured unit.

OIDET_UnitReset:
Indicates that the diagnostic event represents detection that a unit has restarted.

OIDET_UserInjectedFault:
Indicates that the diagnostic event represents a fault injected by a user or remote system.

OIDET_NoFaults:
Special event type that does not represent an actual fault, but is used to indicate that there are no existing fault events on the storage of the PRAESENSA system.

OIDET_ZoneLineFault:
Indicates that the diagnostic event represents that a Zone Line Fault that is injected by a remote system by triggering configured control input.

OIDET_NetworkChangeDiagEvent:
Indicates that the diagnostic event represents that there was a change in the network (broken links between devices).

OIDET_IncompatibleFirmware:
Indicates that the diagnostic event represents that a device contains incompatible firmware and cannot be used in the PRAESENSA system.

OIDET_Amp48VAFault:
This diagnostic event indicates the loss of 48V A supply for the amplifier. Severity is high if DET_Amp48VBFault is also reported.

OIDET_Amp48VBFault:
This diagnostic event indicates the loss 48V B supply. Severity is high if DET_Amp48VAFault is also reported.

OIDET_AmpChannelFault:
This diagnostic event indicates a channel fault internally in the amplifier. If not used already, the spare channel takes over the functionality of the channel. Severity is high if the spare channel is already in use.

OIDET_AmpShortCircuitLineAFault:
This diagnostic event indicates for the amplifier channel the hardware short detection is triggered or the output voltage is too low due to a short on line A.

OIDET_AmpShortCircuitLineBFault:
This diagnostic event indicates for the amplifier channel the hardware short detection is triggered or the output voltage is too low due to a short on line B.

OIDET_AmpAcc18VFault:
This diagnostic event indicates failure of the amplifier lifeline power supply. The severity is not used.

OIDET_AmpSpareInternalFault:
This diagnostic event indicates an internal failure in the amplifier spare channel and can no longer be used. Severity is always high.

OIDET_AmpChannelOverloadFault:
This diagnostic event indicates for the amplifier channel an output overload has occurred.

OIDET_AmpEolFailureLineAFault:
This diagnostic event indicates that the end-of-line device for the amplifier channel on line A is disconnected (the end-of-line pilot tone is not present).

OIDET_AmpEolFailureLineBFault:
This diagnostic event indicates that the end-of-line device for the amplifier channel on line B is disconnected (the end-of-line pilot tone is not present).

OIDET_GroundShortFault:
This diagnostic event indicates that a ground fault is signaled by the amplifier hardware.

OIDET_OverheatFault:
This diagnostic event indicates that amplifier hardware is overheated. All channels are disabled and severity is always high.

OIDET_PowerMainsSupply:
Indicates that the diagnostic event represents detection of loss of mains power for a Multifunction Power Supply.

OIDET_PowerBackupSupply:
Indicates that the diagnostic event represents detection of loss of the backup power supply for a Multifunction Power Supply.

OIDET_MainsAbsentPSU1Fault:
This diagnostic event indicates absence of the output 1 mains power. The number matches the screening at the back-panel of the Multifunction Power Supply .

OIDET_MainsAbsentPSU2Fault:
This diagnostic event indicates absence of the output 2 mains power. The number matches the screening at the back-panel of the Multifunction Power Supply .

OIDET_MainsAbsentPSU3Fault:
This diagnostic event indicates absence of the output 3 mains power. The number matches the screening at the back-panel of the Multifunction Power Supply .

OIDET_BackupAbsentPSU1Fault:
This diagnostic event indicates absence of the output 1 12V DC backup power. The number matches the screening at the back-panel of the multifunction power supply.

OIDET_BackupAbsentPSU2Fault:
This diagnostic event indicates absence of the output 2 12V DC backup power. The number matches the screening at the back-panel of the Multifunction power supply.

OIDET_BackupAbsentPSU3Fault:
This diagnostic event indicates absence of the output 3 12V DC backup power. The number matches the screening at the back-panel of the Multifunction Power Supply .

OIDET_DcOut1PSU1Fault:
This diagnostic event indicates a missing 48V DC output for connector 1A. The numbers match the screening at the back-panel of the Multifunction Power Supply .

OIDET_DcOut2PSU1Fault:
This diagnostic event indicates a missing 48V DC output for connector 1B. The numbers match the screening at the back-panel of the Multifunction Power Supply .

OIDET_DcOut1PSU2Fault:
This diagnostic event indicates a missing 48V DC output for connector 2A. The numbers match the screening at the back-panel of the Multifunction Power Supply .

OIDET_DcOut2PSU2Fault:
This diagnostic event indicates a missing 48V DC output for connector 2B. The numbers match the screening at the back-panel of the Multifunction Power Supply .

OIDET_DcOut1PSU3Fault:
This diagnostic event indicates a missing 48V DC output for connector 3A. The numbers match the screening at the back-panel of the Multifunction Power Supply .

OIDET_DcOut2PSU3Fault:
This diagnostic event indicates a missing 48V DC output for connector 3B. The numbers match the screening at the back-panel of the Multifunction Power Supply .

OIDET_AudioLifelinePSU1Fault:
This diagnostic event indicates a wiring problem in the ACC connector with the lifeline analog audio signal for output 1. The number matches the screening at the back-panel of the Multifunction Power Supply .

OIDET_AudioLifelinePSU2Fault:
This diagnostic event indicates a wiring problem in the ACC connector with the lifeline analog audio signal for output 2. The number matches the screening at the back-panel of the Multifunction Power Supply .

OIDET_AudioLifelinePSU3Fault:
This diagnostic event indicates a wiring problem in the ACC connector with the lifeline analog audio signal for output 3. The number matches the screening at the back-panel of the Multifunction Power Supply .

OIDET_AccSupplyPSU1Fault:
This diagnostic event indicates a missing 10 to 18V at the ACC connector for output 1. The number matches the screening at the back-panel of the Multifunction Power Supply .

OIDET_AccSupplyPSU2Fault:
This diagnostic event indicates a missing 10 to 18V at the ACC connector for output 2. The number matches the screening at the back-panel of the Multifunction Power Supply .

OIDET_AccSupplyPSU3Fault:
This diagnostic event indicates a missing 10 to 18V at the ACC connector for output 3. The number matches the screening at the back-panel of the Multifunction Power Supply .

OIDET_Fan1Fault:
This diagnostic event indicates that fan 1 in the Multifunction Power Supply is broken.

OIDET_Fan2Fault:
This diagnostic event indicates that fan 2 in the Multifunction Power Supply is broken.

OIDET_DcAux1Fault:
This diagnostic event indicates the absence of 24V DC aux 1 voltage of the Multifunction Power Supply. The number matches the screening at the back-panel of the device.

OIDET_DcAux2Fault:
This diagnostic event indicates the absence of 24V DC aux 2 voltage of the Multifunction Power Supply. The number matches the screening at the back-panel of the device.

OIDET_BatteryShortFault:
This diagnostic event indicates a short in the external battery of the Multifunction Power Supply.

OIDET_BatteryRiFault:
This diagnostic event indicates a Ri fault for the connected battery of the Multifunction Power Supply. This fault depends on the configured battery capacity in the PRAESENSA system if a fault is reported.

OIDET_BatteryOverheatFault:
This diagnostic event indicates that the temperature of the connected battery of the Multifunction Power Supply is not in correct working range

OIDET_BatteryFloatChargeFault:
This diagnostic event indicates that the battery of the Multifunction Power Supply is most likely broken. The charger enters a float state when the State of Charge (SoC) is 100%. In this state a low charge current is expected just to compensate the self-discharge of the battery. When the charge current is very high the battery is probably broken and therefore the fault is reported. The charger is suspended for safety reasons.

OIDET_MainsAbsentChargerFault:
This diagnostic event indicates that the mains converter for the charger is defect which prevents charging the battery correctly.

OIDET_PoESupplyFault:
This diagnostic event indicates that a mismatch is detected the number of Power over Ethernet connections to the call station and the number of expected Power Over Ethernet inputs configured in the PRAESENSA system.

OIDET_PowerSupplyAFault:
This diagnostic event indicates that the power supply input A level on the system controller is not within range. The fault is only reported if the power supply input is configured to be supervised in the PRAESENSA system.

OIDET_PowerSupplyBFault:
This diagnostic event indicates that the power supply input B level on the system controller is not within range. The fault is only reported if the power supply input is configured to be supervised in the PRAESENSA system.

OIDET_ExternalPowerFault:
This diagnostic event indicates that the PRAESENSA system is now in backup power mode. This event is only generated when backup power mode (in the system settings) has been configured to generate a fault event.

OIDET_ChargerSupplyVoltageTooLowFault:
This diagnostic event indicates that the charger supply voltage is too low which prevents charging the battery correctly.

OIDET_BatteryOvervoltageFault:
This diagnostic event indicates that the internal charger is defect and is switched off for safety reasons.

OIDET_BatteryUndervoltageFault:
This diagnostic event indicates that there is an undervoltage situation when mains is absent. The battery is too empty to operate on.

OIDET_MediaClockFault:
This diagnostic event indicates there are one or more devices that failed to lock to PTP for a longer period of time.

OIDET_ChargerFault

This diagnostics event indicates an internal charger fault which prevents charging the battery correctly.

OIDET_Amp20VFault

This diagnostic event indicates the failure of the power convertor for the controller section of the amplifier.

OIDET_AmpPsuFault

This diagnostic event indicates the failure of the power convertor for the audio section of the amplifier.

OIDET_NetworkLatencyFault

This diagnostic event indicates that an audio flow gets interrupted by network delay and network jitter.

OIDET_SynchronizationFault

This diagnostic event indicates that the configuration synchronization between a backup controller and the master controller of a redundant system failed.

13.2.14 TOIDiagEventGroup

The `TOIDiagEventGroup` type divides each event into groups. Each event belongs to maximum one group. The groups are used to divide the event generation. The group-type is used for subscription of the events. The relation between the groups and the event-types is given in section 13.2.11, presented as sub-sections.

OIDEG_CALLEVENTGROUP:
Indicates that the diagnostic event is related to call events.

OIDEG_GENERALEVENTGROUP:
Indicates that the diagnostic event represents a general event.

OIDEG_FAULTEVENTGROUP:
Indicates that the diagnostic event represents a fault event. Faults have a state and can be acknowledged, resolved or reset.

OIDEG_UNKNOWNDIAGEVENTGROUP = UINT_MAX:
Indicates that the diagnostic event couldn’t be grouped in one of the groups above.

13.2.15 TOIEventOriginatorType

The `TOIEventOriginatorType` type represents the various types of the originators that generated the received event.

OIEOT_NOEVENTORIGINATOR = 0x00477002:
Indicates that the event originator is not known.

OIEOT_UNITEVENTORIGINATOR = 0x00477003:
Indicates that the event originator is a unit.

OIEOT_OPENINTERFACEEVENTORIGINATOR = 0x00477004:
Indicates that the event originator is a system connected to the open interface of the PRAESENSA system.

OIEOT_CONTROLINPUTEVENTORIGINATOR = 0x00477005:
Indicates that the event originator is a control-input.

OIEOT_AUDIOOUTPUTEVENTORIGINATOR = 0x00477006:
Indicates that the event originator is an audio-output.

OIEOT_AUDIOINPUTEVENTORIGINATOR = 0x00477007:
Indicates that the event originator is an audio input.

OIEOT_USEREVENTORIGINATOR = 0x00477009:
Indicates that the event originator is a user.

OIEOT_NETWORKEVENTORIGINATOR = 0x0047700A:
Indicates that the event originator represents a network connection. Used for user login events.

13.2.16 TOIDiagEventState

The `TOIDiagEventState` type represents the state of the fault-group diagnostic events. Other diagnostic event-types always will have the state RESET.

OIDES_NEW:
Indicates that the diagnostic event is added to the system.

OIDES_ACKNOWLEDGED:
Indicates that the diagnostic fault event is acknowledged.

OIDES_RESOLVED:
Indicates that the diagnostic fault event is resolved.

OIDES_RESET:
Indicates that the diagnostic fault event is reset.

13.2.17 TOIActionType

The TOIActionType type represents the action done on the Fault-type events. Other diagnostic event-types always received the action type NEW or REMOVED.

OIACT_NEW:
Indicates that the Diagnostic event is added to the system.

OIACT_ACKNOWLEDGED:
Indicates that the Diagnostic event is acknowledged (fault events only).

OIACT_RESOLVED:
Indicates that the Diagnostic event is resolved (fault events only).

OIACT_RESET:
Indicates that the Diagnostic event is reset (fault events only).

OIACT_UPDATED:
Indicates that the Diagnostic event is updated (additional information is added to an existing event)

OIACT_REMOVED:
Indicates that the Diagnostic event is removed from the system.

OIACT_ EXISTING:
The specified diagnostic event is already present in the PRAESENSA System. This action type is passed for each diagnostic event already present on the storage of the PRAESENSA System after subscription for the events.

OIACT_EXISTING_LAST:
The specified diagnostic event is already present on the PRAESENSA System storage and it is the last present event sent, or there are actually no fault events present on the storage of the PRAESENSA System, in which case the specified diagnostic event is of type OIDET_NOFAULTS.

13.2.18 TOICallOutputHandling

Describes how calls behave on routing availability.

OICOH_PARTIAL:
Partial calls are calls that proceed even in case not all required zones are available.

OICOH_NON_PARTIAL:
Non-partial calls are calls that require the entire routing to be available at the start of the call and during the call. When during the call a part of the routing becomes unavailable, the call is aborted. Currently not supported in the PRAESENSA system.

OICOH_STACKED:
Stacked calls are calls that extend partial calls with replays to previously unavailable zones. Currently not supported in the PRAESENSA system.

13.2.19 TOICallStackingMode

Describes when recorded calls replay. A stacked call or a stacked call waits for each zone to become available for replay.

OICSM_WAIT_FOR_ALL:
Wait with replay for all zones to become available

OICSM_WAIT_FOR_EACH:
Start a replay for each zone to become available

13.2.20 TOICallTiming

Indicates the way the call must be handled.

OICTM_IMMEDIATE:
Broadcast to the selected zones and zone groups when the call is started.

OICSM_TIME_SHIFTED:
Broadcast to the selected zones and zone groups when the original call is finished to prevent audio feedback during live speech.

OICSM_MONITORED:
Broadcast when the call is not cancelled within 2 seconds after the monitoring phase has finished.

13.3 Methods

13.3.1 Connect

Make a connection with a system controller. A connection is required before other methods can be used. The connection is done using the port number 9401 (non-secure) or 9403 (secure).

For secure connections, TLS 1.2 is used. The certificate supplied by the system controller is automatically accepted without validation.

```
TOIErrorCode Connect(string hostnameOrIP, string username, string
      password, bool secure = true)
```

Parameters:

hostnameOrIP	IP address of the system controller, format “127.0.0.1” or the DNS name of the PRAESENSA system controller.

username	Name of the user as defined during the “User Management” configuration of the PRAESENSA system.
password	Password of the user.
secure	Use a secure connection. True by default.

Return value:
Error code indicating success or failure.

13.3.2 Disconnect

Gracefully terminates a connection with the system controller. After a successful call to this function it is no longer possible to use functions of the open interface.

```
TOIErrorCode Disconnect()
```

Return value:
Error code indicating success or failure.

13.3.3 GetNcoVersion

Retrieves the software release of the system controller.

```
TOIErrorCode GetNcoVersion(out string release)
```

Parameters:

release	Software release of the system controller

Return value:
Error code indicating success or failure.

13.3.4 GetProtocolVersion

Retrieves the protocol version of the Open Interface as `major.minor`. This is not the software version of the PRAESENSA system.

```
TOIErrorCode GetProtocolVersion(out int major, out int minor)
```

Parameter

major	Major version number of the Open Interface protocol
minor	Minor version number of the Open Interface protocol

Error codes:
Error code indicating success or failure.

13.3.5 CreateCallEx2

Create (but do not start) a call with the given parameters.

```
TOIErrorCode CreateCallEx2(List<string> routing, uint priority,
      TOICallOutputHandling outputHandling, TOICallStackingMode
      stackingMode, uint stackingTimeout, string startchime, string
      endchime, bool livespeech, string audioinput, List<string>
      messages, uint repeat, TOICallTiming callTiming, string
      preMonitorDest, uint liveSpeechAttenuation, uint
      startChimeAttenuation, uint endChimeAttenuation, uint
      messageAttenuation, out uint callId)
```

Parameters:

routing	List of names of zone groups, zones and/or control outputs. The routing is formatted as a comma separated set of resource names.
priority	The priority of the call. See §13.2.5 for the value range definitions.
outputHandling	Whether the call is partial, non-partial or stacked. There are three possible values: OICOH_PARTIAL, OICOH_NON_PARTIAL and OICOH_STACKED. Partial calls are calls that proceed even in case not all required zones are available. Non-partial calls are calls

	that require the entire routing to be available at the start of the call and during the call. When during the call a part of the routing becomes unavailable, the call is aborted. Stacked calls are calls that extend partial calls with replays to previously unavailable zones. Stacked calls are only available within the business call priority range. This means that stacking emergency and BGM priority calls is not possible.
stackingMode	Whether a stacked call waits for all zones to become available or a stacked call waits for each zone to become available for replay. There are two possible values: OICSM_WAIT_FOR_ALL and OICSM_WAIT_FOR_EACH. This parameter is ignored when outputHandling is set to OICOH_PARTIAL or OICOH_NON_PARTIAL.
stackingTimeout	Amount of minutes for a stacked call to wait for available resources. The time-out countdown is started at the moment the original call has ended. The accepted range is 1 to 255 minutes; the value OICST_INFINITE is used to wait infinitely. This parameter is ignored when outputHandling is set to OICOH_PARTIAL or OICOH_NON_PARTIAL.
startChime	The name of the start chime.
endChime	The name of the end chime.
liveSpeech	Whether or not the call has a live speech phase. `True` = live speech, `False` = no live speech.
audioInput	Name of the audio Input (only used when live speech is true).
messages	List of names of prerecorded messages. The messages parameter is formatted as a comma separated set of message names.
repeat	How many times the messages should be repeated. If the messages needs to be played only once or forever, then the enumerated values of the `TOICallRepeatCount` can be used (see §**Error! Reference source not found.**). Otherwise, the following value range can be used: 1 .. 32767.
callTiming	Indicates the way the call must be handled. There are three possible values: OICTM_IMMEDIATE, OICTM_TIME_SHIFTED and OICTM_MONITORED. An immediate call will be broadcast to the selected zones and zone groups when the call is started. A time shifted call will be broadcast to the selected zones and zone groups when the original call is finished to prevent audio feedback during live speech. A monitored call will broadcast when it is not cancelled within 2 seconds after the monitoring phase has finished.
preMonitorDest	The destination zone of the pre-monitor phase of a pre-monitored call. When the call is not pre-monitored, this value is ignored. This parameter is ignored when callTiming is set to OICTM_IMMEDIATE or OICTM_TIME_SHIFTED.
liveSpeechAttenuation	The attenuation to be used for the audio input during the live speech phase. Range: 0..60 dB.
startChimeAttenuation	The attenuation to be used for the chime generator during the start chime phase. Range: 0..60 dB.
endChimeAttenuation	The attenuation to be used for the chime generator during the end chime phase. Range: 0..60 dB.
messageAttenuation	The attenuation to be used for the message generator during the prerecorded message phase. Range: 0..60 dB.
callId	Unique identification of the call (only valid when return value is OIERROR_OK)

Return value:
Error code indicating success or failure.

13.3.6 CreateCallEx3

Create (but do not start) a call with the given parameters.

```
TOIErrorCode CreateCallEx3(List<string> routing, uint priority,
      TOICallOutputHandling outputHandling, TOICallStackingMode
      stackingMode, uint stackingTimeout, string startchime, string
      endchime, bool livespeech, string audioinput, List<string>
      messages, uint repeat, TOICallTiming callTiming, string
      preMonitorDest, uint liveSpeechAttenuation, uint
      startChimeAttenuation, uint endChimeAttenuation, uint
      messageAttenuation, bool restartCall, out uint callId)
```

Parameters:

routing	List of names of zone groups, zones and/or control outputs.
priority	The priority of the call. See §13.2.5 for the value range definitions.
outputHandling	Whether the call is partial, non-partial or stacked. There are three possible values: OICOH_PARTIAL, OICOH_NON_PARTIAL and OICOH_STACKED. Partial calls are calls that proceed even in case not all required zones are available. Non-partial calls are calls that require the entire routing to be available at the start of the call and during the call. When during the call a part of the routing becomes unavailable, the call is aborted. Stacked calls are calls that extend partial calls with replays to previously unavailable zones. Stacked calls are only available within the business call priority range. This means that stacking emergency and BGM priority calls is not possible.
stackingMode	Whether a stacked call waits for all zones to become available or a stacked call waits for each zone to become available for replay. There are two possible values: OICSM_WAIT_FOR_ALL and OICSM_WAIT_FOR_EACH. This parameter is ignored when outputHandling is set to OICOH_PARTIAL or OICOH_NON_PARTIAL.
stackingTimeout	Amount of minutes for a stacked call to wait for available resources. The time-out countdown is started at the moment the original call has ended. The accepted range is 1 to 255 minutes; the value OICST_INFINITE is used to wait infinitely. This parameter is ignored when outputHandling is set to OICOH_PARTIAL or OICOH_NON_PARTIAL.
startChime	The name of the start chime.
endChime	The name of the end chime.
liveSpeech	Whether or not the call has a live speech phase. `True` = live speech, `False` = no live speech.
audioInput	Name of the audio Input (only used when live speech is true).
messages	List of names of prerecorded messages. The messages parameter is formatted as a comma separated set of message names.
repeat	How many times the messages should be repeated. If the messages needs to be played only once or forever, then the enumerated values of the `TOICallRepeatCount` can be used (see §**Error! Reference source not found.**). Otherwise, the following value range can be used: 1 .. 32767.
callTiming	Indicates the way the call must be handled. There are

	three possible values: OICTM_IMMEDIATE, OICTM_TIME_SHIFTED and OICTM_MONITORED. An immediate call will be broadcast to the selected zones and zone groups when the call is started. A time shifted call will be broadcast to the selected zones and zone groups when the original call is finished to prevent audio feedback during live speech. A monitored call will broadcast when it is not cancelled within 2 seconds after the monitoring phase has finished.
preMonitorDest	The destination zone of the pre-monitor phase of a pre-monitored call. When the call is not pre-monitored, this value is ignored. This parameter is ignored when callTiming is set to OICTM_IMMEDIATE or OICTM_TIME_SHIFTED.
liveSpeechAttenuation	The attenuation to be used for the audio input during the live speech phase. Range: 0..60 dB.
startChimeAttenuation	The attenuation to be used for the chime generator during the start chime phase. Range: 0..60 dB.
endChimeAttenuation	The attenuation to be used for the chime generator during the end chime phase. Range: 0..60 dB.
messageAttenuation	The attenuation to be used for the message generator during the prerecorded message phase. Range: 0..60 dB.
restartCall	Indication if the call should be restarted after an interruption.
callId	Unique identification of the call (only valid when return value is OIERROR_OK).

Return value:
Error code indicating success or failure

13.3.7 StartCreatedCall

Start a previously created call. If the call was started successfully, call state update events for this call will be sent via the Open Interface.

```
TOIErrorCode StartCreatedCall(uint callId)
```

Parameters:

callId	unique identification of the call, returned by `CreateCallEx2` or `CreateCallEx3`.

Return value:
Error code indicating success or failure.

13.3.8 StopCall

Stop a previously created or started call.

```
TOIErrorCode StopCall(uint callId)
```

Parameters:

callId	Unique identification of the call, returned by `CreateCallEx2` or `CreateCallEx3`.

Return value:
Error code indicating success or failure.

13.3.9 AbortCall

Abort a previously created or started call.

```
TOIErrorCode AbortCall(uint callId)
```

Parameters:

callId	Unique identification of the call, returned from

`CreateCallEx2` or `CreateCallEx3`.

Return value:
Error code indicating success or failure

13.3.10 CancelAll

Cancel all available stacked calls that were started by this connection.

```
TOIErrorCode CancelAll()
```

Return value:
Error code indicating success or failure

13.3.11 CancelLast

Cancel (if still available) the last stacked call that was started by this connection.

```
TOIErrorCode CancelLast()
```

Return value:
Error code indicating success or failure.

13.3.12 AddToCall

Add resources to a previously created or started call.

```
TOIErrorCode AddToCall(uint callId, List<string> resources)
```

Parameters:

callId	Unique identification of the call, returned from `CreateCallEx2` or `CreateCallEx3`.
resources	List of names of zone groups, zones and/or control outputs to be added to the call. A comma separates each name in the routing list.

Return value:
Error code indicating success or failure.

13.3.13 RemoveFromCall

Removes resources from the running call.

```
TOIErrorCode RemoveFromCall(uint callId, List<string> resources)
```

Parameters:

callId	Unique identification of the call, returned from `CreateCallEx2` or `CreateCallEx3`.
resources	List of names of zone groups, zones and/or control outputs to be removed from the call. A comma separates each name in the routing list.

Return value:
Error code indicating success or failure.

13.3.14 ReportFault

Reports a fault diagnostics event in the system. The fault will be reported as a DET_UserInjectedFault.

```
TOIErrorCode ReportFault(string faultname, out uint eventId)
```

Parameters:

faultname	Textual representation of the fault to be reported.
eventId	Identification of the diagnostic fault event reported (only valid when return value is OIERROR_OK).

Return value:
Error code indicating success or failure.

13.3.15 ResolveFault

Resolve a specific diagnostic fault event. The received `eventId` of the `ReportFault` function or a diagnostic event should be used as parameter.

```
TOIErrorCode ResolveFault(uint eventid)
```

Parameters:

eventId	Identification of the diagnostic fault event, received by the `ReportFault` function.

Return value:
Error code indicating success or failure.

13.3.16 AckFault

Acknowledge a specific diagnostic fault event. Because the fault alarm depends on the states of all fault events, this function can possibly acknowledge the system fault alarm state (in case it was the last non-acknowledged fault). If the fault alarm changes state, this will indicate an alarm state change using the `AlarmUpdate` event.

```
TOIErrorCode AckFault(uint eventid)
```

Parameters:

eventId	Identification of the diagnostic fault event.

Return value:
Error code indicating success or failure.

13.3.17 ResetFault

Reset a specific diagnostic fault event. Because the fault alarm depends on the states of all fault events, this function can possibly reset the system fault alarm state (in case it was the last non-reset fault). If the fault alarm changes state, this will indicate an alarm state change using the `AlarmUpdate` event.

```
TOIErrorCode ResetFault(uint eventid)
```

Parameters:

eventId	Identification of the diagnostic fault event.

Return value:
Error code indicating success or failure.

13.3.18 AckAllFaults

Acknowledges all fault events. Because the fault alarm depends on the states of all fault events, this will also acknowledge the fault alarm. If the fault alarm changes state, this will indicate an alarm state change using the `AlarmUpdate` event.

```
TOIErrorCode AckAllFaults()
```

Return value:
Error code indicating success or failure.

13.3.19 ResetAllFaults

Resets all fault events. Because the fault alarm depends on the state of all fault events, this can possibly reset the fault alarm, dependent whether the faults are resolved. If the fault alarm changes state, this will indicate an alarm state change using the `AlarmUpdate` event.

```
TOIErrorCode ResetAllFaults()
```

Return value:
Error code indicating success or failure.

13.3.20 AckEvacAlarm

Acknowledges the emergency alarm. If the emergency alarm changes state, this will indicate an alarm state change using the `AlarmUpdate` event.

```
TOIErrorCode AckEvacAlarm()
```

Return value:
Error code indicating success or failure.

13.3.21 ResetEvacAlarmEx

Resets the emergency alarm. If the emergency alarm changes state, this will indicate an alarm state change using the `AlarmUpdate` event.

```
TOIErrorCode ResetEvacAlarmEx(bool bAbortEvacCalls)
```

Parameters:

bAbortEvacCalls	Whether or not currently running evacuation priority calls must be aborted. `true` = abort running evacuation priority calls, `false` = do not abort running evacuation priority calls.

Return value:
Error code indicating success or failure.

13.3.22 AckFaultAlarm

Acknowledges the emergency alarm. If the fault alarm changes state, this will indicate an alarm state change using the `AlarmUpdate` event.

```
TOIErrorCode AckFaultAlarm()
```

Return value:
Error code indicating success or failure.

13.3.23 ResetFaultAlarm

Resets the fault alarm. If the fault alarm changes state, this will indicate an alarm state change using the `AlarmUpdate` event.

```
TOIErrorCode ResetFaultAlarm()
```

Return value:
Error code indicating success or failure.

13.3.24 GetAudioInputNames

Retrieve the list of configured audio inputs.

```
TOIErrorCode GetAudioInputNames(out List<string> names)
```

Parameters:

names	The list with the names of all configured audio inputs.

Return value:
Error code indicating success or failure.

13.3.25 GetBgmChannelNames

Retrieve the list of configured BGM channels.

```
TOIErrorCode GetBgmChannelNames(out List<string> names)
```

Parameters:

names	The list with the names of all configured BGM channels.

Return value:
Error code indicating success or failure.

13.3.26 GetChimeNames

Retrieves the list of configured chime names. Note that this is the same list as returned by `GetMessageNames`.

```
TOIErrorCode GetChimeNames(out List<string> names)
```

Parameters:

names	Comma (,) separated list of chime names.

Return value:
Error code indicating success or failure.

13.3.27 GetMessageNames

Retrieves the list of configured message names. Note that the chimes are also messages and therefore are included in the list of message names.

```
TOIErrorCode GetMessageNames(out List<string> names)
```

Parameters:

names	Comma (,) separated list of message names.

Return value:
Error code indicating success or failure.

13.3.28 GetZoneGroupNames

Retrieves the list of configured zone group names.

```
TOIErrorCode GetZoneGroupNames(out List<string> names)
```

Parameters:

names	List of zone group names.

Return value:
Error code indicating success or failure.

13.3.29 GetZoneNames

Retrieve the list of configured zone names. When the zone group parameter is empty all zone names are returned otherwise the zone names in that zone group are returned.

```
TOIErrorCode GetZoneNames(string sZoneGroupName, out List<string>
      names)
```

Parameters:

sZoneGroupName	The zone group to get the names of (empty string for all zones).
names	The names of the zones

Return value:
Error code indicating success or failure.

13.3.30 GetConfigId

Retrieve the configuration identifier from the PRAESENSA system. This is a number which is increased each time the configuration is saved.

```
TOIErrorCode GetConfigId(out uint configId)
```

Parameters:

configId	Configuration identifier.

Return value:
Error code indicating success or failure.

13.3.31 SetSubscriptionResources

Subscribe or unsubscribe the Open Interface client to resource (read zone groups, zones or control outputs) state updates of particular resources. Only when a subscription is set for a resource, resource state updates will be sent for that resource. When a subscription is set for a resource, the `ResourceState` event will be used with the current state of that resource.

```
TOIErrorCode SetSubscriptionResources(bool bSub, List<string>
      resources)
```

Parameters:

bSub	Whether to subscribe or unsubscribe. `true` = subscribe, `false` = unsubscribe.
resources	List of names of zone groups, zones and/or control outputs. A comma separates each name in the routing list. Resources already having the subscription state are ignored.

Return value:
Error code indicating success or failure.

13.3.32 SetSubscriptionResourcesFaultState

Subscribes or unsubscribes to resource (read zone groups or zones) fault state notifications of particular resources for faults that affect the audio distribution of that zone or zone group. Only when a subscription is set for a resource, resource fault state notifications are sent for that resource. When a subscription is set for a resource, the `ResourceFaultState` event will be used with the current state of that resource.

```
TOIErrorCode SetSubscriptionResourcesFaultState(bool bSub,
      List<string> resources)
```

Parameters:

bSub	Whether to subscribe or unsubscribe. `true` = subscribe, `false` = unsubscribe.
resources	List of names of zone groups and/or zones. A comma separates each name in the routing list. Resources already having the subscription state are ignored. Subscription for control output resources is not allowed.

Return value:
Error code indicating success or failure.

13.3.33 SetSubscriptionBgmVolume

Subscribes or unsubscribes the client to BGM volume updates. Only when a subscription is set for a BGM channel, BGM volume updates will be sent for that BGM zone. When a subscription is set for a BGM zone, the `BgmVolumeChanged` event will be used with the current volume of that BGM channel.

```
TOIErrorCode SetSubscriptionBgmVolume(bool bSub, List<string>
      resources)
```

Parameters:

bSub	Whether to subscribe or unsubscribe. `true` = subscribe, `false` = unsubscribe.
resources	List of BGM channel names.

Return value:
Error code indicating success or failure.

13.3.34 SetSubscriptionBgmRouting

Subscribes or unsubscribes the client to BGM routing updates. Only when a subscription is set for a BGM channel, BGM routing updates will be sent for that BGM channel. When a subscription

is set for a BGM channel the `BgmRoutingChanged` event will be used with the current routing of that BGM channel.

```
TOIErrorCode SetSubscriptionBgmRouting(bool bSub, string channel)
```

Parameters:

bSub	Whether to subscribe or unsubscribe. `true` = subscribe, `false` = unsubscribe.
channel	name of the BGM channel.

Return value:
Error code indicating success or failure.

13.3.35 SetSubscriptionEvents

Subscribe or unsubscribe the Open Interface client to diagnostic event updates. Only when a subscription is set for an event group and there are events, diagnostic event updates will be sent for that group. When a subscription is set for an event group, the `DiagEventNotification` event will be used with the diagnostic event for that group.

```
TOIErrorCode SetSubscriptionEvents(bool bSubscribe, TOIDiagEventGroup
      eventGroup)
```

Parameters:

bSub	Whether to subscribe or unsubscribe. `true` = subscribe, `false` = unsubscribe.
eventGroup	Group identification of the diagnostic events. The associated event-types for each group is represented in §13.2.14.

Return value:
Error code indicating success or failure.

13.3.36 SetSubscriptionAlarm

Subscribe or unsubscribe the client to fault or evac alarm state updates. Only when a subscription is set for the fault alarm, fault alarm state updates will be sent. When a subscription is set for an alarm type, the `AlarmUpdate` event will be used with the alarm state for that alarm type.

```
TOIErrorCode SetSubscriptionAlarm(TOIAlarmType alarmType, bool bSub)
```

Parameters:

alarmType	Alarm type to subscribe or unsubscribe to. The associated alarm type is represented in §13.2.3.
bSub	Whether to subscribe or unsubscribe. `true` = subscribe, `false` = unsubscribe.

Return value:
Error code indicating success or failure.

13.3.37 SetSubscriptionUnitCount

Subscribe or unsubscribe the Open Interface client to connected unit count updates. Only when a subscription is set for the unit count, unit count updates will be sent. When a subscription is set, the `UnitCountChanged` event will be used with the current number of connected units.

```
TOIErrorCode SetSubscriptionUnitCount(bool bSub)
```

Parameters:

bSub	Whether to subscribe or unsubscribe. `true` = subscribe, `false` = unsubscribe.

Return value:
Error code indicating success or failure.

13.3.38 IncrementBgmVolume

Increments the BGM volume of routing with 3 dB.

```
TOIErrorCode IncrementBgmVolume(List<string> resources)
```

Parameters:

resources	List of names of zone groups and/or zones.

Return value:
Error code indicating success or failure.

13.3.39 DecrementBgmVolume

Decrements the BGM volume of routing with 3 dB.

```
TOIErrorCode DecrementBgmVolume(List<string> resources)
```

Parameters:

resources	List of names of zone groups and/or zones.

Return value:
Error code indicating success or failure.

13.3.40 IncrementBgmChannelVolume

Increments the BGM volume of a channel with 3 dB.

```
TOIErrorCode IncrementBgmChannelVolume(string channelName)
```

Parameters:

channelName	BGM channel name

Return value:
Error code indicating success or failure.

13.3.41 DecrementBgmChannelVolume

Decrements the BGM volume of a channel with 3 dB.

```
TOIErrorCode DecrementBgmChannelVolume(string channelName)
```

Parameters:

channelName	BGM channel name

Return value:
Error code indicating success or failure.

13.3.42 SetBgmVolume

Sets the BGM volume of routing.

```
TOIErrorCode SetBgmVolume(int volume, List<string> resources)
```

Parameters:

volume	Volume to set. Value range: 0 .. –96 (dB). Use –96 (dB) to mute the BGM.
resources	List of names of zone groups and/or zones.

Return value:
Error code indicating success or failure.

13.3.43 AddBgmRouting

Adds a routing to a BGM channel. Either all specified routing is added or, in case of an error, no routing at all.

```
TOIErrorCode AddBgmRouting(string channel, List<string> resources)
```

Parameters:

channel	Name of the BGM channel.
resources	List of names of zone groups and/or zones.

Return value:
Error code indicating success or failure.

13.3.44 RemoveBgmRouting

Removes routing from a BGM channel. Either all specified routing is removed or, in case of an error, no routing at all.

```
TOIErrorCode RemoveBgmRouting(string channel, List<string> resources)
```

Parameters:

channel	Name of the BGM channel.
resources	List of names of zone groups and/or zones.

Return value:
Error code indicating success or failure.

13.3.45 ToggleBgmRouting

Toggles routing in a BGM channel. When none of names in the specified routing are part the BGM channel, all specified routing is added, else all supplied routing is removed or, in case of an error, the current routing of the BGM channel remains unchanged.

```
TOIErrorCode ToggleBgmRouting(string channel, List<string> resources)
```

Parameters:

channel	Name of the BGM channel.
resources	List of names of zone groups and/or zones.

Return value:
Error code indicating success or failure.

13.3.46 SetBgmRouting

Sets the routing of a BGM channel. Either replaces the current routing of a BGM channel with all specified routing or, in case of an error, the current routing of the BGM channel remains unchanged.

```
TOIErrorCode SetBgmRouting(string channel, List<string> resources)
```

Parameters:

channel	Name of the BGM channel.
resources	List of names of zone groups and/or zones.

Return value:
Error code indicating success or failure.

13.3.47 ActivateVirtualControlInput

Activate a virtual control input. If the virtual control input is already active then activating it again will not have any effect.

```
TOIErrorCode ActivateVirtualControlInput(string virtualControlInput)
```

Parameters:

virtualControlInput	Name of the virtual control input to activate.

Return value:
Error code indicating success or failure.

13.3.48 DeactivateVirtualControlInput

Deactivate a virtual control input. If the virtual control input is already inactive then deactivating it again will not have any effect.

```
TOIErrorCode DeactivateVirtualControlInput(string virtualControlInput,
      TOIVirtualControlInputDeactivation deactivationType)
```

Parameters:

virtualControlInput	Name of the virtual control input to deactivate.
deactivationType	Specifier how the associated action should be deactivated (see §13.2.11).

Return value:
Error code indicating success or failure.

13.3.49 SetSubscriptionVirtualControlInputs

Subscribe or unsubscribe the Open Interface client to virtual control input state updates. Only when a subscription is set for the virtual control input state, virtual control input state updates will be sent. When a subscription is set, the `VirtualControlInputStateChanged` event will be used with the current state of the virtual control inputs.

```
TOIErrorCode SetSubscriptionVirtualControlInputs(bool bSubscription,
      List<string> virtualControlInputs)
```

Parameters:

bSubscription	Whether to subscribe or unsubscribe. `true` = subscribe, `false` = unsubscribe.
virtualControlInputs	List of names of virtual control inputs.

Return value:
Error code indicating success or failure.

13.3.50 GetVirtualControlInputNames

Retrieve the configured virtual control input names.

```
TOIErrorCode GetVirtualControlInputNames(out List<string> names)
```

Parameters:

names	List with the names of virtual control input names.

Return value:
Error code indicating success or failure.

13.3.51 GetConfiguredUnits

Retrieve the list of configured units from the PRAESENSA system. Only the units that are enabled are returned.

```
TOIErrorCode GetConfiguredUnits(out List<string> units)
```

Parameters:

names	List of unit names.

Return value:
Error code indicating success or failure.

13.3.52 GetConnectedUnits

Retrieve the list of connected units from the PRAESENSA system. Only the units that are configured, enabled and connected with the correct software release (units that can be controlled) are returned.

```
TOIErrorCode GetConnectedUnits(out List<string> units)
```

Parameters:

names	List of unit names

Return value:
Error code indicating success or failure.

13.4 Events

Apart from enabling the subscription in the system controller by calling to corresponding subscription method, the application needs to attach an event handler to the event. The following example shows how to handle events.

NOTE: Make sure to attach the event handler before enabling the subscription as enabling a subscription will trigger an initial fetch of the data from the system controller.

```
static void Main(string[] args)
{
    OpenInterfaceNetClient client = new OpenInterfaceNetClient();

    string ip = "192.168.53.100";
    string username = "user";
    string password = "password";

    TOIErrorCode ec = client.Connect(ip, username, password);
    if (ec != TOIErrorCode.OIERROR_OK)
    {
        return;
    }

    List<string> resources = new List<string>();
    resources.Add("Zone 1");
    resources.Add("Zone 2");

    // Attach the event handler to the event
    client.BgmVolumeChanged += OnBgmVolumeChanged;

    // Set the subscription in the system controller
    client.SetSubscriptionBgmVolume(true /*bSub*/, resources);
}

private static void OnBgmVolumeChanged(object sender,
      OIBgmVolumeChangedEventArgs e)
{
    // Handle the event
}
```

13.4.1 ConnectionBroken

Will be called when the connection with the system controller is broken (closed by other means than `Disconnect()`). When this function is called, it is necessary to make a new connection and set all subscriptions again.

```
event EventHandler ConnectionBroken;
```

Note that this event is also triggered when the PRAESENSA System detects a message transmission buffer overflow due to too slow reception by the application.

13.4.2 CallStateChanged

Indicates the state of a call has changed.

```
event EventHandler<OICallStateChangedEventArgs> CallStateChanged
```

Where the event argument `OICallStateChangedEventArgs` consists of:

CallId	Unique identification of the call.

State	State of the call. See §13.2.6 for the definitions of the call states.

13.4.3 ResourceStateChanged

Indication that a resource state has changed.

```
event EventHandler<OIResourceStateEventArgs> ResourceStateChanged
```

Where the event argument `OIResourceStateEventArgs` consists of:

Resources	List of names of zone groups, zones and/or control outputs.
CallId	Unique identification of the call (when state is activated).
Priority	Priority of the call using the resource (when state is activated). See §13.2.5 for the priority value ranges.
State	State of the resource. See §13.2.6 for the definitions of the resource states.

Note that when a zone-group is partial occupied by a call, the state of that zone-group is marked as occupied (OIRS_INUSE). Only when all zones in the zone-group are free, the state of the zone-group is marked free (OIRS_FREE).

13.4.4 ResourceFaultStateChanged

Indicates that a resource state has changed.

```
event EventHandler<OIResourceFaultStateEventArgs>
      ResourceFaultStateChanged
```

Where the event argument `OIResourceFaultStateEventArgs` consists of:

Resources	List of names of zone groups and/or zones. A comma separates each name in the routing list.
FaultState	Fault state for faults that affect the audio distribution of these zone or zone group resources.

Note that when a zone-group has a fault in one of its zones then that zone group is marked as in fault (OIRS_FAULT).

13.4.5 BgmRoutingChanged

Indication that a BGM routing has changed.

```
event EventHandler<OIBgmRoutingChangedEventArgs> BgmRoutingChanged
```

Where the event argument `OIBgmRoutingChangedEventArgs` consists of:

Channel	Name of the BGM channel.
Routing	List of names of zone groups, zones and/or control outputs.
Added	Whether the routing was added (true) or removed (false).

13.4.6 BgmVolumeChanged

Indication that BGM volume has changed.

```
event EventHandler<OIBgmVolumeChangedEventArgs> BgmVolumeChanged
```

Where the event argument `OIBgmVolumeChangedEventArgs` consists of:

Routing	Name of the BGM zone.
Volume	The new volume of the zone.

13.4.7 AlarmUpdate

Indication of alarm state change.

```
event EventHandler<OIAlarmStateChangedEventArgs> AlarmUpdate
```

Where the event argument `OIAlarmStateChangedEventArgs` consists of:

State	State of the fault alarm. See §13.2.4 for the definitions of the fault alarm states.

13.4.8 UnitCountChanged

Indication of unit count change

```
event EventHandler<OIUnitCountChangedEventArgs> UnitCountChanged
```

Where the event argument `OIUnitCountChangedEventArgs` consists of:

UnitCount	The number of connected units

13.4.9 DiagEventNotification

Will be called when a diagnostic event is logged inside the system controller. See section I for use of the diagEvent.

```
event EventHandler<OIDiagEventEventArgs> DiagEventNotification
```

Where the event argument `OIDiagEventEventArgs` consists of:

ActionType	Action done on the event. See §13.2.17 for the definitions of the action types.
Event	Reference to a `DiagEvent` object.

13.4.10 VirtualControlInputStateChanged

Will be called when the state of one or more virtual control inputs changes.

```
event EventHandler<OIVirtualControlInputStateChangedEventArgs>
      VirtualControlInputStateChanged
```

Where the event argument `OIVirtualControlInputStateChangedEventArgs` consists of:

VirtualControlInputs	List of names of virtual control inputs that have changed state. A comma separates each name in the list.
State	State of the virtual control inputs. See §13.2.12 for the definitions of the virtual input contact states.

13.5 DiagEvent Classes

The following chapter describes the derived `DiagEvent` classes and corresponding properties available. The derived `DiagEvent` classes are annotated with the `ClassIdAttribute`. This attribute contains the `ClassId` which corresponds with the `TOIDiagEventType` described in §13.2.13. The ClassId should be used to cast the `DiagEvent` base class to the correct derived class, as shown in the example below.

```
private static void OnDiagEventNotification(object sender,
      OIDiagEventEventArgs e)
{
    DiagEvent diagEvent = e.Event;
    TOIDiagEventType eventType = (TOIDiagEventType)diagEvent.ClassId;

    switch (eventType)
    {
        case TOIDiagEventType.OIDET_UserInjectedFault:
            DET_UserInjectedFault userInjectedFaultDiagEvent =
      (DET_UserInjectedFault)diagEvent;
            string description =
      userInjectedFaultDiagEvent.ErrorDescription;
            break;
        default:
            break;
    }
}
```

13.5.1 DiagEvent

Base class for all DiagEvents.

Appendix A. Properties:

TOIDiagEventGroup EventGroup	The group this event belongs to. See §13.2.14 for the definitions of the event groups.
uint EventId	Unique identification id.
DateTime AddTimeStamp	Time the event was created.
DateTime AcknowledgeTimeStamp	Time the event was acknowledged.
DateTime ResolveTimeStamp	Time the event was resolved.
DateTime ResetTimeStamp	Time the event was reset.
EventOriginator AddEventOriginator	The originator that added the event. See §**Error! Reference source not found.** for the class definitions.
EventOriginator AcknowledgeEventOriginator	The originator that acknowledged the event. See §**Error! Reference source not found.** for the class definitions.
EventOriginator ResolveEventOriginator	The originator that resolved the event. See §**Error! Reference source not found.** for the class definitions.
EventOriginator ResetOriginator	The originator that reset the event. See §**Error! Reference source not found.** for the class definitions.

13.5.2 GeneralEvent

Base class for all general events. Derived from `DiagEvent`. This class is empty and does not have any properties.

13.5.2.1 DET_EvacAcknowledge

ClassId: TOIDiagEventType.OIDET_EvacAcknowledge.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.2 DET_EvacReset

ClassId: TOIDiagEventType.OIDET_EvacReset.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.3 DET_EvacSet

ClassId: TOIDiagEventType.OIDET_EvacSet.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.4 DET_UnitConnect

ClassId: TOIDiagEventType.OIDET_UnitConnect.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.5 DET_DemoteToBackup

ClassId: TOIDiagEventType.OIDET_DemoteToBackup.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.6 DET_SCStartup

ClassId: TOIDiagEventType.OIDET_SCStartup.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.7 DET_OpenInterfaceConnect

ClassId: TOIDiagEventType.OIDET_OpenInterfaceConnect.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.8 DET_OpenInterfaceDisconnect

ClassId: TOIDiagEventType.OIDET_OpenInterfaceDisconnect.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.9 DET_OpenInterfaceConnectFailed

ClassId: TOIDiagEventType.OIDET_OpenInterfaceConnectFailed.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.10 DET_CallLoggingSuspended

ClassId: TOIDiagEventType.OIDET_CallLoggingSuspended.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.11 DET_CallLoggingResumed

ClassId: TOIDiagEventType.OIDET_CallLoggingResumed.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.12 DET_UserLogIn

ClassId: TOIDiagEventType.OIDET_UserLogIn.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.13 DET_UserLogOut

ClassId: TOIDiagEventType.OIDET_UserLogOut.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.14 DET_UserLogInFailed

ClassId: TOIDiagEventType.OIDET_UserLogInFailed.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.15 DET_BackupPowerModeStart

ClassId: TOIDiagEventType.OIDET_BackupPowerModeStart.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.16 DET_BackupPowerModeEnd

ClassId: TOIDiagEventType.OIDET_BackupPowerModeEnd.

Derived from `GeneralEvent`. This class is empty and does not have any properties.

13.5.2.17 DET_ConfigurationRestored

ClassId: TOIDiagEventType.OIDET_ConfiguredRestored.

Derived from `GeneralEvent`.

Appendix B. Properties:

bool ConfigurationSettingsRestored	Indication if configuration settings are restored
bool SecuritySettingsRestored	Indication if security settings are restored.
bool MessagesRestored	Indication if messages are restored. Not supported (yet) in the PRAESENSA system.

13.5.3 CallDiagEventV2

Base class for all call events. Derived from `DiagEvent`.

Appendix C. Properties:

uint CallId	Id of the call.

13.5.3.1 DET_CallStartDiagEventV2

ClassId: TOIDiagEventType.OIDET_CallStartV2.

Derived from `CallDiagEventV2`.

Properties:

string AudioInput	Audio input of the call.
string StartChime	Configured start chime name.
string EndChime	Configured end chime name.
bool LiveSpeech	Indicates if live speech is used.
string MessageNames	Comma separated string of messages.
string OutputNames	Comma separated string of configured outputs.
uint Priority	Call priority.
uint MessageRepeat	Number of message repeats configured.
uint OriginalCallId	Call id of the original call in case this is a replay.
TOICallOutputHandling OutputHandling	Call output handling. See §13.2.18.
TOICallTiming CallTiming	Call timing. See §13.2.20.

13.5.3.2 DET_CallEndDiagEventV2

ClassId: TOIDiagEventType.OIDET_CallEndV2.

Derived from `CallComplete`.

Properties:

TOICallState StateCompleted	Call state at which the call has ended. See §13.2.6.
bool Aborted	Indicates if the call was aborted.
TOICallStopReason StopReason	Reason the call was stopped. See §13.2.7.

13.5.3.3 DET_CallChangeResourceDiagEventV2

ClassId: TOIDiagEventType.OIDET_CallChangeResourceV2.

Derived from `CallDiagEventV2`.

Properties:

string Removed	Comma separated string of removed resources.
string Added	Comma separated string of added resources.

13.5.3.4 DET_CallTimeoutDiagEventV2

ClassId: TOIDiagEventType.OIDET_CallTimeoutV2.

Derived from `CallDiagEventV2`.

Properties:

string Unreached	Comma separated string of unreached resources.

13.5.3.5 DET_CallRestartDiagEvent

ClassId: TOIDiagEventType.OIDET_CallRestart.

Derived from `DET_CallStartDiagEventV2`.

Properties:

string Unreached	Comma separated string of unreached resources.

13.5.3.6 DET_CallResetDiagEvent

ClassId: TOIDiagEventType.OIDET_CallReset.

Derived from `CallComplete`.

Properties:

TOICallState StateCompleted	Call state at which the call has ended. See §13.2.6.
TOICallResetReason	Reason the call is reset. See §13.2.8.

13.5.4 FaultEvent

Base class for all fault events. Derived from `DiagEvent`. This class is empty and does not have any properties.

13.5.4.1 DET_AudioPathSupervision

ClassId: TOIDiagEventType.OIDET_AudioPathSupervision.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.2 DET_MicrophoneSupervision

ClassId: TOIDiagEventType.OIDET_MicrophoneSupervision.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.3 DET_SystemInputContact

ClassId: TOIDiagEventType.OIDET_SystemInputContact.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.4 DET_CallStationExtension

ClassId: TOIDiagEventType.OIDET_CallStationExtension.

Derived from `FaultEvent`.

Properties:

uint NumberConfigured	Number of configured extensions.
uint NumberConnected	Number of connected extensions.

13.5.4.5 DET_ConfigurationFile

ClassId: TOIDiagEventType.OIDET_ConfigurationFile.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.6 DET_ConfigurationVersion

ClassId: TOIDiagEventType.OIDET_ConfigurationVersion.

Derived from `FaultEvent`.

Properties:

string Expected	Expected configuration version.
string Loaded	Loaded configuration version.

13.5.4.7 DET_IllegalConfiguration

ClassId: TOIDiagEventType.OIDET_IllegalConfiguration.

Derived from `FaultEvent`.

Properties:

uint ErrorCode	Code of the illegal configuration error. Not used at the moment, currently filled with the value ‘0’.

13.5.4.8 DET_PrerecordedMessagesNames

ClassId: TOIDiagEventType.OIDET_PrerecordedMessagesNames.

Derived from `FaultEvent`.

Properties:

string MissingNames	Comma separated string with the missing messages.

13.5.4.9 DET_PrerecordedMessagesCorrupt

ClassId: TOIDiagEventType.OIDET_PrerecordedMessagesCorrupt.

Derived from `FaultEvent`.

Properties:

string CorruptNames	Comma separated string with the corrupt messages.

13.5.4.10 DET_UnitMissing

ClassId: TOIDiagEventType.OIDET_UnitMissing.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.11 DET_UnitReset

ClassId: TOIDiagEventType.OIDET_UnitReset.

Derived from `FaultEvent`.

Properties:

string ChipType	Processor type which caused the unit reset.

13.5.4.12 DET_UserInjectedFault

ClassId: TOIDiagEventType.OIDET_UserInjectedFault.

Properties:

string ErrorDescription	User injected error description.

13.5.4.13 DET_NoFaults

ClassId: TOIDiagEventType.OIDET_NoFaults.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.14 DET_ZoneLineFault

ClassId: TOIDiagEventType.OIDET_ZoneLineFault.

Derived from `FaultEvent`.

Properties:

string ZoneNames	Comma separated string with zone names for which the fault occurred.
string ControlInputName	Control input name which is configured for the zone line fault.

13.5.4.15 DET_NetworkChange

ClassId: TOIDiagEventType.OIDET_NetworkChangeDiagEvent.

Derived from `FaultEvent`.

Properties:

uint NrOfNetworkChanges	Number of network changes.
TNetworkChangeData[] NetworkChanges	Detailed information about network changes.

Where the `TNetworkChangeData` consists of:

string LocalPortId	The port ID of the local device.
string LocalSystemName	The name of the local device as configured in the PRAESENSA system.
string RemotelPortId	The port ID of the remote device.
string RemoteSystemName	The name of the remote device as configured in the PRAESENSA system.

13.5.4.16 DET_IncompatibleFirmware

ClassId: TOIDiagEventType.OIDET_IncompatibleFirmware.

Derived from `FaultEvent`.

Properties:

string Current	Current firmware in the device.
string Expected	Expected firmware the device should contain.

13.5.4.17 DET_Amp48VAFault

ClassId: TOIDiagEventType.OIDET_Amp48VAFault.

Derived from `FaultEvent`.

Properties:

Severity Severity	Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.18 DET_Amp48VBFault

ClassId: TOIDiagEventType.OIDET_Amp48VBFault.

Derived from `FaultEvent`.

Properties:

Severity Severity	Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.19 DET_AmpChannelFault

ClassId: TOIDiagEventType.OIDET_AmpChannelFault.

Derived from `FaultEvent`.

Properties:

Severity Severity	Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.20 DET_AmpShortCircuitLineAFault

ClassId: TOIDiagEventType.OIDET_AmpShortCircuitLineAFault.

Derived from `FaultEvent`.

Properties:

Severity Severity	Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.21 DET_AmpShortCircuitLineBFault

ClassId: TOIDiagEventType.OIDET_AmpShortCircuitLineBFault.

Derived from `FaultEvent`.

Properties:

Severity Severity	Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.22 DET_AmpAcc18VFault

ClassId: TOIDiagEventType.OIDET_AmpAcc18VFault.

Derived from `FaultEvent`.

Properties:

Severity Severity	Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.23 DET_AmpSpareInternalFault

ClassId: TOIDiagEventType.OIDET_AmpSpareInternalFault.

Derived from `FaultEvent`.

Properties:

Severity Severity	Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.24 DET_AmpChannelOverloadFault

ClassId: TOIDiagEventType.OIDET_AmpChannelOverloadFault.

Derived from `FaultEvent`.

Properties:

Severity Severity	Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.25 DET_EolFailureLineAFault

ClassId: TOIDiagEventType.OIDET_EolFailureLineAFault.

Derived from `FaultEvent`.

Properties:

Severity Severity — Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.26 DET_EolFailureLineBFault

ClassId: TOIDiagEventType.OIDET_EolFailureLineBFault.

Derived from `FaultEvent`.

Properties:

Severity Severity — Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.27 DET_GroundShortFault

ClassId: TOIDiagEventType.OIDET_GroundShortFault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.28 DET_OverheatFault

ClassId: TOIDiagEventType.OIDET_OverheatFault.

Derived from `FaultEvent`.

Properties:

Severity Severity — Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.29 DET_PowerMainsSupply

ClassId: TOIDiagEventType.OIDET_PowerMainsSupply.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.30 DET_PowerBackupSupply

ClassId: TOIDiagEventType.OIDET_PowerBackupSupply.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.31 DET_MainsAbsentPSU1Fault

ClassId: TOIDiagEventType.OIDET_MainsAbsentPSU1Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.32 DET_MainsAbsentPSU2Fault

ClassId: TOIDiagEventType.OIDET_MainsAbsentPSU2Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.33 DET_MainsAbsentPSU3Fault

ClassId: TOIDiagEventType.OIDET_MainsAbsentPSU3Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.34 DET_BackupAbsentPSU1Fault

ClassId: TOIDiagEventType.OIDET_BackupAbsentPSU1Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.35 DET_BackupAbsentPSU2Fault

ClassId: TOIDiagEventType.OIDET_BackupAbsentPSU2Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.36 DET_BackupAbsentPSU3Fault

ClassId: TOIDiagEventType.OIDET_BackupAbsentPSU3Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.37 DET_DcOut1PSU1Fault

ClassId: TOIDiagEventType.OIDET_DcOut1PSU1Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.38 DET_DcOut2PSU1Fault

ClassId: TOIDiagEventType.OIDET_DcOut2PSU1Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.39 DET_DcOut1PSU2Fault

ClassId: TOIDiagEventType.OIDET_DcOut1PSU2Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.40 DET_DcOut2PSU2Fault

ClassId: TOIDiagEventType.OIDET_DcOut2PSU2Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.41 DET_DcOut1PSU3Fault

ClassId: TOIDiagEventType.OIDET_DcOut1PSU3Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.42 DET_DcOut2PSU3Fault

ClassId: TOIDiagEventType.OIDET_DcOut2PSU3Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.43 DET_AudioLifelinePSU1Fault

ClassId: TOIDiagEventType.OIDET_AudioLifelinePSU1Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.44 DET_AudioLifelinePSU2Fault

ClassId: TOIDiagEventType.OIDET_AudioLifelinePSU2Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.45 DET_AudioLifelinePSU3Fault

ClassId: TOIDiagEventType.OIDET_AudioLifelinePSU3Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.46 DET_AccSupplyPSU1Fault

ClassId: TOIDiagEventType.OIDET_AccSupplyPSU1Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.47 DET_AccSupplyPSU2Fault

ClassId: TOIDiagEventType.OIDET_AccSupplyPSU2Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.48 DET_AccSupplyPSU3Fault

ClassId: TOIDiagEventType.OIDET_AccSupplyPSU3Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.49 DET_Fan1Fault

ClassId: TOIDiagEventType.OIDET_Fan1Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.50 DET_Fan2Fault

ClassId: TOIDiagEventType.OIDET_Fan2Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.51 DET_DcAux1Fault

ClassId: TOIDiagEventType.OIDET_DcAux1Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.52 DET_DcAux2Fault

ClassId: TOIDiagEventType.OIDET_DcAux2Fault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.53 DET_BatteryShortFault

ClassId: TOIDiagEventType.OIDET_BatteryShortFault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.54 DET_BatteryRiFault

ClassId: TOIDiagEventType.OIDET_BatteryRiFault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.55 DET_BatteryOverheatFault

ClassId: TOIDiagEventType.OIDET_BatteryOverheatFault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.56 DET_BatteryFloatChargeFault

ClassId: TOIDiagEventType.OIDET_BatteryFloatChargeFault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.57 DET_MainsAbsentChargerFault

ClassId: TOIDiagEventType.OIDET_MainsAbsentChargerFault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.58 DET_PoESupplyFault

ClassId: TOIDiagEventType.OIDET_PoESupplyFault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.59 DET_PowerSupplyAFault

ClassId: TOIDiagEventType.OIDET_PowerSupplyAFault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.60 DET_PowerSupplyBFault

ClassId: TOIDiagEventType.OIDET_PowerSupplyBFault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.61 DET_ExternalPowerFault

ClassId: TOIDiagEventType.OIDET_ExternalPowerFault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.62 DET_ChargerSupplyVoltageTooLowFault

ClassId: TOIDiagEventType.OIDET_ChargerSupplyVoltageTooLowFault

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.63 DET_BatteryOvervoltageFault

ClassId: TOIDiagEventType.OIDET_BatteryOvervoltageFault

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.64 DET_BatteryUndervoltageFault

ClassId: TOIDiagEventType.OIDET_BatteryUndervoltageFault

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.65 DET_MediaClockFault

ClassId: TOIDiagEventType.OIDET_MediaClockFault

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.66 DET_ChargerFault

ClassId: TOIDiagEventType.OIDET_ChargerFault

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.5.4.67 DET_Amp20VFault

ClassId: TOIDiagEventType.OIDET_Amp20VFault.

Derived from `FaultEvent`.

Properties:

Severity Severity	Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.68 DET_AmpPsuFault

ClassId: TOIDiagEventType.OIDET_AmpPsuFault.

Derived from `FaultEvent`.

Properties:

Severity Severity	Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.69 DET_NetworkLatencyFault

ClassId: TOIDiagEventType.OIDET_NetworkLatencyFault.

Derived from `FaultEvent`.

Properties:

Severity Severity	Severity of the fault. LOW = 0, HIGH = 1.

13.5.4.70 DET_SynchronizationFault

classId: TOIDiagEventType.OIDET_SynchronizationFault.

Derived from `FaultEvent`. This class is empty and does not have any properties.

13.6 EventOriginator classes

The following chapter describes the derived `EventOriginator` classes and corresponding properties available. The derived `DiagEvent` classes are annotated with the `ClassIdAttribute`. This attribute contains the `ClassId` which corresponds with the

`TOIEventOriginatorType` described in §13.2.15. The ClassId should be used to cast the `EventOriginator` base class to the correct derived class, as shown in the example below.

```
private static void OnDiagEventNotification(object sender,
      OIDiagEventEventArgs e)
{
    DiagEvent diagEvent = e.Event;
    EventOriginator addEventOriginator = diagEvent.AddEventOriginator;

    TOIEventOriginatorType originatorType =
      (TOIEventOriginatorType)addEventOriginator.ClassId;
    switch (originatorType)
    {
        case TOIEventOriginatorType.OIEOT_UserEventOriginator:
            UserEventOriginator userEventOriginator =
      (UserEventOriginator)addEventOriginator;
            string unitName = userEventOriginator.UnitName;
            break;
    }
}
```

13.6.1 EventOriginator

Base class for all EventOriginators. This class is empty and does not have any properties

13.6.1.1 NoEventOriginator

ClassId: OIEOT_NoEventOriginator. Event originator indicating there is no event originator.

Derived from `EventOriginator`. This class is empty and does not have any properties.

13.6.1.2 UnitEventOriginator

ClassId: OIEOT_UnitEventOriginator. Event originator indicating the event is originated by a unit.

Derived from `EventOriginator`.

Properties:

string UnitName	Name of the unit.

13.6.1.3 OpenInterfaceEventOriginator

ClassId: OIEOT_OpenInterfaceEventOriginator. Event originator indicating the event is originated by an open interface client.

Derived from `EventOriginator`.

Properties:

string TcpIpDeviceName	Device name as configured in the PRAESENSA system.
uint IpAddress	IP address of the client, in the form of “127.0.0.1”.
ushort PortNumber	TCP-port number of the device on the network.
string Username	User name of the client logged into the PRAESENSA system.

13.6.1.4 ControlInputEventOriginator

ClassId: OIEOT_ControlInputEventOriginator. Event originator indicating the event is originated by a control input on a unit.

Derived from `EventOriginator`.

Properties:

string OriginatorName	Name of the control input as configured in the PRAESENSA system.

13.6.1.5 AudioOutputEventOriginator

ClassId: OIEOT_AudioOutputEventOriginator. Event originator indicating the event is originated by an audio output on a unit.

Derived from `EventOriginator`. This class is empty and does not have any properties.

13.6.1.6 AudioInputEventOriginator

ClassId: OIEOT_AudioInputEventOriginator. Event originator indicating the event is originated by an audio input on a unit.

Derived from `EventOriginator`. This class is empty and does not have any properties.

13.6.1.7 UserEventOriginator

ClassId: OIEOT_UserEventOriginator. Event originator indicating a user action performed on the system.

Derived from `EventOriginator`.

Properties:

string OriginatorName	Name of the user as configured in the PRAESENSA system.

13.6.1.8 NetworkEventOriginator

ClassId: OIEOT_NetworkEventOriginator. Event originator indicating the event is originated by a network connection. Used for user login events.

Derived from `EventOriginator`.

Properties:

uint IpAddress	IP address of the client, in the form of “127.0.0.1”.
ushort PortNumber	TCP-port number of the device on the network.
string Username	The user name of the originator network connection.

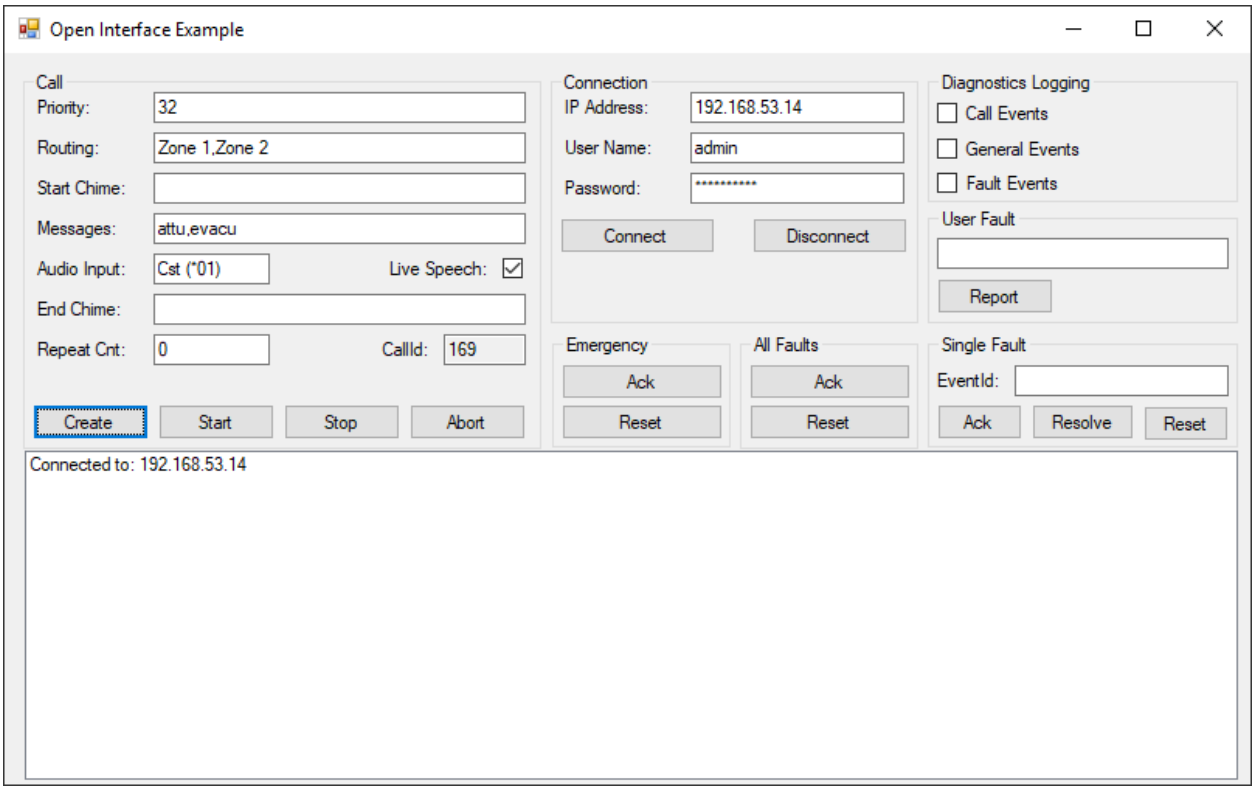
14 EXAMPLES

I. Interface usage

In the example code below a simple C# .NET application is pre-coded, containing a single form, where a subset of the functions mentioned above or present. This example could help you as a starting point for application development based on the Open Interface .NET library

Most of the fields used in the form have self-explaining names.

Form layout



Form Code

```
using Bosch.PRAESENSA.OpenInterface;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace OpenInterfaceNetExample
{
    /// <summary>
    /// This class implements an example with which it is possible
    /// to execute some actions in a PRAESENSA system using the Open Interface.
    /// It also shows how events generated by the system can be monitored.
    /// Only a part of the available functionality is used. Other functionality
    /// can be used in similar ways as shown in this example.
    /// This example uses the PRAESENSA Open Interface .NET library.
    /// There is only limited error checking.
    /// </summary>
    /// <remarks>This is only an example! Do not use it in real systems.</remarks>
    public partial class OILExample : Form
    {
        private OpenInterfaceNetClient m client;

        public OILExample()
        {
            InitializeComponent();

            Console.SetOut(new ListBoxWriter(lb Console));

            // Instantiate the OpenInterfaceNetClient and add the event handlers
            m_client = new OpenInterfaceNetClient();
```

```
            m client.ConnectionBroken += OnConnectionBroken;
            m client.DiagEventNotification += OnDiagEventNotification;
        }

        /// <summary>
        /// Function that is called when the Open Interface connection between the
       client
        /// and the system controller is broken.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void OnConnectionBroken(object sender, EventArgs e)
        {
            Console.WriteLine("Connect broken reported (remote)");
        }

        /// <summary>
        /// Function that is called when a new diagnostic event is logged in the
       PRAESENSA system.
        /// In order for this function to be called the client must be connected to the
       system controller
        /// and subscribed to at least one event group (call, general or fault).
        /// </summary>
        /// <param name="sender">Indicates who raised the event..</param>
        /// <param name="e">Diagnostic event logged in the system.</param>
        private void OnDiagEventNotification(object sender, OIDiagEventEventArgs e)
        {
            Console.WriteLine("EventId = {0}, action = {1}, event = {2}",
       e.Event.EventId, e.ActionType, e.Event.ToString());
        }

        /// <summary>
        /// Function that is called when the connect button is clicked.
        /// It will connect the client to the system controller using the parameters
        /// defined by the user. Only when connected can other functionality be used.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn_Connect_Click(object sender, EventArgs e)
        {
            TOIErrorCode ec = m client.Connect(tb IpAddress.Text, tb UserName.Text,
       tb Password.Text, true /*secure*/);
            if (ec == TOIErrorCode.OIERROR OK)
            {
                Console.WriteLine("Connected to: {0}", tb_IpAddress.Text);
            }
            else
            {
                Console.WriteLine("Failed to connect: {0}", ec);
            }
        }

        /// <summary>
        /// Function that is called when the disconnect button is clicked.
        /// It will disconnect the client from the system controller.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn Disconnect Click(object sender, EventArgs e)
        {
            TOIErrorCode ec = m_client.Disconnect();
            if (ec == TOIErrorCode.OIERROR_OK)
            {
                Console.WriteLine("Disconnected");
            }
            else
            {
                Console.WriteLine("Failed to disconnect: {0}", ec);
            }
        }

        /// <summary>
        /// Function that is called when the call create button is clicked.
        /// It will create a call using the parameters defined by the user and
        /// the returned call ID is stored.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn_CallCreate_Click(object sender, EventArgs e)
        {
```

```
            uint callId = 0;
            List<string> routing = tb Routing.Text.Split(new char[] { ',' }).ToList();
            List<string> messages = tb Messages.Text.Split(new char[] { ',' }).ToList();

            TOIErrorCode ec = m_client.CreateCallEx2(routing,
                                                     Convert.ToUInt32(tb Priority.Text),

        TOICallOutputHandling.OICOH PARTIAL,

        TOICallStackingMode.OICSM_WAIT_FOR_ALL,
                                                     0,
                                                     tb StartChime.Text,
                                                     tb EndChime.Text,
                                                     cb LiveSpeech.Checked,
                                                     tb AudioInput.Text,
                                                     messages,

        Convert.ToUInt32(tb RepeatCnt.Text),
                                                     TOICallTiming.OICTM IMMEDIATE,
                                                     "",
                                                     0, 0, 0, 0,
                                                     out callId);

            if (ec == TOIErrorCode.OIERROR OK)
            {
                tb_CallId.Text = callId.ToString();
            }
            else
            {
                Console.WriteLine("Failed to create call: {0}", ec);
            }
        }

        /// <summary>
        /// Function that is called when the call start button is clicked.
        /// This will start the call that is created earlier.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn_CallStart_Click(object sender, EventArgs e)
        {
            uint callId = Convert.ToUInt32(tb CallId.Text);

            TOIErrorCode ec = m_client.StartCreatedCall(callId);
            if (ec != TOIErrorCode.OIERROR_OK)
            {
                Console.WriteLine("Failed to start call: {0}", ec);
            }
        }

        /// <summary>
        /// Function that is called when the call stop button is clicked.
        /// This will stop the call that was started earlier.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn_CallStop_Click(object sender, EventArgs e)
        {
            uint callId = Convert.ToUInt32(tb CallId.Text);

            TOIErrorCode ec = m_client.StopCall(callId);
            if (ec != TOIErrorCode.OIERROR_OK)
            {
                Console.WriteLine("Failed to stop call: {0}", ec);
            }

            tb_CallId.Text = "";
        }

        /// <summary>
        /// Function that is called when the call abort button is clicked.
        /// This will abort the call that was started earlier.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn CallAbort Click(object sender, EventArgs e)
        {
            uint callId = Convert.ToUInt32(tb_CallId.Text);

            TOIErrorCode ec = m_client.AbortCall(callId);
```

```
            if (ec != TOIErrorCode.OIERROR OK)
            {
                Console.WriteLine("Failed to abort call: {0}", ec);
            }

            tb CallId.Text = "";
        }

        /// <summary>
        /// Function that is called when the emergency acknowledge button is clicked.
        /// This will acknowledge the evac alarm (if present).
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn EmergencyAck Click(object sender, EventArgs e)
        {
            TOIErrorCode ec = m_client.AckEvacAlarm();
            if (ec != TOIErrorCode.OIERROR OK)
            {
                Console.WriteLine("Failed to acknowledge evac alarm: {0}", ec);
            }
        }

        /// <summary>
        /// Function that is called when the emergency reset button is clicked.
        /// This will reset the evac alarm and abort all running evac calls.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn EmergencyReset Click(object sender, EventArgs e)
        {
            TOIErrorCode ec = m_client.ResetEvacAlarmEx(true /*bAbortEvacCalls*/);
            if (ec != TOIErrorCode.OIERROR_OK)
            {
                Console.WriteLine("Failed to reset evac alarm: {0}", ec);
            }
        }

        /// <summary>
        /// Function that is called when the all faults acknowledge button is clicked.
        /// This will acknowledge all fault events in the system.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn_AllFaultsAck_Click(object sender, EventArgs e)
        {
            TOIErrorCode ec = m client.AckAllFaults();
            if (ec != TOIErrorCode.OIERROR OK)
            {
                Console.WriteLine("Failed to acknowledge all faults: {0}", ec);
            }
        }

        /// <summary>
        /// Function that is called when the all faults reset button is clicked.
        /// This will reset all fault events in the system.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn AllFaultsReset Click(object sender, EventArgs e)
        {
            TOIErrorCode ec = m_client.ResetAllFaults();
            if (ec != TOIErrorCode.OIERROR OK)
            {
                Console.WriteLine("Failed to reset all faults: {0}", ec);
            }
        }

        /// <summary>
        /// Function that is called when the report fault button is clicked.
        /// This will trigger a UserInjectedFault event in the system with the
        /// description provided by the user.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn ReportFault Click(object sender, EventArgs e)
        {
            uint eventId = 0;

            TOIErrorCode ec = m_client.ReportFault(tb_UserFault.Text, out eventId);
```

```
            if (ec == TOIErrorCode.OIERROR OK)
            {
                Console.WriteLine("Fault reported, eventId: {0}", eventId);
            }
            else
            {
                Console.WriteLine("Failed to report fault: {0}", ec);
            }
        }

        /// <summary>
        /// Function that is called when the fault acknowledge button is clicked.
        /// This will acknowledge a single fault in the system.
        /// The user should provide the fault ID.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn FaultAck Click(object sender, EventArgs e)
        {
            TOIErrorCode ec = m client.AckFault(Convert.ToUInt32(tb EventId.Text));
            if (ec != TOIErrorCode.OIERROR OK)
            {
                Console.WriteLine("Failed to acknowledge fault: {0}", ec);
            }
        }

        /// <summary>
        /// Function that is called when the fault resolve button is clicked.
        /// This will resolve a single fault in the system.
        /// The user should provide the fault ID.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn_FaultResolve_Click(object sender, EventArgs e)
        {
            TOIErrorCode ec = m client.ResolveFault(Convert.ToUInt32(tb EventId.Text));
            if (ec != TOIErrorCode.OIERROR OK)
            {
                Console.WriteLine("Failed to resolve fault: {0}", ec);
            }
        }

        /// <summary>
        /// Function that is called when the fault reset button is clicked.
        /// This will reset a single fault in the system.
        /// The user should provide the fault ID.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void btn_FaultReset_Click(object sender, EventArgs e)
        {
            TOIErrorCode ec = m client.ResetFault(Convert.ToUInt32(tb EventId.Text));
            if (ec != TOIErrorCode.OIERROR OK)
            {
                Console.WriteLine("Failed to reset fault: {0}", ec);
            }
        }

        /// <summary>
        /// Function that is called when the call events checkbox is clicked.
        /// This will either subscribe or unsubscribe from call events.
        /// Call events are handled by <see cref="OnDiagEventNotification(object,
       OIDiagEventEventArgs)"/>.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void cb_CallEvents_CheckedChanged(object sender, EventArgs e)
        {
            TOIErrorCode ec = m client.SetSubscriptionEvents(cb CallEvents.Checked,
       TOIDiagEventGroup.OIDEG CALLEVENTGROUP);
            if (ec != TOIErrorCode.OIERROR OK)
            {
                Console.WriteLine("Failed to (un)subscribe to/from call events: {0}",
       ec);
            }
        }

        /// <summary>
        /// Function that is called when the general events checkbox is clicked.
        /// This will either subscribe or unsubscribe from general events.
```

```
        /// General events are handled by <see cref="OnDiagEventNotification(object,
       OIDiagEventEventArgs)"/>.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void cb GeneralEvents CheckedChanged(object sender, EventArgs e)
        {
            TOIErrorCode ec = m client.SetSubscriptionEvents(cb GeneralEvents.Checked,
       TOIDiagEventGroup.OIDEG GENERALEVENTGROUP);
            if (ec != TOIErrorCode.OIERROR_OK)
            {
                Console.WriteLine("Failed to (un)subscribe to/from general events: {0}",
       ec);
            }
        }

        /// <summary>
        /// Function that is called when the fault events checkbox is clicked.
        /// This will either subscribe or unsubscribe from fault events.
        /// Fault events are handled by <see cref="OnDiagEventNotification(object,
       OIDiagEventEventArgs)"/>.
        /// </summary>
        /// <param name="sender">Indicates who raised the event.</param>
        /// <param name="e">Parameters of the event (not used).</param>
        private void cb FaultEvents CheckedChanged(object sender, EventArgs e)
        {
            TOIErrorCode ec = m_client.SetSubscriptionEvents(cb_FaultEvents.Checked,
       TOIDiagEventGroup.OIDEG_FAULTEVENTGROUP);
            if (ec != TOIErrorCode.OIERROR OK)
            {
                Console.WriteLine("Failed to (un)subscribe to/from fault events: {0}",
       ec);
            }
        }
    }
}
```

Bosch Security Systems B.V.
Torenallee 49
5617 BA Eindhoven
Netherlands
www.boschsecurity.com