Praesideo 4.3

Digital Public Address and Emergency Sound System

BOSCH

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 disclaim all warranties with regard to the information provided in these instructions. In no event shall Bosch Security Systems 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.

Table of Contents

Part 1 - Open Interface Protocol

Intentionally left blank.

1 Introduction

1.1 Purpose

This part of the Open Interface Programming Instructions describes the open interface protocol of the Praesideo Public Address System. Praesideo 2.2 is the first version on which the open interface protocol is available. (You can obtain the version number, as application version, of Praesideo from the configuration menu of the network controller, refer to the Praesideo Installation and User Instructions.)

1.2 Scope

This part of the Open Interface Programming Instructions is intended for persons, who want to integrate Praesideo in their applications with the Praesideo native communication interface. They must have knowledge about:

- The Praesideo system and its installation (see the Praesideo Installation and User Instructions).
- The TCP/IP protocol and how to communicate using TCP/IP.

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

Note
A Praesideo network controller is able to communicate with up to five or seven (version 4.3 or later) Open Interface clients at the same time. This includes connection to Logging Servers.

1.3 Definitions, acronyms and abbreviations

table 1.1: Definitions, acronyms and abbreviations

Term	Description
NCO	Praesideo Network Controller
OI	Open Interface
OIP	Open Interface Protocol
LSB	Least Significant Byte
MSB	Most Significant Byte
WLS	Wireless line supervision, with: • WLS1: LBB4442 line supervision set • WLS2: LBB4440 supervision master, LBB4441 Loudspeaker supervision board, LBB4443 End-of-line supervision board

1.4 Structure

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 NCO 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. Besides 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 Praesideo is a public address and emergency sound 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 section 2.1.3)
- Call content (refer to section 2.1.4)
- Routing (refer to section 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.

table 2.1: Priorities

Priority	Call type
0 to 31	BGM (Background Music) calls. Always partial call, regardless the partiality setting (see section 2.1.6).
32 to 223	Business calls
224 to 255	Emergency calls. Always partial call, regardless the partiality setting.

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:

- Start chime (optional)
- Pre-recorded message(s) (optional)
- Live speech (optional)
- 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 section 2.1.6).

2.1.6 Output handling

Calls can be partial, non-partial or stacked. By definition, partial calls do not require the entire routing to be available at the start of the call and during the call. When a partial call is started and a part of the routing is not available, the call is only distributed to the available part of the routing. When a part of the routing becomes unavailable during the call, the call continues in the parts of the routing that are still available. Stacked calls have the ability to record and replay the call to routing parts that were not available during the original call.

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.

Note
BGM calls and emergency calls without live speech are started in the unavailable parts of the routing as soon as these parts are released.

Note
Stacked calls are only available within the business call priority range. This means that stacking emergency and BGM priority calls are not possible.

2.1.7 Identification by names

The routing and call content are identified by unique names. These names are entered with the configuration webpages during the installation of the system. Before version 3.1, it was not possible to retrieve these unique

names from the system via the Open Interface communication. Instead, some customers have been retrieving configuration information from the NCO via webpage content for use in an Open Interface client application. This function was never supported but no alternative other than manual input was available at that moment in the Open Interface.
However, from version 3.1 onwards retrieving this type of information is available in the Open Interface as supported functions for retrieving zone names, zone group names, message names, chime names, audio input names and BGM channel names.
In version 3.4 of Praesideo the web configuration pages have been redone completely to make the visual presentation and user interface less depending on the browser program and version. As a consequence of this update the data transfer between the browser and server in the NCO has changed. Normally that will not be a problem, but the former method of retrieving configuration information will not work anymore (without making changes) and is strongly advised against. Use the supported Open Interface commands instead, see section 4.40 through section 4.45.

2.2 Diagnostics

As Praesideo is an emergency compliant system, it monitors his equipment and signals activity performed on the system. Systems connected to a Praesideo System can subscribe to activity and equipment signals for long term storage and reporting facilities. To receive events from the Praesideo 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 Praesideo System are sent. Existing event 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 Praesideo System, the Praesideo System responds with a message with the action type **OIACT_EXISTING_LAST** and an event of type **DET_NoFaults**. 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 as described in section 9.3.9.

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

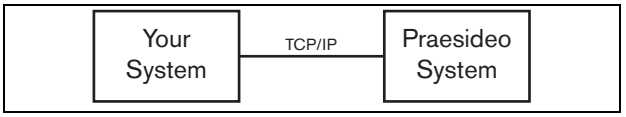

figure 2.1: Hardware connection

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

- Use the IP address of the network controller.
- Port number 9401.

3 Protocol considerations

3.1 Set-up a connection

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

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 Praesideo for verification. If either the user name or the password are incorrect, an error is reported back. In this case, the socket connection is disconnected on demand of the network controller. If the user name and password are correct, all control functions of Praesideo become available.

3.2 Heartbeat

After the connection between your system and Praesideo has been established, the network controller of Praesideo starts the heartbeat checks of your system. The network 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 network 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 Praesideo on your system. To signal that the connection is still present, you must transmit a **MESSAGETYPE_OIP_KeepAlive** message (refer to section 3.5) to the network controller every 5 seconds when no other messages are ready for transmission.

3.3 Response times

When your system sends a message to the network 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 Praesideo is based on messages. This section describes the structures that are used in the data field of the messages for Praesideo.

3.4.2 General Message Layout

Each message must have this layout:

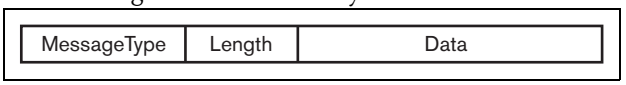

figure 3.1: General message layout

Defined in (c-style) structure format:

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

figure 3.2: Structure format

Where:

- **messageType**
  The **message-type**, which describes the content of the actual data passed. Refer to the various message-type definitions in sections below (see section 4, section 5 and section 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 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 format. Between the data-type is no alignment present. 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

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;
```

figure 3.3: 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 Praesideo 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.

```
struct {
   LONG    time;
} TIME;
```

figure 3.4: Time structure

Where:

- **time**
  Local 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;
```

figure 3.5: Complex structure

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 (section 7 and section 8).
- **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 Praesideo System do not accept spaces around the separation commas in lists of strings. However, notifications and results sent from the Praesideo 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 Praesideo 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 Praesideo System that your system is still alive. The Praesideo System also sends heartbeat messages to indicate that the Praesideo 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;
```

figure 3.6: MESSAGETYPE_OIP_KeepAlive

Where:

- **messageType**
  The message type indicator for the heartbeat message. Constant value **MESSAGETYPE_OIP_KeepAlive** (See section 9.5).
- **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 Praesideo 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;
```

figure 3.7: MESSAGETYPE_OIP_ResponseProtocolError

Where:

- **messageType**
  The message type indicator for the message. Constant value **MESSAGETYPE_OIP_ResponseProtocolError** (See section 9.5).
- **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 Praesideo System which is not conform the message guideline as described in section 3.4.

3.7 Buffer overflow

Purpose:

Messages ready for transmission from the Praesideo System are queued. In case the receive speed of the connected system is too low, the queue may overflow (dependant on the number of generated events, resource update, etc.). Since the queue consumes internal Praesideo 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 Praesideo System. Commands always result in a response from the Praesideo 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 Praesideo System can send notification messages.

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

figure 4.1: 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).

4.2 MESSAGETYPE_OIP_Login

Purpose:

Logs in on the Praesideo System with a user name and password.

Parameter structure:

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

figure 4.2: MESSAGETYPE_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 version number of the Praesideo System, which must be used to verify whether your system is connected with the expected version of the Praesideo System.

Parameter structure:

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

figure 4.3: MESSAGETYPE_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_ 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                   endChimeAttenuation;
   UINT                   messageAttenuation;
} OIP_CreateCallEx;
```

figure 4.4: MESSAGETYPE_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 section 9.3.10 for the value set description.
- **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 section 9.3.11 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 section 9.3.12 for the value set description.

- **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

4.5 MESSAGETYPE_OIP_CreateCallEx (before V3.4)

Purpose:

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

Parameter structure:

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

figure 4.5: MESSAGETYPE_OIP_CreateCallEx

Where:

- **header**
  Header of the message, where the messageType element is equal to **MESSAGETYPE_OIP_CreateCallEx.**
- **priority**
  Priority of the call.
  - **Range: 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 section 9.3.10 for the value set description.
- **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 section 9.3.11 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 excepted ranges 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 section 9.3.12 for the value set description.
- **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.**

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_CreateCall (before V3.1)

Purpose:

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

Parameter structure:

```
struct {
   COMMANDHEADER   header;
   UINT            priority;
   BOOLEAN         partial;
   BOOLEAN         liveSpeech;
   UINT            repeat;
   STRING          routing;
   STRING          startChime;
   STRING          endChime;
   STRING          audioInput;
   STRING          messages;
} OIP_CreateCall;
```

figure 4.6: MESSAGETYPE_OIP_CreateCall

Where:

- **header**
  Header of the message, where the messageType element is equal to **MESSAGETYPE_OIP_CreateCall**.
- **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.
- **partial**
  Whether or not the call is partial. A partial call is started even when not all specified zones are available. It also accepts extension and removal of zones to/from the routing. **TRUE** = partial, **FALSE** = not 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.

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;
```

figure 4.7: MESSAGETYPE_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 createCall (see section 4.4, section 4.5 and section 4.6). See section 9.3.2 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_StartCall

Purpose:

Starts a call with the given parameters. If the call was started successfully, call state update notification messages are sent.

Parameter structure:

```
struct {
   COMMANDHEADER    header;
   UINT             priority;
   BOOLEAN          partial;
   BOOLEAN          liveSpeech;
   UINT             repeat;
   STRING           routing;
   STRING           startChime;
   STRING           endChime;
   STRING           audioInput;
   STRING           messages;
} OIP_StartCall;
```

figure 4.8: MESSAGETYPE_OIP_StartCall

Where:

- **header**
  Header of the message, where the **messageType** element is equal to **MESSAGETYPE_OIP_StartCall**.
- **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.
- **partial**
  Whether or not the call is partial. A partial call is started even when not all specified zones are available. It also accepts extension and removal of zones to/from the routing.**TRUE** = partial, **FALSE** = not 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.

Response message type:
- **MESSAGETYPE_OIP_ResponseCallId.**

Update notifications:
- **MESSAGETYPE_OIP_NotifyCall**
- **MESSAGETYPE_OIP_NotifyResources**

Related messages:
- **MESSAGETYPE_OIP_StopCall**
- **MESSAGETYPE_OIP_AbortCall**
- **MESSAGETYPE_OIP_AddToCall**
- **MESSAGETYPE_OIP_RemoveFromCall**

4.9 MESSAGETYPE_OIP_StopCall

Purpose:
Stops a previously created or started call.

Parameter structure:

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

figure 4.9: MESSAGETYPE_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 **startCall** (section 4.4, section 4.5 and section 4.6). See section 9.3.2 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_StartCreatedCall**
- **MESSAGETYPE_OIP_StartCall**
- **MESSAGETYPE_OIP_AbortCall**
- **MESSAGETYPE_OIP_AddToCall**
- **MESSAGETYPE_OIP_RemoveFromCall**

4.10 MESSAGETYPE_OIP_ AbortCall

Purpose:

Aborts a previously created or started call.

Parameter structure:

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

figure 4.10: MESSAGETYPE_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 **startCall** (section 4.4, section 4.5 and section 4.6). See section 9.3.2 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_StartCreatedCall**
- **MESSAGETYPE_OIP_StartCall**
- **MESSAGETYPE_OIP_StopCall**
- **MESSAGETYPE_OIP_AddToCall**
- **MESSAGETYPE_OIP_RemoveFromCall**

4.11 MESSAGETYPE_OIP_ AddToCall

Purpose:

Adds routing to a previously created or started call.

Parameter structure:

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

figure 4.11: MESSAGETYPE_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 **startCall** (section 4.4, section 4.5 and section 4.6). See section 9.3.2 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_CreateCall**
- **MESSAGETYPE_OIP_StartCreatedCall**
- **MESSAGETYPE_OIP_StartCall**
- **MESSAGETYPE_OIP_StopCall**
- **MESSAGETYPE_OIP_AbortCall**
- **MESSAGETYPE_OIP_RemoveFromCall**

4.12 MESSAGETYPE_OIP_RemoveFromCall

Purpose:

Removes routing from a previously created or started call.

Parameter structure:

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

figure 4.12: MESSAGETYPE_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 **startCall** (section 4.4, section 4.5 and section 4.6). See section 9.3.2 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_CreateCall**
- **MESSAGETYPE_OIP_StartCreatedCall**
- **MESSAGETYPE_OIP_StartCall**
- **MESSAGETYPE_OIP_StopCall**
- **MESSAGETYPE_OIP_AbortCall**
- **MESSAGETYPE_OIP_AddToCall**

4.13 MESSAGETYPE_OIP_CancelAll

Purpose:

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

Parameter structure:

```
struct {
   COMMANDHEADER    header;
} OIP_CancelAll;
```

figure 4.13: MESSAGETYPE_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_CreateCall**
- **MESSAGETYPE_OIP_StartCreatedCall**
- **MESSAGETYPE_OIP_StartCall**
- **MESSAGETYPE_OIP_CancelLast**

4.14 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;
```

figure 4.14: MESSAGETYPE_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_CreateCall**
- **MESSAGETYPE_OIP_StartCreatedCall**
- **MESSAGETYPE_OIP_StartCall**
- **MESSAGETYPE_OIP_CancelAll**

4.15 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 section 6.3).

Parameter structure:

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

figure 4.15: MESSAGETYPE_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**

4.16 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 section 6.3).

Parameter structure:

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

figure 4.16: MESSAGETYPE_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**

4.17 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;
```

figure 4.17: MESSAGETYPE_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**

4.18 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 section 6.3).

Parameter structure:

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

figure 4.18: MESSAGETYPE_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 section 9.3.1 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**

4.19 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;
```

figure 4.19: MESSAGETYPE_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 section 9.3.1 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**

4.20 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 section 6.3).

Parameter structure:

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

figure 4.20: MESSAGETYPE_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 section 9.3.1 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**

4.21 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 section 4.34).

Parameter structure:

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

figure 4.21: MESSAGETYPE_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_ResetEvacAlarm**
- **MESSAGETYPE_OIP_ResetEvacAlarmEx**
- **MESSAGETYPE_OIP_StartCall** (when an emergency call is started)

4.22 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 section 4.34).

Parameter structure:

```
struct {
   COMMANDHEADER    header;
   BOOLEAN          bAbortEvacCalls
} OIP_ResetEvacAlarmEx;
```

figure 4.22: MESSAGETYPE_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**

4.23 MESSAGETYPE_OIP_ ResetEvacAlarm

Purpose:

Resets the emergency alarm. If the state of the emergency alarm changes, it results in the message **MESSAGETYPE_OIP_NotifyAlarm** (if subscribed, see section 4.34). Running evacuation priority calls will be aborted.

Parameter structure:

```
struct {
   COMMANDHEADER    header;
} OIP_ResetEvacAlarm;
```

figure 4.23: MESSAGETYPE_OIP_ResetEvacAlarm

Where:

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

Response message type:

- **MESSAGETYPE_OIP_Response**

Update notifications:

- **MESSAGETYPE_OIP_NotifyAlarm** (alarm-type equals **OIAT_EVAC**)

Related messages:

- **MESSAGETYPE_OIP_AckEvacAlarm**
- **MESSAGETYPE_OIP_ResetEvacAlarmEx**

4.24 MESSAGETYPE_OIP_ IncrementBgmVolume

Purpose:

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

Parameter structure:

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

figure 4.24: MESSAGETYPE_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**

4.25 MESSAGETYPE_OIP_ IncrementBgmChannelVolume

Purpose:

Increments the BGM volume of a channel with 3 dB.

Parameter structure:

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

figure 4.25: MESSAGETYPE_OIP_ IncrementBgmChannelVolume

Where:

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

Response message type:

- **MESSAGETYPE_OIP_Response**

Related messages:

- **MESSAGETYPE_OIP_DecrementBgmChannelVolume**
- **MESSAGETYPE_OIP_SetBgmVolume**
- **MESSAGETYPE_OIP_GetBgmChannelNames**

4.26 MESSAGETYPE_OIP_DecrementBgmVolume

Purpose:

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

Parameter structure:

```
Struct {
   COMMANDHEADER    header;
   STRING           routing;
} OIP_DecrementBgmVolume;
```

figure 4.26: MESSAGETYPE_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**

4.27 MESSAGETYPE_OIP_DecrementBgmChannelVolume

Purpose:

Decrements the BGM volume of a channel with 3 dB.

Parameter structure:

```
Struct {
   COMMANDHEADER    header;
   STRING           channel;
} OIP_DecrementBgmChannelVolume;
```

figure 4.27: MESSAGETYPE_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 Praesideo configuration.

Response message type:

- **MESSAGETYPE_OIP_Response**

Related messages:

- **MESSAGETYPE_OIP_IncrementBgmChannelVolume**
- **MESSAGETYPE_OIP_SetBgmVolume**
- **MESSAGETYPE_OIP_GetBgmChannelNames**

4.28 MESSAGETYPE_OIP_SetBgmVolume

Purpose:

Sets the BGM volume of the given routing.

Parameter structure:

```
struct {
   COMMANDHEADER   header;
   INT             volume;
   STRING          routing;
} OIP_SetBgmVolume;
```

figure 4.28: MESSAGETYPE_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**

4.29 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;
```

figure 4.29: MESSAGETYPE_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 Praesideo 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_SetBgmRouting**
- **MESSAGETYPE_OIP_ToggleBgmRouting**

4.30 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;
```

figure 4.30: MESSAGETYPE_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 Praesideo 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_SetBgmRouting**
- **MESSAGETYPE_OIP_ToggleBgmRouting**

4.31 MESSAGETYPE_OIP_ ToggleBgmRouting

Purpose:

Toggles a routing in a BGM channel. When none of names in the specified routing are part of 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;
```

figure 4.31: MESSAGETYPE_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 Praesideo 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**

4.32 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 Praesideo System.

Parameter structure:

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

figure 4.32: MESSAGETYPE_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 Praesideo 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_AddBgmRouting**
- **MESSAGETYPE_OIP_ToggleBgmRouting**

4.33 MESSAGETYPE_OIP_ SetDateAndTime

Purpose:

Sets the clock of the network controller of the Praesideo System to a new date and time.

Parameter structure:

```
Struct {
   COMMANDHEADER   header;
   UINT            year;
   UINT            month;
   UINT            day;
   UINT            hour;
   UINT            minute;
   UINT            second;
} OIP_SetDateAndTime;
```

figure 4.33: MESSAGETYPE_OIP_SetDateAndTime

Where:

- **header**
  Header of the message, where the messageType element is equal to **MESSAGETYPE_OIP_SetDateAndTime**.
- **year**
  The year of the new date, in the range 1970…2037
- **month**
  The month of the new date, month in the range 1...12.
- **day**
  The day of the new date, day in the range 1…31.
- **hour**
  The hour of the new time, hour in the range 0...23.
- **minute**
  The minute of the new time, minute in the range 0...59.
- **second**
  The second of the new time, second in the range 0...59.

Response message type:

- **MESSAGETYPE_OIP_Response**

4.34 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;
```

figure 4.34: MESSAGETYPE_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 section 9.3.3.
- **subscription**
  Whether to subscribe or unsubscribe. **TRUE** = subscribe, **FALSE** = unsubscribe

Response message type:

- **MESSAGETYPE_OIP_Response**

Update notifications:

- **MESSAGETYPE_OIP_NotifyAlarm**

4.35 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;
```

figure 4.35: MESSAGETYPE_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**

4.36 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;
```

figure 4.36: MESSAGETYPE_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**
  Whether to subscribe or unsubscribe. **TRUE** = subscribe, **FALSE** = unsubscribe.

Response message type:

- **MESSAGETYPE_OIP_Response**

Update notifications:

- **MESSAGETYPE_OIP_NotifyResourceFaultState**

4.37 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;
```

figure 4.37: MESSAGETYPE_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 Praesideo configuration.
- **subscription**
  Whether to subscribe or unsubscribe. **TRUE** = subscribe, **FALSE** = unsubscribe.

Response message type:

- **MESSAGETYPE_OIP_Response**

Update notifications:

- **MESSAGETYPE_OIP_NotifyBgmRouting**

4.38 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;
```

figure 4.38: MESSAGETYPE_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 section 9.4.2.
- **subscription**
  Whether to subscribe or unsubscribe. **TRUE** = subscribe, **FALSE** = unsubscribe.

Response message type:

- **MESSAGETYPE_OIP_Response**

Update notifications:

- **MESSAGETYPE_OIP_NotifyDiagEvent**

4.39 MESSAGETYPE_OIP_ SetSubscriptionBgmVolume

Purpose:

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

Parameter structure:

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

figure 4.39: MESSAGETYPE_OIP_ SetSubscriptionBgmVolume

Where:

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

Response message type:

- **MESSAGETYPE_OIP_Response**

Update notifications:

- **MESSAGETYPE_OIP_NotifyBgmVolume**

4.40 MESSAGETYPE_OIP_ GetZoneNames

Purpose:

Retrieve the configured zone names from the Praesideo 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;
```

figure 4.40: MESSAGETYPE_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**

4.41 MESSAGETYPE_OIP_ GetZoneGroupNames

Purpose:

Retrieve the configured zone group names from the Praesideo system.

Parameter structure:

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

figure 4.41: MESSAGETYPE_OIP_ GetZoneGroupNames

Where:

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

Response message type:

- **MESSAGETYPE_OIP_ResponseNames**

4.42 MESSAGETYPE_OIP_ GetMessageNames

Purpose:

Retrieve the configured message names from the Praesideo system.

Parameter structure:

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

figure 4.42: MESSAGETYPE_OIP_ GetMessageNames

Where:

- **header**
  Header of the message, where the **messageType** element is equal to **MESSAGETYPE_ OIP_GetMessageNames.**

Response message type:

- **MESSAGETYPE_OIP_ResponseNames**

4.43 MESSAGETYPE_OIP_ GetChimeNames

Purpose:

Retrieve the chime names from the Praesideo system.

Parameter structure:

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

figure 4.43: MESSAGETYPE_OIP_GetChimeNames

Where:

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

Response message type:

- **MESSAGETYPE_OIP_ResponseNames**

4.44 MESSAGETYPE_OIP_ GetAudioInputNames

Purpose:

Retrieve the configured audio input names from the Praesideo system.

Parameter structure:

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

figure 4.44: MESSAGETYPE_OIP_ GetAudioInputNames

Where:

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

Response message type:

- **MESSAGETYPE_OIP_ResponseNames**

4.45 MESSAGETYPE_OIP_ GetBgmChannelNames

Purpose:

Retrieve the configured BGM channel names from the Praesideo system.

Parameter structure:

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

figure 4.45: MESSAGETYPE_OIP_ GetBgmChannelNames

Where:

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

Response message type:

- **MESSAGETYPE_OIP_ResponseNames**

4.46 MESSAGETYPE_OIP_ GetConfigId

Purpose:

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

Parameter structure:

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

figure 4.46: MESSAGETYPE_OIP_GetConfigId

Where:

- **header**
  Header of the message, where the **messageType** element is equal to **MESSAGETYPE_OIP_ GetConfigId.**

Response message type:

- **MESSAGETYPE_OIP ResponseConfigId**

4.47 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;
```

figure 4.47: MESSAGETYPE_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**

4.48 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;
```

figure 4.48: MESSAGETYPE_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 section 9.3.14).

Response message type:

- **MESSAGETYPE_OIP_Response**.

Related messages:

- **MESSAGETYPE_OIP_ActivateVirtualControlInput**

4.49 MESSAGETYPE_OIP_ SetSubscriptionUnitCount

Purpose:

Subscribes or unsubscribes to MOST 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 MOST units.

Parameter structure:

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

figure 4.49: MESSAGETYPE_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**

4.50 MESSAGETYPE_OIP_ SetSubscriptionVirtual ControlInputs

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;
```

figure 4.50: MESSAGETYPE_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**

4.51 MESSAGETYPE_OIP_ GetVirtualControlInput Names

Purpose:

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

Parameter structure:

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

figure 4.51: MESSAGETYPE_OIP_ GetVirtualControlInputNames

Where:

- **header**
  Header of the message, where the messageType element is equal to **MESSAGETYPE_OIP_ GetVirtualControlInputNames.**

Response message type:

- **MESSAGETYPE_OIP_ResponseNames**

4.52 MESSAGETYPE_OIP_ GetConfiguredUnits

Purpose:

Retrieve the configured units from the Praesideo system.

Parameter structure:

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

figure 4.52: MESSAGETYPE_OIP_ GetConfiguredUnits

Where:

- **header**
  Header of the message, where the messageType element is equal to **MESSAGETYPE_OIP_ GetConfiguredUnits.**

Response message type:

- **MESSAGETYPE_OIP_ResponseUnits**

4.53 MESSAGETYPE_OIP_ GetConnectedUnits

Purpose:

Retrieve the connected units from the Praesideo system.

Parameter structure:

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

figure 4.53: MESSAGETYPE_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 Praesideo 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 section 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;
```

figure 5.1: MESSAGETYPE_OIP_Response

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 section 10.

Related messages:

- Any command message not described in section 5.

5.3 MESSAGETYPE_OIP_ResponseGetNcoVersion

Purpose:

Responses to the command message **MESSAGETYPE_OIP_GetNcoVersion**.

Parameter structure:

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

figure 5.2: MESSAGETYPE_OIP_ResponseGetNcoVersion

Where:

- **header**
  Header of the message, where the **messageType** element is equal to **MESSAGETYPE_OIP_ResponseGetNcoVersion**.
- **version**
  Version of the NCO software in the format "M.m.b". Where:
  - M: The major version number
  - m: The minor version number
  - b: The build number

Related messages:

- **MESSAGETYPE_OIP_GetNcoVersion**

5.4 MESSAGETYPE_OIP_ ResponseCallId

Purpose:

Responses to the command message **MESSAGETYPE_OIP_CreateCall** and **MESSAGETYPE_OIP_StartCall**.

Parameter structure:

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

figure 5.3: MESSAGETYPE_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_CreateCall**
- **MESSAGETYPE_OIP_StartCreatedCall**
- **MESSAGETYPE_OIP_StartCall**
- **MESSAGETYPE_OIP_StopCall**
- **MESSAGETYPE_OIP_AbortCall**
- **MESSAGETYPE_OIP_AddToCall**
- **MESSAGETYPE_OIP_RemoveFromCall**

5.5 MESSAGETYPE_OIP_ ResponseReportFault

Purpose:

Response to the command message **MESSAGETYPE_OIP_ReportFault**.

Parameter structure:

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

figure 5.4: MESSAGETYPE_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**

5.6 MESSAGETYPE_OIP_ ResponseNames

Purpose:

Responses to the command messages **MESSAGETYPE_OIP_GetXXXNames**.

Parameter structure:

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

figure 5.5: MESSAGETYPE_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**

5.7 MESSAGETYPE_OIP_ ResponseConfigId

Purpose:

Responses to the command message **MESSAGETYPE_OIP_GetConfigId**.

Parameter structure:

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

figure 5.6: MESSAGETYPE_OIP_ ResponseConfigId

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**

5.8 MESSAGETYPE_OIP_ ResponseUnits

Purpose:

Responses to the command message **MESSAGETYPE_OIP_GetXXXUnits**.

Parameter structure:

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

figure 5.7: MESSAGETYPE_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 Praesideo 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;
```

figure 6.1: Notification messages

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).

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 started on Call-Stations or other Open Interface connections.

Parameter structure:

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

figure 6.2: MESSAGETYPE_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 section 9.3.7 for the definitions of the call states.

Related messages:

- **MESSAGETYPE_OIP_StartCall**
- **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;
```

figure 6.3: MESSAGETYPE_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 section 9.3.3 for the different types.
- **alarmState**
  The new state of the alarm. See section 9.3.4 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;
```

figure 6.4: MESSAGETYPE_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 section 9.3.5 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:

Send 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;
```

figure 6.5: MESSAGETYPE_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 section 9.3.6 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;
```

figure 6.6: MESSAGETYPE_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_ NotifyDiagEvent

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;
```

figure 6.7: MESSAGETYPE_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 section 9.3.9 for the action definitions.
- **diagnosticEvent**
  Diagnostic event information. See section 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;
```

figure 6.8: MESSAGETYPE_OIP_NotifyBgmVolume

Where:

- **header**
  Header of the message, where the **messageType** element is equal to **MESSAGETYPE_OIP_NotifyBgmVolume**.
- **zone**
  The name of the BGM zone, which volume was changed.
- **volume**
  Volume setting of the BGM. Value range: **0…-96** (dB).

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 MOST units has changed.

Parameter structure:

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

figure 6.9: MESSAGETYPE_OIP_NotifyUnitCount

Where:

- **header**
  Header of the message, where the messageType element is equal to **MESSAGETYPE_OIP_ NotifyUnitCount**.

- **numberConnected**
  The number of connected MOST units.

Related messages:

- **MESSAGETYPE_OIP_SetSubscriptionUnitCount**

6.10 MESSAGETYPE_OIP_ NotifyVirtualControl InputState

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;
```

figure 6.10: MESSAGETYPE_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 routing list.

- **state**
  The state of the virtual control inputs. See section 9.3.15 for the definitions of the states.

Related messages:

- **MESSAGETYPE_OIP_SetSubscriptionVirtual ControlInputs**
- **MESSAGETYPE_OIP_activateVirtualControlInput**
- **MESSAGETYPE_OIP_deactivateVirtualControlInput**

7 Diagnostic event structures

7.1 Introduction

The Praesideo 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 Praesideo 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 section 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;
```

figure 7.1: Diagnostic event structure

Where:

- **diagMessageType**
  The message type indicator for the diagnostic structure as defined in section 9.5. 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 section 9.4.2 for the diagnostic group definitions.
- **diagEventId**
  The identification of the event as generated by the Praesideo 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 section 8.

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 section 9.4.1).
- 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_BoosterSpareSwitch

Purpose:

This diagnostic event indicates that a Power Amplifier switches over to its associated spare Power Amplifier. In general this diagnostic events is related to supervision fault diagnostic events described in section 7.4.1.

Parameter structure:

```
struct {
   DIAGEVENTHEADER   header;
   STRING            spareName;
   DWORD             spareSerialNumber;
} BoosterSpareSwitchDiagEvent;
```

figure 7.2:
MESSAGETYPE_OIP_BoosterSpareSwitch

Where:

- **header**
  Header of the event, where the **diagMessageType** element is equal to **DET_BoosterSpareSwitch**.
- **spareName**
  The name of the spare Power Amplifier.
- **spareSerialNumber**
  The serial number of the spare Power Amplifier.

7.2.5 DET_BoosterSpareSwitchReturn

Purpose:

This diagnostic event indicates that a Power Amplifier is operational again and that the switchover to the spare is ended.

Parameter structure:

The Diagnostic Event structure is equal to the **BoosterSpareSwitchDiagEvent** as described in section 7.2.4, wherein the **diagMessageType** is equal to **DET_BoosterSpareSwitchReturn**.

7.2.6 DET_UnitConnect

Purpose:

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

Parameter structure:

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

7.2.7 DET_MostHalfPowerModeStart

Purpose:
This diagnostic event indicates that the Praesideo network is operating in half power mode.

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

7.2.8 DET_MostHalfPowerModeEnd

Purpose:
This diagnostic event indicates that the Praesideo network is operating on full power (default) mode. Note that on start of the Praesideo system the full power mode is operational, but is not represented in a diagnostic event. Only the switch to and from half power mode is reported.

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

7.2.9 DET_NCStartup

Purpose:
This diagnostic event indicates that the Praesideo 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_NCStartup**.

7.2.10 DET_OpenInterfaceConnect

Purpose:
This diagnostic event indicates that a remote system has connected to the Praesideo 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**.

7.2.11 DET_OpenInterfaceDisconnect

Purpose:
This diagnostic event indicates that a remote system has disconnected from the Praesideo 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**.

7.2.12 DET_OpenInterfaceConnectFailed

Purpose:
This diagnostic event indicates that a remote system has attempted to connect to the Praesideo 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**.

7.2.13 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 **DIAGEVENTHEAER**, wherein the **diagMessageType** is equal to **DET_CallLoggingSuspended**.

7.2.14 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**.

7.2.15 DET_BoosterSpareSwitch2

Purpose:

This diagnostic event indicates that an Audio Output on a Multi-Channel Interface switches over to its associated spare. In general this diagnostic event is related to supervision fault diagnostic events described in section 7.4.1.

Parameter structure:

```
struct {
   DIAGEVENTHEADER   header;
   STRING            spareName;
  } BoosterSpareSwitch2DiagEvent;
```

figure 7.3: DET_BoosterSpareSwitch2

Where:

- **header**
  Header of the event, where the **diagMessageType** element is equal to **DET_BoosterSpareSwitch2**.
- **spareName**
  The name of the spare Basic Amplifier.

7.2.16 DET_BoosterSpareSwitchReturn2

Purpose:

This diagnostic event indicates that an Audio Output on a Multi-Channel Interface is operational again and that the switchover to the spare has ended.

Parameter structure:

The Diagnostic Event structure is equal to the **BoosterSpareSwitch2DiagEvent** as described in section 7.2.15, wherein the diagMessageType is equal to **DET_BoosterSpareSwitchReturn2**.

7.2.17 DET_UserLogIn

Purpose:

This diagnostic event Indicates that a user has logged in on 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_UserLogIn**.

7.2.18 DET_UserLogOut

Purpose:

This diagnostic event indicates that a user has logged out on 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_UserLogOut**.

7.2.19 DET_UserLogInFailed

Purpose:

This diagnostic event indicates that a login attempt on a call station 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**.

7.2.20 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.

7.2.21 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.

7.3 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 section 7.2.

7.3.1 DET_CallStartDiagEventV2

Purpose:

This diagnostic event indicates the start of a call in the Praesideo 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;
```

figure 7.4: DET_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 section 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 section 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 Praesideo 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;
```

figure 7.5: DET_CallEndDiagEventV2

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 section 9.3.7 for the definitions of the call states.
- **callAborted**
  Whether a call was aborted.
- **callStopReason**
  Why the call was stopped or aborted. See section 9.3.8 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;
```

figure 7.6: DET_CallChangeResourceDiagEventV2

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;
```

figure 7.7: DET_CallTimeoutDiagEventV2

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.

7.4 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 Praesideo 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 Supervision Faults

Purpose:

A special group within the diagnostic faults is the supervision faults. This group of faults supervises the hardware equipment of the Praesideo System for correct operation. Failures in operation are detected and reported using the diagnostic event described within the supervision faults. After the report, the Praesideo System tries to bypass the problem, using spare equipment. This process can result in one or more general diagnostic events as described in section 7.2.

Parameter structure:

```
struct {
   DIAGEVENTHEADER   header;
   WORD              supervisionItemId;
} SviDiagEvent;
```

figure 7.8: Supervision fault

Where:

- **header**
  Header of the event, where the **diagMessageType** element is set to one of the items described in the sections below.

- **supervisionItemId**
  The identification of the supervision Item. This is a number of the amplifier, input or output dependant on the kind of supervision diagnostic event and equipment.

7.4.1.1 DET_AmpFailure

Purpose:
This diagnostic event indicates an amplifier failure inside a Power Amplifier.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_AmpFailure**.

7.4.1.2 DET_AmpFailureOrOverload

Purpose:
This diagnostic event indicates that either an amplifier failure or an amplifier overload has occurred inside a Power Amplifier.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_AmpFailureOrOverload**.

7.4.1.3 DET_AmpGroundShort

Purpose:
This diagnostic event indicates that a ground-shorted amplifier inside a Power Amplifier is detected.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_AmpGroundShort**.

7.4.1.4 DET_AmpOverheat

Purpose:
This diagnostic event indicates that an overheated amplifier inside a Power Amplifier is detected.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_AmpOverheat**.

7.4.1.5 DET_AmpOverheatMute

Purpose:
This diagnostic event indicates that an overheated amplifier inside a Power Amplifier is detected and the amplifier is muted.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_AmpOverheatMute**.

7.4.1.6 DET_AmpOverload

Purpose:
This diagnostic event indicates that an overloaded amplifier inside a Power Amplifier is detected.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_AmpOverload**.

7.4.1.7 DET_AmpShortCircuit

Purpose:
This diagnostic event indicates that a short-circuited amplifier inside a Power Amplifier is detected.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_AmpShortCircuit**.

7.4.1.8 DET_AudioPathSupervision

Purpose:
This diagnostic event indicates that an audio-path failure is detected. Note, although it is sent, the **supervisionItemId** element in the structure is not used (value set to zero).

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_AudioPathSupervision**.

7.4.1.9 DET_EolFailure

Purpose:

This diagnostic event indicates that an amplifier has lost its (WLS1) End-of Line board.

Parameter structure:

The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_EolFailure**.

7.4.1.10 DET_LineShortCircuit

Purpose:

This diagnostic event indicates that a short-circuited amplifier outside a Power Amplifier is detected.

Parameter structure:

The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_LineShortCircuit**.

7.4.1.11 DET_PilotToneCalibration

Purpose:

This diagnostic event indicates that a calibration failure of the pilot tone is detected.

Parameter structure:

The Diagnostic Event structure is equal to the **SviDiagEvent** as described insection 7.4.1, wherein the **diagMessageType** is equal to **DET_PilotToneCalibration**.

7.4.1.12 DET_PowerBackupSupply

Purpose:

This diagnostic event indicates a loss of the backup power supply. Note, although it is sent, the **supervisionItemId** element in the structure is not used (value set to zero).

Parameter structure:

The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_PowerBackupSupply**.

7.4.1.13 DET_PowerMainSupply

Purpose:

This diagnostic event indicates a loss of the mains power supply. Note, although it is sent, the **supervisionItemId** element in the structure is not used (value set to zero).

Parameter structure:

The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_PowerMainSupply**.

7.4.1.14 DET_MicrophoneSupervision

Purpose:

This diagnostic event indicates that a microphone failure is detected. Note that this diagnostic event mainly applies to a call-station (where the microphone is connected to the first input, value 0), but also to a normal audio input, configured as supervised microphone input (Power Amplifier specific).
Note that for the 8x60W Power Amplifier the inputs are related to the values 0, 1, 4 and 5.

Parameter structure:

The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_MicrophoneSupervision**.

7.4.1.15 DET_LineInputSupervision

Purpose:

This diagnostic event indicates that a line input failure is detected. Note that this diagnostic event only applies to network controllers, audio expanders and power amplifiers.

Parameter structure:

The Diagnostic Event structure is equal to the SviDiagEvent as described in section 7.4.1, wherein the diagMessageType is equal to DET_LineInputSupervision.

7.4.1.16 DET_SystemInputContact

Purpose:

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

Parameter structure:

The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_SystemInputContact**.

7.4.1.17 DET_CobraNetInterface

Purpose:

This diagnostic event indicates that the CobraNet module inside a CobraNet interface reports an interface problem.

Parameter structure:

```
struct {
   SviDiagEvent    sviHeader;
   DWORD           errorCode;
} CobraNetDiagEvent;
```

figure 7.9: DET_CobraNetInterface

Where:

- **sviHeader**
  Supervision fault header of the event (see section 7.4.1), where the **diagMessageType** element is equal to **DET_CobraNetInterface**.
- **errorCode**
  The error code reported by the CobraNet interface module. For the error-codes specification, see [IUI PRAESIDEO]. Note that the error code is a byte value, stored in the third byte of the error-code (mask 0x00FF0000).

7.4.1.18 DET_CobraNetNetwork

Purpose:

This diagnostic event indicates that the CobraNet module inside a CobraNet interface reports a network problem.

Parameter structure:

The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1.17, wherein the **diagMessageType** is equal to **DET_CobraNetNetwork**.

7.4.1.19 DET_EndOfLineSupervision

Purpose:

This diagnostic event indicates that a WLS2 end-of-line failure(s) is detected. Note that the presented structure contains a variable length array.

Parameter structure:

```
struct {
   SviDiagEvent    sviHeader;
   BYTE            nrSlaves;
   TWls2Slave      slaves[];
} WLS2DiagEvent;
```

figure 7.10: DET_EndOfLineSupervision

Where **TWls2Slave** is defined as:

```
struct {
   BYTE            slaveAddress;
   STRING          slaveName;
} TWls2Slave
```

figure 7.11: TWls2Slave

Where:

- **sviHeader**
  Supervision fault header of the event, where the **diagMessageType** element is equal to **DET_EndOfLineSupervision**.
- **nrSlaves**
  The number of slaves present in the slaves array element. Only this amount of array elements is transmitted.
- **slaves[ ]**
  Array element holding the slave information. The actual length of the array is defined in the **nrSlave** element. The structure of each array element is described below.
  - **slaveAddress**
    The address of the WLS2 Board as preset on the WLS2 board.
  - **slaveName**
    The name of the WLS2 slave board as given in the Praesideo System configuration.

7.4.1.20 DET_LoudspeakerSupervision

Purpose:
This diagnostic event indicates that a WLS2 loudspeaker failure(s) is detected.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1.19, wherein the **diagMessageType** is equal to **DET_LoudspeakerSupervision**.

7.4.1.21 DET_RemotePowerSupply

Purpose:
This diagnostic event indicates a loss of the power supply on the remote part of the Long Distance Call Station connected to its base unit.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_RemotePowerSupply**.

7.4.1.22 DET_RemoteBackupPowerSupply

Purpose:
This diagnostic event indicates a loss of the backup power supply on the remote part of the Long Distance Call Station connected to its base unit.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_RemoteBackupPowerSupply**.

7.4.1.23 DET_RemoteConnection

Purpose:
This diagnostic Event indicates the loss of the connection between the remote part of the Long Distance Call Station and its base unit.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_RemoteConnection**.

7.4.1.24 DET_PowerMainsSupply2

Purpose:
This diagnostic Event indicates the loss of mains power supply for a Basic Amplifier.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEven**t as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_PowerMainsSupply2**.

7.4.1.25 DET_PowerBackupSupply2

Purpose:
This diagnostic Event indicates the loss of backup power supply for a Basic Amplifier.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_PowerBackupSupply2**.

7.4.1.26 DET_GroupAFault

Purpose:
This diagnostic Event indicates a failure in group A for audio outputs with A/B switching or class-A wiring.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_GroupAFault**.

7.4.1.27 DET_GroupBFault

Purpose:
This diagnostic Event indicates a failure in group B for audio outputs with A/B switching or class-A wiring.

Parameter structure:
The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_ GroupBFault**.

7.4.1.28 DET_HundredVLineFault

Purpose:

This diagnostic Event indicates a 100VlineFault in A/B wiring mode while determining whether a GroupAFault or GroupBFault must be generated.

Parameter structure:

The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_HundredVLineFault**.

7.4.1.29 DET_ClassASwitchOver

Purpose:

This diagnostic Event indicates detection of closure of the second (B) relay while in class-A mode.

Parameter structure:

The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_ClassASwitchOver**.

7.4.1.30 DET_Wls2CcbSyncError

Purpose:

This diagnostic Event indicates Supervision Control Board failure.

Parameter structure:

The Diagnostic Event structure is equal to the **SviDiagEvent** as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_Wls2CcbSyncError**.

7.4.1.31 DET_OmneoInterface

Purpose:

This diagnostic event indicates that the OMNEO module inside an OMNEO unit reports an interface problem.

Parameter structure:

```
struct {
   SviDiagEvent    sviHeader;
   DWORD           errorCode;
} OmneoDiagEvent;
```

figure 7.12: DET_OmneoInterface

Where:

- **sviHeader**
  Supervision fault header of the event (see section 7.4.1), where the **diagMessageType** element is equal to **DET_OmneoInterface**.
- **errorCode**
  The error code reported by the OMNEO interface module. For the error-codes specification, see the Praesideo Installation and User Instructions.
  Note that the error code is a byte value, stored in the third byte of the error-code (mask 0x00FF0000).

7.4.1.32 DET_OmneoNetwork

Purpose:

This diagnostic event indicates that the OMNEO module inside an OMNEO unit reports a network problem.

Parameter structure:

The Diagnostic Event structure is equal to the SviDiagEvent as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_OmneoNetwork**.

7.4.1.33 DET_ AmpFanFault

Purpose:

This diagnostic event indicates that a fan fault inside a Power Amplifier is detected.

Parameter structure:

The Diagnostic Event structure is equal to the SviDiagEvent as described in section 7.4.1, wherein the **diagMessageType** is equal to **DET_AmpFanFault**.

7.4.2 DET_BoosterStandby

Purpose:

This diagnostic event indicates that a Power Amplifier remains in standby mode. It refuses to go out of standby mode.

Parameter structure:

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

7.4.3 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;
```

figure 7.13: DET_CallStationExtension

Where:

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

7.4.4 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**.

7.4.5 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;
```

figure 7.14: DET_ConfigurationVersion

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.

7.4.6 DET_FlashCardChecksum

Purpose:

This diagnostic event indicates that a checksum mismatch of the prerecorded messages on the flashcard inside the Network Controller is detected.

Parameter structure:

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

7.4.7 DET_FlashCardMissing

Purpose:

This diagnostic event indicates that the flashcard is reported absent (not present inside the Network controller or the flashcard is defect).

Parameter structure:

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

7.4.8 DET_IllegalConfiguration

Purpose:

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

Parameter structure:

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

figure 7.15: DET_IllegalConfiguration

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'.

7.4.9 DET_MemoryError

Purpose:

This diagnostic event indicates that a memory failure inside the Network Controller of the Praesideo System is detected.

Parameter structure:

```
struct {
   DIAGEVENTHEADER   header;
   BOOLEAN           eePromError;
   BOOLEAN           flashError;
} MemoryErrorDiagEvent;
```

figure 7.16: DET_MemoryError

Where:

- **header**
  Header of the event, where the **diagMessageType** element is equal to **DET_MemoryError**.
- **eePromError**
  Indicates that an error is detected in the EE-Prom memory.
- **flashError**
  Indicates that an error is detected in the main-flash memory.

7.4.10 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 flash-card.

Parameter structure:

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

figure 7.17: DET_PrerecordedMessagesNames

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 flash-card, but used in the Praesideo System configuration. A comma separates each name in the list.

7.4.11 DET_RedundantRingBroken

Purpose:
This diagnostic event indicates that the Praesideo System has detected that the redundant network ring is not closed.

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

7.4.12 DET_UnitMissing

Purpose:
This diagnostic event indicates a missing unit, which was configured in the Praesideo 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**.

7.4.13 DET_UnitNotConfigured

Purpose:
This diagnostic event indicates that a unit is detected that is not configured within the Praesideo System.

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

7.4.14 DET_UnitReset

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

Parameter structure:

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

figure 7.18: DET_UnitReset

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. See section 9.4.3 for the description of the chip-types

7.4.15 DET_UnitUnknownType

Purpose:
This diagnostic event indicates that an unknown unit-type is detected.

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

7.4.16 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 Praesideo System.

Parameter structure:

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

figure 7.19: DET_UserInjectedFault

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**

7.4.17 DET_WLSBoard

Purpose:

This diagnostic event indicates that a mismatch is detected between the master WLS-board that has been configured and the actual installed master WLS-board that is present in the Power Amplifier. Note this diagnostic event is not reported when the board is present in the Power Amplifier and disabled in the configuration.

Parameter structure:

```
typedef struct
{
   DIAGEVENTHEADER   header;
   UINT              amplifier;
   BOOLEAN           present;
} WLSBoardDiagEvent;
```

figure 7.20: DET_WLSBoard

Where:

- **header**
  Header of the event, where the **diagMessageType** element is equal to **DET_WLSBoard**.
- **amplifier**
  Identifies the amplifier, that misses the WLS-Board.
- **present**
  Indicates whether the board was present in the unit but not configured (**TRUE**) or the board was configured but not present (**FALSE**).

7.4.18 DET_WLSBoardMainSpare

Purpose:

This diagnostic event indicates that a mismatch is detected between the main booster master WLS-board that has been configured and the actual spare booster installed master WLS-board that is present. Note this diagnostic event is not reported when the board is present in spare amplifier and no WLS supervision is configured for main amplifier. The main booster is the originator of the diagnostic event.

Parameter structure:

```
typedef struct {
   DIAGEVENTHEADER   header;
   UINT              amplifier;
   BOOLEAN           precent;
} WLSBoardMainSpareDiagEvent;
```

figure 7.21: DET_WLSBoardMainSpare

Where:

- **header**
  Header of the event, where the diagMessageType element is equal to DET_WLSBoardMainSpare.
- **amplifier**
  Identifies the amplifier, that gives the WLS-Board mismatch.
- **present**
  Indicates whether the board was present in the spare amplifier but not configured in the main amplifier (TRUE) or the board was configured in the main amplifier but not present (FALSE) in the spare amplifier.

7.4.19 DET_IncompatibleHWVersion

Purpose:

This diagnostic event indicates a mismatch between the expected HW version of a unit and the detected HW version of a unit. For full support of all functionality inside the Praesideo System, the hardware must be equipped with related features. When HW features are not present, this mismatch diagnostic event is generated.

Parameter structure:

```
struct {
   DIAGEVENTHEADER   header;
   STRING            currentVersion;
   STRING            minimalVersion;
} ConfigurationVersionDiagEvent;
```

figure 7.22: DET_IncompatibleHWVersion

Where:

- **header**
  Header of the event, where the **diagMessageType** element is equal to **DET_IncompatibleHWVersion**.
- **currentVersion**
  String containing the current HW version of the unit.
- **minimalVersion**
  String containing the minimal expected HW version of the unit to let the current firmware version operate successfully.

7.4.20 DET_AmpMissing

Purpose:

This diagnostic event indicates that a Basic Amplifier is missing.

Parameter structure:

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

7.4.21 DET_InvalidFWVersion

Purpose:

This diagnostic event indicates that a unit has an invalid Firmware Version.

Parameter structure:

```
struct {
   DIAGEVENTHEADER   header;
   STRING            currentFWVersion;
   STRING            requiredFWVersion;
} InvalidFWVersionDiagEvent;
```

figure 7.23: DET_InvalidFWVersion

Where:

- **header**
  Header of the message, where the **messageType** element is equal to **DET_InvalidFWVersion**.
- **currentFWVersion**
  String containing the current FW version of the unit.
- **requiredFWVersion**
  String containing the required FW version of the unit.

7.4.22 DET_RedundantPowerFault

Purpose:

This diagnostic event indicates that a redundant power supply has a failure.

Parameter structure:

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

7.4.23 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 NCO. This event is always sent in a message with the TOIActionType equal to **OIACT_EXISTING_LAST** (See section 9.3.9).

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.

7.4.24 DET_ZoneLineFault

Purpose:

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

Parameter structure:

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

figure 7.24: DET_ZoneLineFault

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.

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;
```

figure 8.1: Event originator structure

Where:

- **originatorType**
  The originator type indicator for the originator structure as defined section 9.6. 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 Praesideo System.

Parameter structure:

```
struct {
   ORIGINATORHEADER    header;
   STRING              unitName;
   DWORD               serialNumber;
   TUnitType           unitType;
} UnitOriginator;
```

figure 8.2: OIEOT_UnitEventOriginator

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 Praesideo System configuration.
- **serialNumber**
  The serial number of the originator unit.
- **unitType**
  The type of the originator unit. See section 9.4.4 for the definition of the unit-types.

8.4 OIEOT_OpenInterfaceEvent Originator

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;
```

figure 8.3: OIEOT_OpenInterfaceEventOriginator

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.
- **portNumber**
  The TCP-port number of the open interface connection.
- **userName**
  The login user name of the open interface connection.

8.5 OIEOT_ControlInputEvent Originator

Purpose:

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

Parameter structure:

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

figure 8.4: OIEOT_ControlInputEventOriginator

Where:

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

8.6 OIEOT_AudioOutputEvent Originator

Purpose:

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

Parameter structure:

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

figure 8.5: OIEOT_AudioOutputEventOriginator

Where:

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

8.7 OIEOT_AudioInputEvent Originator

Purpose:

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

Parameter structure:

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

figure 8.6: OIEOT_AudioInputEventOriginator

Where:

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

8.8 OIEOT_UnitMenuEvent Originator

Purpose:

This originator represents unit menu activation on the front display of the network controller.

Parameter structure:

The Originator structure is equal to the **UnitHeader** as described in section 8.3, wherein the **originatorType** is equal to **OIEOT_UnitMenuEventOriginator**.

8.9 OIEOT_UserEventOriginator

Purpose:

This originator represents user action performed on the system.

Parameter structure:

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

figure 8.7: OIEOT_UserEventOriginator

Where:

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

9 OIP constant values

9.1 Introduction

In this section of the Open Interface Programming Instructions, 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.

9.2 Protocol Constants

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

table 9.1: Protocol Constants

Constant Meaning	Value
Port Number for the connection	9401
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

9.3 General Constants

9.3.1 TOIEventId

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

table 9.2: TEventId

Constant name	Value
OI_UNDEFINED_EVENTID	0xFFFFFFFF

9.3.2 TOICallId

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

table 9.3: TOICallId

Constant name	Value
OI_UNDEFINED_CALLID	0xFFFFFFFF

9.3.3 TOIAlarmType

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

table 9.4: TOIAlarmType

Constant name	Value
OIAT_EVAC	0x00000000
OIAT_FAULT	0x00000001

9.3.4 TOIAlarmState

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

table 9.5: TOIAlarmState

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

9.3.5 TOIResourceState

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

table 9.6: TOIResourceState

Constant name	value
OIRS_FREE	0x00000000
OIRS_INUSE	0x00000001

9.3.6 TOIResourceFaultState

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

table 9.7: TOIResourceFaultState

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

9.3.7 TOICallState

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

table 9.8: TOICallState

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

9.3.8 TOICallStopReason

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

table 9.9: TOICallStopReason

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
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 unknown reason.	0x00000004

9.3.9 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 section 3.4.3.1. The valid values used within this type are described in the table below.

table 9.10: TOIActionType

Constant name	Value
OIACT_NEW The specified diagnostic event is added to the event storage in the Praesideo 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 Praesideo 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 section 4.38).	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 specific diagnostic event is of type DET_NoFaults.	0x00000007

9.3.10 TOICallOutputHandling

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

table 9.11: TOICallOutputHandling

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**

9.3.11 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 section 3.4.3.1.

table 9.12: TOICallStackingMode

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**

9.3.12 TOICallTiming

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

table 9.13: TOICallTiming

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**

9.3.13 TOICallStackingTimeout

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

table 9.14: TOICallStackingTimeout

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

9.3.14 TOIVirtualControlInput Deactivation

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 section 3.4.3.1.

table 9.15: TOIVirtualControlInput-Deactivation

Constant name	Value
OIVCI_STOP Stop the running action gracefully.	**0x00000000**
OIVCI_ABORT Abort the running action immediately.	**0x00000001**

9.3.15 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 section 3.4.3.1.

table 9.16: TOIVirtualControlInput-State

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**

9.4 Diagnostic Constant values

9.4.1 TdiagEventState

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

table 9.17: TDiagEventState

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

9.4.2 TdiagEventGroup

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

table 9.18: TDiagEventGroup

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

9.4.3 TchipType

The chip types as used within the Praesideo System are represented by the chip type. The type is mapped upon an UINT basic type as described in section 3.4.3.1. The valid values used within this type are described in the table below.

table 9.19: TchipType

Constant name	value
OICT_MMP Multi-Media Processor, the processor controlling the units (like Power Amplifier, Audio Expander, Call-Station, CobraNet Interface, etc., but also the audio processor inside the Network Controller).	0x00000000
OICT_MAINCPU The main processor in the Network Controller.	0x00000001
OICT_CNM The controller module of the CobraNet unit.	0x00000002
OICT_UNKNOWN The Praesideo System was not able to determine the chip type.	0x00000003
OICT_ OICCB A Supervision Control Board of a Power Amplifier.	0x00000004
OICT_ REMOTE The remote part of the Remote Call Station.	0x00000005
OICT_OMNEO The controller of the OMNEO unit.	0x00000006

9.4.4 TunitType

The unit types as used within the Praesideo System are represented by the Unit type. The type is mapped upon an UINT basic type as described in section 3.4.3.1. The valid values used within this type are described in the table below.

table 9.20: TunitType

Constant name	value
OIUT_CST Call-Station Unit	0x00000000
OIUT_NC Network Controller Unit	0x00000001
OIUT_BST Power Amplifier Unit	0x00000002
OIUT_AIO Audio Expander Unit	0x00000003
OIUT_POFGOF Fiber Interface Unit	0x00000004
OIUT_CEX CobraNet Interface	0x00000005
OIUT_UNKNOWN Unknown Unit Type	0x00000006
OIUT_MCI Multi Channel Interface	0x00000007
OIUT_CRF Call Stacker Unit	0x00000008

9.4.5 TdiagEventType

The diagnostic event types as used within the Praesideo System are represented by the diagnostic-event type. The type is mapped upon a UINT basic type as described in section 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 later version of Praesideo is probably installed on the network controller.

table 9.21: TDiagEventType

Constant name	Value
DET_CallChangeResourceV2	0x00467105
DET_CallEndV2	0x00467106
DET_CallStartV2	0x00467107
DET_CallTimeoutV2	0x00467108
DET_BoosterSpareSwitch	0x00467202
DET_BoosterSpareSwitchReturn	0x00467203
DET_EvacAcknowledge	0x00467204
DET_EvacReset	0x00467205
DET_EvacSet	0x00467206
DET_MostHalfPowerModeEnd	0x00467207
DET_MostHalfPowerModeStart	0x00467208
DET_NCStartup	0x00467209
DET_OpenInterfaceConnect	0x0046720A
DET_OpenInterfaceDisconnect	0x0046720B
DET_UnitConnect	0x0046720E
DET_CallLoggingSuspended	0x0046720F
DET_CallLoggingResumed	0x00467210
DET_BoosterSpareSwitch2	0x00467211
DET_BoosterSpareSwitchReturn2	0x00467212
DET_UserLogIn	0x00467213
DET_UserLogOut	0x00467214
DET_UserLogInFailed	0x00467215
DET_OpenInterfaceConnectFailed	0x00467216
DET_BackupPowerModeStart	0x00467217
DET_BackupPowerModeEnd	0x00467218
DET_AmpFailure	0x00467301
DET_AmpFailureOrOverload	0x00467302
DET_AmpGroundShort	0x00467303
DET_AmpOverheat	0x00467304
DET_AmpOverheatMute	0x00467305
DET_AmpOverload	0x00467306
DET_AmpShortCircuit	0x00467307
DET_AudioPathSupervision	0x00467308
DET_BoosterStandby	0x00467309
DET_CallStationExtension	0x0046730A
DET_CobraNetInterface	0x0046730B
DET_CobraNetNetwork	0x0046730C
DET_ConfigurationFile	0x0046730D
DET_ConfigurationVersion	0x0046730E
DET_EolFailure	0x0046730F

table 9.22: TDiagEventType (continued)

Constant name	Value
DET_FlashCardChecksum	0x00467310
DET_FlashCardMissing	0x00467311
DET_IllegalConfiguration	0x00467312
DET_LineShortCircuit	0x00467313
DET_MemoryError	0x00467314
DET_MicrophoneSupervision	0x00467315
DET_PilotToneCalibration	0x00467316
DET_PowerBackupSupply	0x00467317
DET_PowerMainSupply	0x00467318
DET_PrerecordedMessagesNames	0x00467319
DET_RedundantRingBroken	0x0046731A
DET_SystemInputContact	0x0046731B
DET_UnitMissing	0x0046731C
DET_UnitNotConfigured	0x0046731D
DET_UnitReset	0x0046731E
DET_UnitUnknownType	0x0046731F
DET_UserInjectedFault	0x00467320
DET_WLSBoard	0x00467321
DET_EndOfLineSupervision	0x00467322
DET_LoudspeakerSupervision	0x00467323
DET_IncompatibleHWVersion	0x00467324
DET_RemotePowerSupply	0x00467325
DET_RemoteBackupPowerSupply	0x00467326
DET_RemoteConnection	0x00467327
DET_PowerBackupSupply2	0x00467328
DET_PowerMainsSupply2	0x00467329
DET_GroupAFault	0x0046732A
DET_GroupBFault	0x0046732B
DET_ClassASwitchOver	0x0046732C
DET_HundredVLineFault	0x0046732D
DET_Wls2CcbSyncError	0x0046732E
DET_AmpMissing	0x0046732F
DET_InvalidFWVersion	0x00467330
DET_RedundantPowerFault	0x00467331
DET_LineInputSupervision	0x00467332
DET_WLSBoardMainSpare	0x00467333
DET_NoFaults	0x00467334
DET_ZoneLineFault	0x00467335
DET_OmneoInterface	0x00467336
DET_OmneoNetwork	0x00467337
DET_AmpFanFault	0x00467338

9.5 Message Types

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

table 9.23: Message Types

Constant name	Value
MESSAGETYPE_OIP_Login	0x00447002
MESSAGETYPE_OIP_StartCall	0x00447003
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_ResetEvacAlarm	0x0044700B
MESSAGETYPE_OIP_SetDateAndTime	0x0044700C
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
MESSAGETYPE_OIP_NotifyDiagEvent	0x00447026
MESSAGETYPE_OIP_KeepAlive	0x00447027
MESSAGETYPE_OIP_CreateCall	0x00447028

table 9.24: Message Types (continued)

Constant name	Value
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_CreateCallEx	0x00447037
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_CreateCallEx2	0x0044703E
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

9.6 Event originator Message Types

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

table 9.25: Event originator Message Types

Constant name	Value
OIEOT_NoEventOriginator	0x00477002
OIEOT_UnitEventOriginator	0x00477003
OIEOT_OpenInterfaceEventOriginator	0x00477004
OIEOT_ControlInputEventOriginator	0x00477005
OIEOT_AudioOutputEventOriginator	0x00477006
OIEOT_AudioInputEventOriginator	0x00477007
OIEOT_UnitMenuEventOriginator	0x00477008
OIEOT_UserEventOriginator	0x00477009

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.

table 10.1: Error codes

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 Praesideo 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 Praesideo 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 Praesideo 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 OIERROR_INVALID_PARAMETERS.	0X0044E009

Intentionally left blank.

Intentionally left blank.

Part 2 - Open Interface Library

Intentionally left blank.

11 Introduction

11.1 Purpose

The purpose of this document is to describe the usage of the Praesideo Open Interface based on a Visual Basic implementation. The interfacing is based on the COM technology (OLE automation subset), as described in Microsoft documentation.

11.2 Scope

This user manual describes the use of the open interface in combination with Visual Basic. To understand this document, knowledge is expected on the following issues:

- The VB.NET programming language and its development environment.
- The principle of COM-interfaces and OLE automation.
- The Praesideo system and its installation.

Note that the use of the COM interface is not limited to Visual Basic, but can also be used from other programming languages that support COM. Visual Basic is just used as an example here.

This document is intended for users, who want to use the Praesideo 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 Praesideo are introduced.

Although the Open Interface DLL's do contain the previous versions of the Open Interface (for backward compatibility), only the listed version of the Interface is described in this document. The Interfaces in the DLL are identified using a name extension, whereby the original version of the interface does not have that version naming extension. Section 15.7 provides an overview of the differences between this version and the previous versions of the Open Interface.

If a Praesideo system is controlled via device that does not use Microsoft Windows as an operating system, then the application should control Praesideo directly via the native communication interface. Refer to Part 1 - Open Interface Protocol.

Note
A Praesideo network controller is able to communicate with up to five or seven (version 4.3 and later) Open Interface clients at the same time. This includes connection to Logging Servers.

12 Application control overview

12.1 Principle

The Praesideo 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 Praesideo 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. The Praesideo 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 Praesideo 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 sends all available events as present in the Praesideo System, followed by the new and updated events.

12.1.1 Limitations

It is not allowed to call a method while a callback is active: a busy indication will be returned in that case.

There are some points which need to be highlighted:

- Do not block callback processing and keep the callback processing as short as possible
- Do not make open interface calls from the callback processing.
- Do not make open interface calls to the same open interface connection from multiple threads.

12.2 Referencing the interface

Before the interface can be used within Visual Basic, you need to add a reference to the interface. This can be done in the Visual Basic development environment, using the Project > Add References… menu entry.
In the Add Reference dialog, select COM and than select **PraesideoOpenInterfaceCOMServer** (if present). When not present, select the browse button and select the file **PraesideoOpenInterfaceCOMServer**.dll on the installed location. This wil automatically register the type library for Visual Basic (assuming that the type-library is present in the same directory as the COM-server DLL).

12.3 Interface usage in Visual Basic

After the reference addition, Visual Basic knows the methods and events of the Praesideo Open Interface. The application can call the method on the interface and the Praesideo system will send events to the application.
The connection point technique to use the Praesideo interface methods and events is described in the following section.

12.3.1 Connection point technique

When using the connection point technique in the code to interface with the Praesideo system, the line displayed in figure 12.1 should be added in the declaration section of your VB code. Constructing the interface can be done using the lines displayed in figure 12.2.

```
Private WithEvents PraesideoOI As PraesideoOpenInterface_V0340
```

figure 12.1: Connection point technique (1)

```
Private Sub Form_Load()
   Try
   Set PraesideoOI = New PraesideoOpenInterface_V0340
   ' Note that the event subscription is done automatically by
   ' Visual Basic
    Catch ex As Exception
   'Handle error
   End Try
End Sub

Private Sub Form_Unload(Cancel As Integer)
   ' Note that the event unsubscription is done automatically by
   ' Visual Basic
   Set PraesideoOI = Nothing
End Sub
```

figure 12.2: Connection point technique (2)

Note that the functions **registerClientInterface** and **deregisterClientInterface** are not needed when using connection point technique.
When the interface is defined using the connection point technique, the events coming from the Praesideo system should be declared as follows (callstate example shown).

```
Private Sub PraesideoOI_callState(ByVal callId As Long, _
ByVal state As PRAESIDEOOPENINTERFACECOMSERVERLib.TOICallState)
...
End Sub
```

figure 12.3: Connection point technique (3)

Note that only the events that are required need to be implemented. Unused events can be omitted.

12.3.2 Custom callback technique

When using the custom callback technique in the code to interface with the Praesideo system, the following lines should be added in the declaration section of your VB code.

```
Private PraesideoOI As PraesideoOpenInterface_V0340
Implements IPraesideoOpenInterfaceEvents_V0340
```

figure 12.4: Custom callback technique (1)

The definition of the **PraesideoOpenInterface** object includes only the methods, without the events.
The implements statement defines the implementation of the **IPraesideoOpenInterfaceEvents** interface within the module.
Constructing the interface should be done using the following lines:

```
Private Sub Form_Load()
   Try
      Set PraesideoOI = New PraesideoOpenInterface_V0340
      ' Register this object as a "sink" for the events coming from
      ' the praesideo open interface
      PraesideoOI.registerClientInterface Me
   Catch ex As Exception
      ' Handle error
   End Try
End Sub

Private Sub Form_Unload(Cancel As Integer)
   ' Deregister the call-back functionality
   PraesideoOI.deregisterClientInterface
   Set PraesideoOI = Nothing
End Sub
```

figure 12.5: Custom callback technique (2)

When the interface is defined using the custom callback technique (see code above), the events (as described in section 13.3.37) coming from the Praesideo system should be declared as follows (callstate example shown).

```
Private Sub IPraesideoOpenInterfaceEvents_callState( _
ByVal callId As Long, _
ByVal state As TOICallState)
   …
End Sub
```

figure 12.6: Custom callback technique (3)

Note that all event functions need to be implemented in this case.

12.4 Catching errors

Problems detected during an interface call of the Praesideo system will be reported by means of so called COM-exceptions. To catch the error in VB there are 2 possibilities.

- Use the VB NET **Try, Catch** mechanism
- Use the VB 6.0 statement **On Error Goto** …

12.4.1 Try, catch

The following code sample shows the use of the **Try. Catch** mechanism to catch interface failures. This is the preferred way for .NET applications.

```
    Try
     m_CallId = PraesideoOI.startCall(Routing, Priority, Partial, _
        StartChime, EndChime, Livespeech, AudioInput, Messages,
        Repeat)
        ' Continue normal execution

  Catch ex As Exception
        ' Handle and/or report error
  End Try
```

figure 12.7: Catching errors

For error handling the Visual Basic object **ex** contains the error information. The description can be found in property **Err.Message**. For more information on the Try, Catch mechanisms please refer to the MSDN library.

12.4.2 On Error Goto

The following code sample shows the use of the On Error statement to catch interface failures.

```
   On Error GoTo StartCallError
   m_CallId = PraesideoOI.startCall(Routing, Priority, Partial, _
StartChime, EndChime, Livespeech, AudioInput, Messages, Repeat)
   ' Continue normal execution
   …
StartCallError:
   ' Handle and/or report error
```

figure 12.8: Catching errors

For error handling the Visual Basic object Err contains the error information. The description can be found in property **Err.Description** and the error value in property **Err.Number**. Explanation about the error numbers can be found in section 13.2.2.

13 Interface definition

13.1 Introduction

This section describes the various remote methods available on the Praesideo 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 section 13.2.2 for a description of the error codes.

13.2 Enumeration type definitions

Within the interface various enumeration types are defined to prevent the use of magic (non explaining) numbers.

13.2.1 TCallId

OI_UNDEFINED_CALLID = -1

- Standard indication for a call identifier to which no call is associated.

The call identifiers are transported over Ethernet connection as an unsigned integer value. However, the received call identifiers will be received as signed integers. Always check **OI_UNDEFINED_CALLID**, before making the cast to the unsigned value (which represents the value of the call identifier).

13.2.2 TIOErrorCode

The **TIOErrorCode** type represents the error values, which can be returned by the open interface functions. All error values, except for the value **OIERROR_OK** will throw an exception. The error values have the following meaning:

OIERROR_OK:

- The open interface function has executed successfully.

OIERROR_ALREADY_BUSY:

- The Praesideo system is busy executing another command.

OIERROR_ALREADY_LOGGED_IN:

- The open interface is already logged in to a Praesideo system. Disconnect from the Praesideo system and try again.

OIERROR_ALREADY_REGISTERED:

- The function **registerClientInterface** is called for the second time.

OIERROR_BAD_CREDENTIALS:

- The open interface could not complete the connection, because the username or the password is incorrect.

OIERROR_BUFFER_TOO_SMALL:

- Internal error: contact Bosch Security Systems.

OIERROR_INTERNAL_ERROR:

- The Praesideo system detected an internal error during the processing of the command. Check the Praesideo System configuration. If persistent, contact Praesideo 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 Praesideo system or that a passed value is out of range. Strings are considered to be invalid when their lengths exceed 15000 characters.

OIERROR_NO_CONNECTION:

- The open interface connection to the Praesideo system is not established.

OIERROR_NOT_REGISTERED:

- The function **deregisterClientInterface** is called without the registration.

OIERROR_UNABLE_TO_MAKE_CONNECTION:

- The open interface could not complete the connection, due to problems with the link to the Praesideo system.

OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER:

- The Praesideo network Controller does not support the function called. In general this means that the Praesideo Open Interface has a newer version than the Praesideo Network Controller. The added functions to the open interface between the two versions cannot be executed.

OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE:

- The request for a member inside the IDiagEvent_Vxxxx object has failed, because the requested member is not supported given the current **TOIDiagEventType** of the object.

OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE:

- The request for a member inside the IEventOriginator_Vxxxx object has failed, because the requested member is not supported given the current **TOIEventOriginatorType** of the object.

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.**

13.2.3 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.

13.2.4 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 proceed/overrule 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.

13.2.5 TOICallRepeatCount

The **TOICallRepeatCount** type gives some standard values for the message repeat counter.

OI_REPEAT4EVER:

- Indicates that the indicated messages will be repeated forever.

OI_PLAY_ONCE:

- Indicates that the indicated messages should be played only once.

OI_REPEAT_ONCE:

- Identical to **OI_PLAY_ONCE**.

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_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 looses all his resources (BGM call).

OICS_START:

- Indicates that the mentioned call has started.

OICS_STARTCHIME:

- Indicates that the mentioned call is busy with its start 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 end chime.

OICS_END:

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

OICS_REPLAY:

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

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 no longer valid after this notification.

13.2.7 TOICallEndReason

The **TOICallEndReason** type defines possible stop and abort reasons for an ended call. This type is returned as a property by the **getCallEndReason()** function supplied in the **OIDET_CALLENDV2** event type The function **isCallAborted()** 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 TOIResourceState

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

13.2.9 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 Praesideo 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.

13.2.10 TOIVirtualControlInput Deactivation

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.

13.2.11 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).

13.2.12 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. Not supported members will throw an exception.
Note that later versions of Praesideo will most likely send newer (other) types. The application should check and report this, so it can be adapted to the new situation.

13.2.12.1 Unknown Diagnostic Event-Group Event-types

OIDET_UNKNOWNDIAGEVENTTYPE:

- Indicates that the Praesideo system was unable to determine the type of the diagnostic event.

13.2.12.2 Call Diagnostic Event-Group Event-types

OIDET_CALLSTARTV2:

- Indicates that the diagnostic event represents the start of a call in the Praesideo system.

OIDET_CALLENDV2:

- Indicates that the diagnostic event represents the end (or abort) of a call in the Praesideo 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.

13.2.12.3 General Diagnostic Event-Group Event-types

OIDET_UNITCONNECT:

- Indicates that the diagnostic event represents a unit that has connected to or disconnected from the Praesideo system.

OIDET_BOOSTERSPARESWITCH:

- Indicates that the diagnostic event represents that a booster is switched over to its associated spare booster. In general this diagnostic events is related to supervision fault diagnostic events described in section 13.2.12.4.

OIDET_BOOSTERSPARESWITCHRETURN:

- Indicates that the diagnostic event represents that a booster is operational again and that the switchover to the spare is ended.

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 enters the emergency state.

OIDET_NCSTARTUP:

- Indicates that the diagnostic event represents that the Praesideo system has started.

OIDET_OPENINTERFACECONNECT:

- Indicates that the diagnostic event represents that a remote system has connected to the Praesideo system using the open interface.

OIDET_OPENINTERFACEDISCONNECT:

- Indicates that the diagnostic event represents that a remote system has disconnected from the Praesideo system using the open interface.

OIDET_OPENINTERFACECONNECTFAILED:

- Indicates that the diagnostic event represents that a remote system has attempted to connect to the Praesideo system using the open interface but failed.

OIDET_MOSTHALFPOWERMODESTART:

- Indicates that the diagnostic event represents that the Praesideo network is operating in half power mode.

OIDET_MOSTHALFPOWERMODEEND:

- Indicates that the diagnostic event represents that the Praesideo network is operating on full power (default) mode. Note that on start of the Praesideo system the full power mode is operational, but is not represented in a diagnostic event. Only the switch to and from half power mode is reported.

OIDET_ACKNOWLEDGEMENT (deprecated):

- Indicates that the diagnostic event represents an acknowledgement of a specific fault event. This event-type is never sent by the Praesideo system version 2.30 and onwards.

OIDET_RESOLVE (deprecated):

- Indicates that the diagnostic event represents a resolve of a specific fault event. This event-type is never sent by the Praesideo system version 2.30 and onwards.

OIDET_RESET (deprecated):

- Indicates that the diagnostic event represents a reset of a specific fault event. This event-type is never sent by the Praesideo system version 2.30 and onwards.

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_BOOSTERSPARESWITCH2:

- Indicates that the diagnostic event represents that an Audio Output on a Multi-Channel Interface is switched over to its associated spare. In general this diagnostic event is related to supervision fault diagnostic events described in section 13.2.12.4.

OIDET_BOOSTERSPARESWITCHRETURN2:

- Indicates that the diagnostic event represents that an Audio Output on a Multi-Channel Interface is operational again and that the switchover to the spare has ended.

OIDET_USERLOGIN:

- Indicates that the diagnostic event represents that a user has logged in on a call station.

OIDET_USERLOGOUT:

- Indicates that the diagnostic event represents that a user has logged out on a call station.

OIDET_USERLOGINFAILED:

- Indicates that the diagnostic event represents that a login attempt on a call station 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.

13.2.12.4 Fault Diagnostic Event-Group Event-types

OIDET_CALLSTATIONEXTENSION:

- Indicates that the diagnostic event represents a mismatch between the number of configured call-station keypads and the number of detected call-station keypads.

OIDET_FLASHCARDCHECKSUM:

- Indicates that the diagnostic event represents a checksum mismatch detection of the prerecorded messages on the flashcard inside the Network Controller.

OIDET_ILLEGALCONFIGURATION:

- Indicates that the diagnostic event represents an inconsistency within the active configuration file.

OIDET_MEMORYERROR:

- Indicates that the diagnostic event represents a memory failure inside either a connected unit or the Network Controller of the Praesideo System.

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_REDUNDANTRINGBROKEN:

- Indicates that the diagnostic event represents detection that the redundant network ring is not closed.

OIDET_UNITMISSING:

- Indicates that the diagnostic event represents a missing configured unit.

OIDET_UNITNOTCONFIGURED:

- Indicates that the diagnostic event represents a detected unit that is not configured.

OIDET_UNITRESET:

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

OIDET_UNITUNKNOWNTYPE:

- Indicates that the diagnostic event represents detection of an unknown unit.

OIDET_USERINJECTEDFAULT:

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

OIDET_WLSBOARD:

- Indicates that the diagnostic event represents a mismatch between a configured and an installed WLS (supervision) board inside a booster.

OIDET_WLSBOARDMAINSPARE:

- Indicates that the diagnostic event represents a mismatch between a main booster (power amplifier) configured WLS (supervision) board and a spare booster installed WLS board. The main booster is the originator of the diagnostic event.

OIDET_FLASHCARDMISSING:

- Indicates that the diagnostic event represents absence of the flashcard.

OIDET_CONFIGURATIONFILE:

- Indicates that the diagnostic event represents detection of a 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_BOOSTERSTANDBY:

- Indicates that the diagnostic event represents a booster remains in standby mode. It refuses to go out of standby mode.

OIDET_AMPFAILURE:

- Indicates that the diagnostic event represents an amplifier failure inside a booster.

OIDET_AMPGROUNDSHORT:

- Indicates that the diagnostic event represents detection of a ground-shorted amplifier.

OIDET_AMPOVERHEAT:

- Indicates that the diagnostic event represents detection of an overheated amplifier.

OIDET_AMPOVERHEATMUTE:

- Indicates that the diagnostic event represents detection of an overheated amplifier, whereby the output is muted.

OIDET_AMPOVERLOAD:

- Indicates that the diagnostic event represents detection of an overloaded amplifier.

OIDET_AMPSHORTCIRCUIT:

- Indicates that the diagnostic event represents detection of a short-circuited amplifier.

OIDET_AUDIOPATHSUPERVISION:

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

OIDET_EOLFAILURE:

- Indicates that the diagnostic event represents detection of the loss of the End-of Line board.

OIDET_MICROPHONESUPERVISION:

- Indicates that the diagnostic event represents detection of microphone failure.

OIDET_LINEINPUTSUPERVISION:

- Indicates that the diagnostic event represents detection of a line input failure.

OIDET_POWERBACKUPSUPPLY:

- Indicates that the diagnostic event represents detection of loss of the backup power supply.

OIDET_POWERMAINSUPPLY:

- Indicates that the diagnostic event represents detection of loss of the mains power supply.

OIDET_SYSTEMINPUTCONTACT:

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

OIDET_LINESHORTCIRCUIT:

- Indicates that the diagnostic event represents detection of a short-circuited output.

OIDET_PILOTTONECALIBRATION:

- Indicates that the diagnostic event represents detection of calibration failure of the pilot tone.

OIDET_AMPFAILUREOROVERLOAD:

- Indicates that the diagnostic event represents detection of either an amplifier failure or an amplifier overload. Note that this diagnostic event is related to a specific unit-type.

OIDET_COBRANETINTERFACE:

- Indicates that the diagnostic event represents detection of a CobraNet board failure.

OIDET_COBRANETNETWORK:

- Indicates that the diagnostic event represents detection of a CobraNet Network failure.

OIDET_ENDOFLINESUPERVISION:

- Indicates that the diagnostic event represents detection of WLS2 end-of-line failure(s).

OIDET_LOUDSPEAKERSUPERVISION:

- Indicates that the diagnostic event represents detection of WLS2 loudspeaker failure(s).

OIDET_INCOMPATIBLEHWVERSION:

- Indicates that the diagnostic event represents a mismatch between the expected HW version of a unit and the detected HW version of a unit. For full support of all functionality inside the Praesideo System, the hardware should be equipped with related features. In case the HW features are not present, this mismatch diagnostic event is generated.

OIDET_REMOTEPOWERSUPPLY:

- Indicates that the diagnostic event represents detection of loss of the main power supply connected to a Remote Call Station.

OIDET_REMOTEBACKUPSUPPLY:

- Indicates that the diagnostic event represents detection of loss of the backup power supply connected to a Remote Call Station.

OIDET_REMOTECONNECTION:

- Indicates that the diagnostic event represent detection of connection loss between the base unit and the Remote Call Station.

OIDET_POWERBACKUPSUPPLY2:

- Indicates that the diagnostic event represents detection of loss of the backup power supply for a Basic Amplifier.

OIDET_POWERMAINSSUPPLY2:

- Indicates that the diagnostic event represents detection of loss of the mains power supply for a Basic Amplifier.

OIDET_GROUPAFAULT:

- Indicates that the diagnostic event represents a failure in group A for audio outputs with A/B switching or class-A wiring.

OIDET_GROUPBFAULT:

- Indicates that the diagnostic event represents a failure in group B for audio outputs with A/B switching or class-A wiring.

OIDET_CLASSASWITCHOVER:

- Indicates that the diagnostic event represents detection of closure of the second (B) relay in class-A mode.

OIDET_HUNDREDVLINEFAULT:

- Indicates that the diagnostic event represents detection of a 100VlineFault in A/B wiring mode while determining whether a **GroupAFault** or **GroupBFault** must be generated.

OIDET_WLS2CCBSYNC:

- Indicates that the diagnostic event represents a Supervision Control Board failure.

OIDET_AMPMISSING:

- Indicates that the diagnostic event represents a missing Basic Amplifier channel.

OIDET_INVALIDFWVERSION:

- Indicates that the diagnostic event represents a mismatch between the expected FW version of a unit and the detected FW version of a unit.

OIDET_REDUNDANTPOWERFAULT:

- Indicates that the diagnostic event represents detection of an internal fault.

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 Praesideo System.

OIDET_OMNEOINTERFACE:

- Indicates that the diagnostic event represents detection of an OMNEO board failure.

OIDET_OMNEONETWORK:

- Indicates that the diagnostic event represents detection of an OMNEO Network failure.

OIDET_AMPFANFAULT:

- Indicates that the diagnostic event represents an amplifier fan fault.

13.2.13 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 13.2.10, presented as sub-sections.

OIDEG_UNKNOWNDIAGEVENTGROUP:

- Indicates that the diagnostic event couldn't be grouped in one of the groups below.

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.

13.2.14 TOIEventOriginatorType

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

OIEOT_NOEVENTORIGINATOR:

- Indicates that the event originator is not known.

OIEOT_UNITEVENTORIGINATOR:

- Indicates that the event originator is a unit.

OIEOT_OPENINTERFACEEVENTORIGINATOR:

- Indicates that the event originator is a system connected to the open interface of the Praesideo system.

OIEOT_CONTROLINPUTEVENTORIGINATOR:

- Indicates that the event originator is a control-input.

OIEOT_AUDIOOUTPUTEVENTORIGINATOR:

- Indicates that the event originator is an audio-output.

OIEOT_AUDIOINPUTEVENTORIGINATOR:

- Indicates that the event originator is an audio input.

OIEOT_UNITMENUEVENTORIGINATOR:

- Indicates that the event originator is a user of the rotary control on the network controller.

OIEOT_USEREVENTORIGINATOR:

- Indicates that the event originator is a user.

13.2.15 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.

13.2.16 TOIUnitType

The **TOIUnitType** type represents the kind of unit on which the originator is located. Note that each unit-type may have multiple inputs and/or output (e.g. keys on a CST type unit) which can trigger an event.

OIUT_CST:

- Indicates that the event originator is a Call-Station unit.

OIUT_NC:

- Indicates that the event originator is a Network Controller unit.

OIUT_BST:

- Indicates that the event originator is a Booster unit (power amplifier).

OIUT_AIO:

- Indicates that the event originator is an Audio Expander unit.

OIUT_POFGOF:

- Indicates that the event originator is a Fiber Interface unit.

OIUT_CEX:

- Indicates that the event originator is a CobraNet Interface unit.

OIUT_MCI:

- Indicates that the event originator is a Multi-Channel Interface unit.

OIUT_CRF:

- Indicates that the event originator is a Call Stacker unit.

OIUT_OMI:

- Indicates that the event originator is an OMNEO interface unit.

OIUT_UNKNOWN:

- Indicates that the event originator is an unknown unit.

13.2.17 TOIChipType

The **TOIChipType** type represents the processor type related to the **UNITRESET** event-type.

OICT_MMP:

- Indicates that the event is related to the unit processor (processor inside each unit connected to the Praesideo network).

OICT_MAINCPU:

- Indicates that the event is related to the Main processor (Major processor inside the Network Controller).

OICT_CNM:

- Indicates that the event is related to the CobraNet Module.

OICT_CCB:

- Indicates that the event is related to a Supervision Control Board.

OICT_REMOTE:

- Indicates that the event is related to the remote unit connected to its base unit (e.g. Remote Call Station).

OICT_OMNEO:

- Indicates that the event is related to the OMNEO Module.

OICT_UNKNOWN:

- Indicates that the event is related to an unknown processor.

13.2.18 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 Praesideo System. This action type is passed for each diagnostic event already in the Praesideo System after subscription for the events.

OIACT_EXISTING_LAST:

- The specified diagnostic event is already present in the Praesideo System and it is the last present event sent, or there are actually no fault events present in the Praesideo System, in which case the specified diagnostic event is of type **OIDET_NOFAULTS**.

13.2.19 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.

OICOH_STACKED:

- Stacked calls are calls that extend partial calls with replays to previously unavailable zones.

13.2.20 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

13.2.21 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.

13.2.22 TOICallStackingTimeout

Defines the limit of time for stacked call broadcasting.

OICST_INFINITE:

- Wait infinitely for zones to become available for broadcasting.

13.3 PraesideoOpenInterface Methods

13.3.1 connect

Make a connection with an NCO. A connection is required before many other methods (like **startCall**) can be used. Note that the port parameter is no longer used. The connection is always done using the port number 9401.

```
Sub connect(ip As String, port As Long, username As String, password As String)
```

figure 13.1: connect

Parameters:

- **ip**
  IP address of the NCO, format "**192.168.0.15**" or the DNS name of the network controller.
- **port**
  Not used any more. Remains in the interface for backward compatibility.
- **username**
  Name of the user as defined during the "User Management" configuration of the Praesideo system.
- **password**
  Password of the user.

Error codes:

- **OIERROR_OK**
- **OIERROR_UNABLE_TO_MAKE_CONNECTION**
- **OIERROR_BAD_CREDENTIALS**
- **OIERROR_ALREADY_LOGGED_IN**
- **OIERROR_INVALID_PARAMETERS**

13.3.2 disconnect

Gracefully terminates a connection with the NCO. After any call to this function it is no longer possible to use most functions of the OI. The only error that is returned indicates that there is no connection.

```
Sub disconnect( )
```

figure 13.2: disconnect

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**

13.3.3 getVersion

Retrieves the version of the OIL. This funtion can also be called when there is no connection.

```
Function getVersion( ) As String
```

figure 13.3: getVersion

Return value:

- **Version**
  Version of the open interface, not of the Praesideo system!

Error codes:

- **OIERROR_OK**
- **OIERROR_BUFFER_TOO_SMALL**
  Internal **COM**-server to Praesideo system error, format conversion between **COM**-server and Praesideo system operates with a too small buffer, should not occur).

13.3.4 getNcoVersion

Retrieves the version of the network controller.

```
Function getNcoVersion( ) As String
```

figure 13.4: getNcoVersion

Return value:

- **Version**
  Version of the connected Praesideo system (i.e. the network controller).

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_BUFFER_TOO_SMALL**
  Internal **COM**-server to Praesideo system error, format conversion between **COM**-server and Praesideo system operates with a too small buffer, should not occur.

13.3.5 createCallEx2

Create (but do not start) a call with the given parameters.

```
Function createCallEx2 (routing As String, priority As Long,
outputHandling As TOICallOutputHandling, stackingMode As TOICallStackingMode,
stackingTimeout As Long, startChime As String, endChime As String,
bLiveSpeech As Boolean, audioInput As String, messages As String,
repeat As Long, callTiming As TOICallTiming,
preMonitorDest As String, liveSpeechAtt As Long, startChimeAtt As Long,
endChimeAtt As Long, messageAtt As Long) As Long
```

figure 13.5: createCallEx2

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.
  Zone active outputs should not be used in time shifted calls.
- **priority**
  The priority of the call. See section 13.2.4 for the value range definitions. Note that call is always partial for BGM and Emergency calls, 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**. 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 are 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.
- **bLiveSpeech**
  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 section 13.2.5). 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**.
- liveSpeechAtt
  The attenuation to be used for the audio input during the live speech phase. Range: 0..60 dB.
- startChimeAtt
  The attenuation to be used for the chime generator during the start chime phase. Range: 0..60 dB.

- endChimeAtt
  The attenuation to be used for the chime generator during the end chime phase. Range: 0..60 dB.
- messageAtt
  The attenuation to be used for the message generator during the prerecorded message phase.
  Range: 0..60 dB.

Return value:

- **callId**
  Unique identification of the call (only valid when Error code is **OIERROR_OK**).

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.6 createCallEx

Create (but do not start) a call with the given parameters.. This function is deprecated since Praesideo 3.4. Please use createCallEx2 instead.

```
Function createCallEx (routing As String, priority As Long,
outputHandling As TOICallOutputHandling, stackingMode As TOICallStackingMode,
stackingTimeout As Long, startChime As String, endChime As String,
bLiveSpeech As Boolean, audioInput As String, messages As String,
repeat As Long, callTiming As TOICallTiming,
preMonitorDest As String) As Long
```

figure 13.6: createCallEx

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 section 13.2.4 for the value range definitions. Note that call is always partial for BGM and Emergency calls, 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**. 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 are 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.
- **bLiveSpeech**
  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 section 13.2.5). 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**.

Return value:

- **callId**
  Unique identification of the call (only valid when Error code is **OIERROR_OK**).

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

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.

```
Sub startCreatedCall(callId As Long)
```

figure 13.7: startCreatedCall

Parameters:

- **callId**
  unique identification of the call, returned by **createCallEx2**.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_CALL_NO_LONGER_EXISTS**
- **OIERROR_INTERNAL_ERROR**

13.3.8 startCall

Start a call with the given parameters. If the call was started successfully, call state update events for this call will be sent via the Open interface. This method is deprecated. Please use **createCallEx2** and **startCreatedCall** instead.

Note
Both call and resource state update events can be sent for the started call, even before the startCall() method has returned. Because the callId for the started call is unknown to the caller, until the startCall() method has returned, both call and resource state update events should be queued when calling the startCall() method and processed when the method has returned. This is to ensure these update events can be evaluated properly.

```
Function startCall(routing As String, priority As Long, bPartial As Boolean,
startChime As String, endChime As String, bLiveSpeech As Boolean, audioInput As
String, messages As String, repeat As Long) As Long
```

figure 13.8: startCall

Parameters:

- **routing**
  List of names of zone groups, zones and/or control outputs. The routing is formatted as a comma separated list of resource names.
- **priority**
  The priority of the call. See section 13.2.4 for the value range definitions. Note that call is always partial for BGM and Emergency calls, regardless the partial setting.
- **partial**
  Whether or not the call is partial. A partial call accepts extension and removal of zones to/from the routing. **True** = partial, **False** = not partial. Calls with BGM priority (0 - 31) or emergency priority (224 - 255) will always be processed as partial, irrespective of partial being **True** or **False**.
- **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 pre-recorded messages. The messages parameter is formatted as a comma separated list 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 section 13.2.5). Otherwise, the following range can be used: **1 .. 32767**.

Return values:

- **callId**
  unique identification of the call (only valid when Error code is **OIERROR_OK**).

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.9 stopCall

Stop a previously created or started call.

```
Sub stopCall(callId As Long)
```

figure 13.9: stopCall

Parameters:

- **callId**
  unique identification of the call, returned by **createCallEx** or **createCall**.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_CALL_NO_LONGER_EXISTS**
- **OIERROR_INTERNAL_ERROR**

13.3.10 abortCall

Abort a previously created or started call.

```
Sub abortCall(callId As Long)
```

figure 13.10: abortCall

Parameters:

- **callId**
  unique identification of the call, returned from **createCallEx** or **createCall**.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_CALL_NO_LONGER_EXISTS**
- **OIERROR_INTERNAL_ERROR**

13.3.11 addToCall

Add routing to a previously created or started call.

```
Sub addToCall(callId As Long, routing As String)
```

figure 13.11: addToCall

Parameters:

- **callId**
  Unique identification of the call, returned from **createCallEx** or **createCall**.
- **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.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_CALL_NO_LONGER_EXISTS**
- **OIERROR_INTERNAL_ERROR**

13.3.12 removeFromCall

Remove routing from a previously created or started call.

```
Sub removeFromCall(callId As Long, routing As String)
```

figure 13.12: removeFromCall

Parameters:

- **callId**
  Unique identification of the call, returned from **createCallEx** or **createCall**.
- **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.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_CALL_NO_LONGER_EXISTS**
- **OIERROR_INTERNAL_ERROR**

13.3.13 cancelAll

Cancel all available stacked calls that were started by this connection.

```
Sub cancelAll()
```

figure 13.13: cancelAll

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.14 cancelLast

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

```
Sub cancelAll()
```

figure 13.14: cancelLast

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.15 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 result in a call to **faultAlarmState** method on the open interface event interface.

```
Sub ackAllFaults()
```

figure 13.15: ackAllFaults

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.16 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 result in a call to **faultAlarmState** method on the open interface event interface.

```
Sub resetAllFaults( )
```

figure 13.16: resetAllFaults

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.17 ackEvacAlarm

Acknowledges the emergency alarm. If the emergency alarm changes state, this will result in a call to **evacAlarmState** method on the open interface event interface.

```
Sub ackEvacAlarm( )
```

figure 13.17: ackEvacAlarm

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.18 resetEvacAlarmEx

Resets the emergency alarm. If the emergency alarm changes state, this will result in a call to **evacAlarmState** method on the Open Interface event interface.

```
Sub resetEvacAlarmEx(bAbortEvacCalls as Boolean)
```

figure 13.18: resetEvacAlarmEx

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, False

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.19 resetEvacAlarm

Resets the emergency alarm. If the emergency alarm changes state, this will result in a call to **evacAlarmState** method on the open interface event interface. Running evacuation priority calls will be aborted.

```
Sub resetEvacAlarm( )
```

figure 13.19: resetEvacAlarm

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.20 setDateAndTime

Sets the date and time.

```
Sub setDateAndTime(year As Long, month As Long, day As Long, hour As Long,
minute As Long, second As Long)
```

figure 13.20: setDateAndTime

Parameters:

- **year**
  Year of the new date. Value range **1970 .. 2037.**
- **month**
  Month of the new date. Value range **1 .. 12**.
- **day**
  Day of the new date. Value range **1 .. 31.**
- **hour**
  Hour of the new time. Value range **0 .. 23.**
- **minute**
  Minute of the new time. Value range **0 .. 59**.
- **second**
  Second of the new time. Value range **0 .. 59**.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.21 registerClientInterface

Register the client interface. This is required to receive state updates through the Open Interface, using the Custom Callbacks technique (12.3.2). This has to be used making a C++ console application to interface with the Praesideo system, where the Connection point technique cannot be used.

```
Sub registerClientInterface (pUnk As Unknown)
```

figure 13.21: registerClientInterface

Parameters:

- **pUnk**
  Reference to a custom callback instance of the events sink. Most times " 'Me' can be used to indicate that the calling object implements the sink itself.

Error codes:

- **OIERROR_OK**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_REGISTERED**

13.3.22 deregisterClientInterface

Deregisters the client interface. When this function is called successfully, no more state updates will be sent to the Open interface client via the custom callback interface. This function is required, when the Custom Callbacks technique is used.

```
Sub deregisterClientInterface ()
```

figure 13.22: deregisterClientInterface

Error codes:

- **OIERROR_OK**
- **OIERROR_NOT_REGISTERED**

13.3.23 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** method will be called on the open interface event interface client with the current state of that resource.

```
Sub setSubscriptionResources(subscription As Boolean, resources As String)
```

figure 13.23: setSubscriptionResources

Parameters:

- **subscription**
  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.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.24 setSubscriptionResourceFaultState

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** method will be called on the Open Interface event interface client with the current state of that resource.

```
Sub setSubscriptionResourceFaultState(subscription As Boolean,
resources As String)
```

figure 13.24: setSubscriptionResourceFaultState

Parameters:

- **subscription**
  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.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.25 setSubscriptionFaultAlarm

Subscribe or unsubscribe the open interface client to fault 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, **faultAlarmState** method will be called on the open interface event interface client with the current state of the fault alarm.

Sub **setSubscriptionFaultAlarm**(subscription As Boolean)

figure 13.25: setSubscriptionFaultAlarm

Parameters:

- **subscription**
  Whether to subscribe or unsubscribe.
  True = subscribe, **False** = unsubscribe.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.26 setSubscriptionEvacAlarm

Subscribe or unsubscribe the open interface client to emergency alarm state updates. Only when a subscription is set for the emergency alarm, emergency alarm state updates will be sent. When a subscription is set, **evacAlarmState** will be called on the open interface client with the current state of the emergency alarm.

Sub **setSubscriptionEvacAlarm**(subscription As Boolean)

figure 13.26: setSubscriptionEvacAlarm

Parameters:

- **subscription**
  Whether to subscribe or unsubscribe.
  True = subscribe, **False** = unsubscribe.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.27 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.

Sub **setBgmRouting**(channel As String, routing As String)

figure 13.27: setBgmRouting

Parameters:

- **channel**
  Name of the BGM channel.
- **routing**
  List of names of zone groups and/or zones.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.28 addBgmRouting

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

Sub **addBgmRouting**(channel As String, routing As String)

figure 13.28: addBgmRouting

Parameters:

- **channel**
  Name of the BGM channel.
- **routing**
  List of names of zone groups and/or zones.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.29 removeBgmRouting

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

Sub **removeBgmRouting**(channel As String, routing As String)

figure 13.29: removeBgmRouting

Parameters:

- **channel**
  Name of the BGM channel.
- **routing**
  List of names of zone groups and/or zones.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.30 toggleBgmRouting

Toggles routing in a BGM channel. When none of the 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.

Sub **toggleBgmRouting** (channel As String, routing As String)

figure 13.30: toggleBgmRouting

Parameters:

- **channel**
  name of the BGM channel.
- **routing**
  list of names of zone groups and/or zones.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.31 setBgmVolume

Sets the BGM volume of routing.

Sub **setBgmVolume**(routing As String, volume As Long)

figure 13.31: setBgmVolume

Parameters:

- **routing**
  List of names of zone groups and/or zones.
- **volume**
  Volume. Range: 0 .. -96 (dB). Use -96 (dB) to mute the BGM.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.32 incrementBgmVolume

Increments the BGM volume of routing with 3 dB.

Sub **incrementBgmVolume**(routing As String)

figure 13.32: incrementBgmVolume

Parameters:

- **routing**
  List of names of zone groups and/or zones.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.33 incrementBgmChannelVolume

Increments the BGM volume of a channel with 3 dB.

Sub **incrementBgmChannelVolume** (channel As String)

figure 13.33: incrementBgmChannelVolume

Parameters:

- **channel**
  BGM channel name

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.34 decrementBgmVolume

Decrement the BGM volume of routing with 3 dB.

Sub **decrementBgmVolume**(routing As String)

figure 13.34: decrementBgmVolume

Parameters:

- **routing**
  List of names of zone groups and/or zones.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.35 decrementBgmChannelVolume

Decrements the BGM volume of a channel with 3 dB.

Sub **decrementBgmChannelVolume** (channel As String)

figure 13.35: decrementBgmChannelVolume

Parameters:

- **channel**
  BGM channel name

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.36 setSubscriptionBgmRouting

Subscribes or unsubscribes the OICI 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, bgmRouting will be called on the OICI with the current routing of that BGM channel and the addition parameter set to true.

Sub **setSubscriptionBgmRouting**(subscription as Boolean, channel as String)

figure 13.36: setSubscriptionBgmRouting

Parameters:

- **subscription**
  Whether to subscribe or unsubscribe.
- **channel**
  name of the BGM channel.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.37 reportFault

Reports a fault diagnostics event in the system. The fault will be reported as a **USERINJECTEDFAULT**.

```
Function reportFault(description As String) As Long
```

figure 13.37: reportFault

Parameters:

- **description**
  Textual representation of the fault to be reported.

Return value:

- **eventId**
  Identification of the diagnostic fault event reported.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.38 resolveFault

Resolve a specific diagnostic fault event. The received **eventId** of the **reportFault** function or a diagnostic event should be used as parameter.

```
Sub resolveFault(eventId As Long)
```

figure 13.38: resolveFault

Parameters:

- **eventId**
  Identification of the diagnostic fault event, received by the **reportFault** function.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.39 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 result in a call to **faultAlarmState** method on the Open Interface event interface.

```
Sub ackFault(eventId As Long)
```

figure 13.39: ackFault

Parameters:

- **eventId**
  Identification of the diagnostic fault event.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.40 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 result in a call to **faultAlarmState** method on the Open Interface event interface.

```
Sub resetFault(eventId As Long)
```

figure 13.40: resetFault

Parameters:

- **eventId**
  Identification of the diagnostic fault event.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.41 activateVirtualControlInput

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

```
Sub activateVirtualControlInput(virtualControlInput As String)
```

figure 13.41: activateVirtualControlInput

Parameters:

- **virtualControlInput**
  Name of the virtual control input to activate.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.42 deactivateVirtualControlInput

Deactivate a virtual control input. If the virtual control input is already inactive then deactivating it again will

```
Sub deactivateVirtualControlInput(virtualControlInput As String,
                                  deactivationType as TOIVirtualControlInputDeactivation)
```

figure 13.42: deactivateVirtualControlInput

Parameters:

- **virtualControlInput**
  Name of the virtual control input to deactivate.
- **deactivationType**
  Specifier how the associated action should be deactivated (see section 13.2.10).

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.43 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 **diagEvent** method will be called on the Open Interface event interface client with the diagnostic event of that group.

```
Sub setSubscriptionEvents(subscription As Boolean, eventGroup As
TOIDiagEventGroup)
```

figure 13.43: setSubscriptionEvents

Parameters:

- **subscription**
  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 section 13.2.12.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.44 setSubscriptionBgmVolume

Subscribes or unsubscribes the OICI to BGM volume updates. Only when a subscription is set for a BGM zone, BGM volume updates will be sent for that BGM zone. When a subscription is set for a BGM zone, bgmVolume will be called on the OICI with the current volume of that BGM zone.

Sub **setSubscriptionBgmVolume** (subscription as Boolean,zone as String)

figure 13.44: setSubscriptionBgmVolume

Parameters:

- **subscription**
  Whether to subscribe or unsubscribe.
- **zones**
  Comma (,) separated list of zone names. Note that it is not possible to subscribe to the BGM volume of zone groups.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_ BY_SERVER**

13.3.45 setSubscriptionUnitCount

Subscribe or unsubscribe the Open Interface client to MOST unit count updates. Only when a subscription is set for the unit count, unit count updates will be sent. When a subscription is set, unitCount method will be called on the Open Interface event interface client with the current number of connected MOST units.

Sub **setSubscriptionUnitCount**(subscription As Boolean)

figure 13.45: setSubscriptionUnitCount

Parameters:

- **subscription**
  Whether to subscribe or unsubscribe. True = subscribe, False = unsubscribe.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.46 setSubscriptionVirtual ControlInputs

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, **virtualControlInputState** method will be called on the Open Interface event interface client with the current state of the virtual control inputs.

```
Sub setSubscriptionVirtualControlInputState(subscription As Boolean,
                                            virtualControlInputs As String)
```

figure 13.46: setSubscriptionVirtualControlInputs

Parameters:

- **subscription**
  Whether to subscribe or unsubscribe. True = subscribe, False = unsubscribe.
- **virtualControlInputs**
  List of names of virtual control inputs. A comma separates each name in the list.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**

13.3.47 getZoneNames

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

```
Function getZoneNames (zoneGroup as String) As String
```

figure 13.47: getZoneNames

Parameters:

- **zoneGroup**
  the zone group to get the names of (empty string for all zones).

Return value:

- **Names**
  Comma (,) separated list of zone names.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_ BY_SERVER**

13.3.48 getZoneGroupNames

Retrieve the configured zone group names from the Praesideo system.

```
Function getZoneGroupNames () As String
```

figure 13.48: getZoneGroupNames

Return value:

- **Names**
  Comma (,) separated list of zone group names.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_ BY_SERVER**

13.3.49 getMessageNames

Retrieve the configured message names from the Praesideo system.

figure 13.49: getMessageNames

Return value:

- **Names**

Comma (,) separated list of message names.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.50 getChimeNames

Retrieve the available chime names from the Praesideo system.

figure 13.50: getChimeNames

Return value:

- **Names**

Comma (,) separated list of chime names.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.51 getAudioInputNames

Retrieve the audio input names from the Praesideo system.

figure 13.51: getAudioInputNames

Return value:

- **Names**

Comma (,) separated list of audio input names.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.52 getBgmChannelNames

Retrieve the configured message names from the Praesideo system.

figure 13.52: getBgmChannelNames

Return value:

- **Names**

Comma (,) separated list of BGM channel names.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.53 getVirtualControlInputNames

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

figure 13.53: getVirtualControlInputNames

Return value:

- **Names**
  Comma (,) separated list of virtual control input names.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.54 getConfigId

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

figure 13.54: getConfigId

Return value:

- **ConfigId**
  Configuration identifier.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.55 getConfiguredUnits

Retrieve the configured units from the Praesideo system.

figure 13.55: getConfiguredUnits

Return value:

- **Names**
  Comma (,) separated list of unit names with serial number, formatted as name(serialnumber).

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.3.56 getConnectedUnits

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

figure 13.56: getConnectedUnits

Return value:

- **Names**
  Comma (,) separated list of unit names with serial number, formatted as name(serialnumber).

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_SERVER**

13.4 PraesideoOpenInterfaceEvents

The alias for this interface is **IPraesideoOpenInterfaceEvents_Vxxx**.

13.4.1 resourceState

Will be called when the state of resources (zone groups, zones, control outputs) change.

Event **resourceState**(ByVal resources As String, ByVal state As TOIResourceState, ByVal priority As Long, ByVal callId As Long)

figure 13.57: resourceState

Parameters:

- **resources**
  List of names of zone groups, zones and/or control outputs. A comma separates each name in the routing list.
- **state**
  State of the resource. See section 13.2.7 for the definitions of the resource states.
- **priority**
  Priority of the call using the resource (when state is activated). See section 13.2.4 for the priority value ranges.
- **callId**
  Unique identification of the call (when state is activated).

Note
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).

13.4.2 resourceFaultState

Will be called when the fault state of resources (zone groups, zones) change.

Event **resourceFaultState**(ByVal resources As String, faultState as TOIResourceFaultState)

figure 13.58: resourceFaultState

Parameters:

- **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
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).

13.4.3 faultAlarmState

Will be called when the state of the fault alarm changes.

Event **faultAlarmState**(ByVal state As TOIAlarmState)

figure 13.59: faultAlarmState

Parameters:

- **state**
  State of the fault alarm. See section 13.2.3 for the definitions of the fault alarm states.

13.4.4 evacAlarmState

Will be called when the state of the emergency alarm changes.

Event **evacAlarmState**(ByVal state As TOIAlarmState)

figure 13.60: evacAlarmState

Parameters:

- **state**
  state of the emergency alarm. See section 13.2.3 for the definitions of the emergency alarm states.

13.4.5 callState

Will be called when the state of any call started changes.

```
Event callState(ByVal callId As Long, ByVal state As TOICallState)
```

figure 13.61: callState

Parameters:

- **callId**
  Unique identification of the call.
- **state**
  State of the call. See section 13.2.6 for the definitions of the call states.

13.4.6 bgmRouting

Will be called when the routing of a BGM channel changes.

```
Event bgmRouting(ByVal channel As String, ByVal addition As Boolean,
ByVal routing As String)
```

figure 13.62: bgmRouting

Parameters:

- **channel**
  Name of the BGM channel.
- **addition**
  Whether the routing was added (**true**) or removed (**false**).
- **routing**
  List of names of zone groups, zones and/or control outputs.

13.4.7 connectionBroken

Will be called when the connection with the network 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 connectionBroken( )
```

figure 13.63: connectionBroken

Note
Note that this event is also triggered when the Praesideo System detects a message transmission buffer overflow due to too slow reception by the application.

13.4.8 diagEvent

Will be called when a diagnostic event is logged inside the NCO. See section 14.1 for use of the diagEvent.

```
Event diagEvent(eventId As Long, action As TOIActionType, pDiagEvent As Object)
```

figure 13.64: diagEvent

Parameters:

- **eventId**
  Identification of the diagnostic event passed.
- **action**
  Action done on the event.
- **pDiagEvent**
  Interface reference to a IdiagEvent object.

13.4.9 bgmVolume

Will be called when the volume of a BGM zone changes.

```
Event bgmVolume (ByVal zone As String, ByVal volume As Long)
```

figure 13.65: bgm Volume

Parameters:

- **zone**
  name of the BGM zone.
- **volume**
  The new volume of the zone.

13.4.10 unitCount

Will be called when the number of connected MOST units changes.

```
Event unitCount(ByVal numberConnected As Long)
```

figure 13.66: unitCount

Parameters:

- **numberConnected**
  the number of connected MOST units

13.4.11 virtualControlInputState

Will be called when the state of one or more virtual control inputs changes.

```
Event virtualControlInputState(ByVal virtualControlInputs As String,
                               ByVal state As TOIVirtualControlInputState )
```

figure 13.67: virtualControlInputState

Parameters:

- **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 section 13.2.11 for the definitions of the virtual input contact states.

13.5 IDiagEvent Methods

13.5.1 getEventType

Get function for the diagnostic event type as described in section 13.2.12.
Note that later versions of Praesideo will most likely send newer (other) types. The application should check and report this, so it can be adapted to the new situation.

```
Function getEventType( ) As TOIDiagEventType
```

figure 13.68: getEventType

Return value:

- **diagEventType**
  The type of the diagnostic event. See section 13.2.12 for the definitions of the diagnostic types.

Related Event types:

- All events in all event groups.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.2 getEventGroup

Get function for the group whereto the diagnostic event belongs to.

figure 13.69: getEventGroup

Return value:

- **diagEventGroup**
  The group identification of the diagnostic event. See section 13.2.13 for the definitions of the diagnostic groups.

Related Event types:

- All events in all event groups.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.3 getEventId

Get function for the event identification. The event identification number can be used to acknowledge the event in case it is a fault event.

figure 13.70: getEventId

Return value:

- **eventId**
  Event identification number.

Related Event types:

- All events in all event groups.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.4 getEventState

Get Function for the state of the received event.

figure 13.71: getEventState

Return value:

- **eventState**
  State of the diagnostic event. See section 13.2.15 for the definitions of the diagnostic event state.

Related Event types:

- All events in all event groups.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.5 getEventAddTime

Get function for the date and time information when the diagnostic event is added to the system.

figure 13.72: getEventAddTime

Return value:

- **addDate**
  Date and time when the event was added.

Related Event types:

- All events in all event groups.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.6 getEventAckTime

Get function for the date and time information when the diagnostic event is acknowledged.

figure 13.73: getEventAckTime

Return value:

- **ackDate**
  Date and time when the event was acknowledged.

Related Event types:

- All events in event-group **OIDEG_FAULTEVENTGROUP**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.7 getEventResolveTime

Get function for the date and time information when the diagnostic event is resolved.

figure 13.74: getEventResolveTime

Return value:

- **resolveDate**
  Date and time when the event was resolved.

Related Event types:

- All events in event-group **OIDEG_FAULTEVENTGROUP**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.8 getEventResetTime

Get function for the date and time information when the diagnostic event is reset.

figure 13.75: getEventResetTime

Return value:

- **resetDate**
  Date and time when the event was reset.

Related Event types:

- All events in event-group **OIDEG_FAULTEVENTGROUP**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.9 getAddEventOriginatorObject

Get function for the originator that has added / created the event to the system.

figure 13.76: getAddEventOriginatorObject

Return value:

- **addOrg**
  Originator of diagnostic event. The object should be casted to a IEventOriginator_xxx interface.
  See section 13.6 for the available functions on the originator interface.

Related Event types:

- All events in all event groups.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.10 getAcknowledgeEventOriginatorObject

Get function for the originator that has acknowledged the diagnostic event.

figure 13.77: getAcknowledgeEventOriginatorObject

Return value:

- **ackOrg**
  Originator object of diagnostic event. The object should be casted to a IEventOriginator_xxx interface. See section 13.6 for the available functions on the originator interface.

Related Event types:

- All events in event-group **OIDEG_FAULTEVENTGROUP**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.11 getResolveEventOriginatorObject

Get function for the originator that has resolved the diagnostic event.

figure 13.78: getResolveEventOriginatorObject

Return value:

- **resolveOrg**
  Originator of diagnostic event. The object should be casted to a IEventOriginator_xxx interface.
  See section 13.6 for the available functions on the originator interface.

Related Event types:

- All events in event-group **OIDEG_FAULTEVENTGROUP**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.12 getResetEventOriginatorObject

Get function for the originator that has reset the diagnostic event.

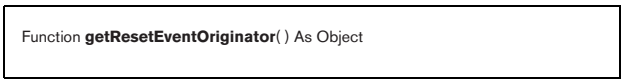

figure 13.79: getResetEventOriginator

- **resetOrg**
  Originator of diagnostic event. The object should be casted to a IEventOriginator_xxx interface.
  See section 13.6 for the available functions on the originator interface.

Related Event types:

- All events in event-group **OIDEG_FAULTEVENTGROUP**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.13 getSviName

Get function for the name of the supervision item.

```
Function getSviName( ) As String
```

figure 13.80: getSviName

Return value:

- **sviName**
  The name of the supervision item.

Related Event types:

- OIDET_AMPFAILURE
- OIDET_AMPFAILUREOROVERLOAD
- OIDET_AMPFANFAULT
- OIDET_AMPGROUNDSHORT
- OIDET_AMPOVERHEAT
- OIDET_AMPOVERHEATMUTE
- OIDET_AMPOVERLOAD
- OIDET_AMPSHORTCIRCUIT
- OIDET_AUDIOPATHSUPERVISION
- OIDET_CLASSASWITCHOVER
- OIDET_COBRANETINTERFACE
- OIDET_COBRANETNETWORK
- OIDET_ENDOFLINESUPERVISION
- OIDET_EOLFAILURE
- OIDET_GROUPAFAULT
- OIDET_GROUPBFAULT
- OIDET_HUNDREDVLINEFAULT
- OIDET_LINEINPUTSUPERVISION
- OIDET_LINESHORTCIRCUIT
- OIDET_LOUDSPEAKERSUPERVISION
- OIDET_MICROPHONESUPERVISION
- OIDET_OMNEOINTERFACE
- OIDET_OMNEONETWORK
- OIDET_PILOTTONECALIBRATION
- OIDET_POWERBACKUPSUPPLY
- OIDET_POWERBACKUPSUPPLY2
- OIDET_POWERMAINSUPPLY
- OIDET_POWERMAINSSUPPLY2
- OIDET_REMOTECONNECTION
- OIDET_REMOTEBACKUPSUPPLY
- OIDET_REMOTEPOWERSUPPLY
- OIDET_SYSTEMINPUTCONTACT
- OIDET_WLS2CCBSYNC

Error codes:

- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE

13.5.14 getSvild

Get function for the identification number of the supervision item.

```
Function getSvild( ) As Long
```

figure 13.81: getSvild

Return value:

- **svild**
  The identification number of the supervision item.

Related Event types:

- OIDET_AMPFAILURE
- OIDET_AMPFAILUREOROVERLOAD
- OIDET_AMPFANFAULT
- OIDET_AMPGROUNDSHORT
- OIDET_AMPOVERHEAT
- OIDET_AMPOVERHEATMUTE
- OIDET_AMPOVERLOAD
- OIDET_AMPSHORTCIRCUIT
- OIDET_AUDIOPATHSUPERVISION
- OIDET_CLASSASWITCHOVER
- OIDET_COBRANETINTERFACE
- OIDET_COBRANETNETWORK
- OIDET_ENDOFLINESUPERVISION
- OIDET_EOLFAILURE
- OIDET_GROUPAFAULT
- OIDET_GROUPBFAULT
- OIDET_HUNDREDVLINEFAULT
- OIDET_LINEINPUTSUPERVISION
- OIDET_LINESHORTCIRCUIT
- OIDET_LOUDSPEAKERSUPERVISION
- OIDET_MICROPHONESUPERVISION
- OIDET_OMNEOINTERFACE
- OIDET_OMNEONETWORK
- OIDET_PILOTTONECALIBRATION
- OIDET_POWERBACKUPSUPPLY
- OIDET_POWERBACKUPSUPPLY2

- OIDET_POWERMAINSUPPLY
- OIDET_POWERMAINSSUPPLY2
- OIDET_REMOTECONNECTION
- OIDET_REMOTEBACKUPSUPPLY
- OIDET_REMOTEPOWERSUPPLY
- OIDET_SYSTEMINPUTCONTACT
- OIDET_WLS2CCBSYNC

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE

13.5.15 getSpareBoosterName

Get function for the spare booster name.

Function **getSpareBoosterName**( ) As String

figure 13.82: getSpareBoosterName

Return value:
- spareBstName
  The name of the spare booster.

Related Event types:
- OIDET_BOOSTERSPARESWITCH
- OIDET_BOOSTERSPARESWITCHRETURN
- OIDET_BOOSTERSPARESWITCH2
- OIDET_BOOSTERSPARESWITCHRETURN2

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE

13.5.16 getSpareBoosterSerialNr

Get function for the serial number of the spare booster.

Function **getSpareBoosterSerialNr**( ) As Long

figure 13.83: getSpareBoosterSerialNr

Return value:
- serialNr
  The serial number of the spare booster.

Related Event types:
- OIDET_BOOSTERSPARESWITCH
- OIDET_BOOSTERSPARESWITCHRETURN
- OIDET_BOOSTERSPARESWITCH2
- OIDET_BOOSTERSPARESWITCHRETURN2

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE

13.5.17 getRemovedResourceNames

Get function for the removed zones, zone-groups and control-outputs from the routing of a call.

Function **getRemovedResourceNames**( ) As String

figure 13.84: getRemovedResourceNames

Return value:

- **resources**
  List of names of zone groups, zones and/or control outputs. A comma separates each name in the routing list.

Related Event types:

- **OIDET_CALLCHANGERESOURCE**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.18 getAddedResourceNames

Get function for the added zones, zone-groups and control-outputs to the routing of a call

Function **getAddedResourceNames**( ) As String

figure 13.85: getAddedResourceNames

Return value:

- **resources**
  List of names of zone groups, zones and/or control outputs. A comma separates each name in the routing list.

Related Event types:

- **OIDET_CALLCHANGERESOURCE**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE**

13.5.19 getCallAudioInputName

Get function for the audio input name (e.g. microphone input).

Function **getCallAudioInputName**( ) As String

figure 13.86: getCallAudioInputName

Return value:

- **inputName**
  Name of the audio Input.

Related Event types:

- **OIDET_CALLSTART**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.20 getCallOutputsNames

Get function for the output names as defined during the start of a call.

```
Function getCallOutputsNames( ) As String
```

figure 13.87: getCallOutputsNames

Return value:

- **resources**
  List of names of zone groups, zones and/or control outputs. A comma separates each name in the routing list.

Related Event types:

- **OIDET_CALLSTART**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.21 getCallStartChimesNames

Get function for the starting chime names as defined during the start of a call.

```
Function getCallStartChimesNames( ) As String
```

figure 13.88: getCallStartChimesNames

Return value:

- **startChime**
  The name of the start chime.

Related Event types:

- **OIDET_CALLSTART**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.22 getCallEndChimesNames

Get function for the ending chime name as defined during the start of a call.

```
Function getCallEndChimesNames( ) As String
```

figure 13.89: getCallEndChimesNames

Return value:

- **endChime**
  The name of the end chime.

Related Event types:

- **OIDET_CALLSTART**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.23 getCallMessagesNames

Get function for the prerecorded message names as defined during the start of a call.

```
Function getCallMessagesNames( ) As String
```

figure 13.90: getCallMessagesNames

Return value:

- **messages**
  List of names of prerecorded messages.
  The messages parameter is formatted as a comma separated set of message names. The ordering of the message is the play-order.

Related Event types:

- OIDET_CALLSTART

Error codes:

- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE

13.5.24 isCallLiveSpeech

Get function for the indicator whether there is a live speech section in the call defined during the start of the call.

```
Function isCallLiveSpeech( ) As Boolean
```

figure 13.91: isCallLiveSpeech

Return value:

- **liveSpeech**
  Whether or not the call has a live speech phase.
  True = live speech, **False** = no live speech.

Related Event types:

- OIDET_CALLSTART

Error codes:

- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE

13.5.25 getCallMessageRepeatCount

Get Function for the repeat count of the prerecorded messages as defined during the start of a call.

```
Function getCallMessageRepeatCount( ) As Long
```

figure 13.92: getCallMessageRepeatCount

Return value:

- **repeat**
  The repeat count of the prerecorded messages.
  Special values are defined in section 13.2.5.
  The valid range for the repeat count is 1 .. 32767.

Related Event types:

- OIDET_CALLSTART

Error codes:

- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE

13.5.26 getCallPriority

Get function for the call-priority as defined during the start of a call.

```
Function getCallPriority( ) As Long
```

figure 13.93: getCallPriority

Return value:

- **priority**
  The priority of the call. See section 13.2.4 for the value range definitions.

Related Event types:

- **OIDET_CALLSTART**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.27 getCallId

Get Function for the call identification. The call identification number can be used to track the call.

```
Function getCallId( ) As Long
```

figure 13.94: getCallId

Return value:

- **callId**
  Event identification number.

Related Call types:

All events in event-group **OIDEG_CALLEVENTGROUP.**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.28 getOriginalCallId

Get Function for the original call identification of an extended call. The original call identification number can be used to track the replays of extended calls.

```
Function getOriginalCallId () As Long
```

figure 13.95: getOriginalCallId

Return value:

- callId
  Event identification number.

Related Call types:

- All events in event-group **OIDEG_CALLEVENTGROUP.**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE**

13.5.29 getCallMacroName

Get function for the macro name as defined during the call.

Function **getCallMacroName**( ) As String

figure 13.96: getCallMacroName

Return value:

- **macroName**
  Name of the macro used during the call.

Related Event types:

- **OIDET_CALLSTARTV2**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE**

13.5.30 getCallStateCompleted

Get Function for the last completed call state the moment the call is stopped or aborted.

Function **getCallStateCompleted**( ) As TOICallState

figure 13.97: getCallStateCompleted

Return value:

- **state**
  Last completed call state

Related Event types:

- **OIDET_CALLENDV2**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE**

13.5.31 isCallAborted

Get Function for the indicator whether a call was aborted.

Function **isCallAborted()** As Boolean

figure 13.98: isCallAborted

Return value:

- **bAborted**
  Whether or not the call was aborted. **True** = aborted, **False** = stopped.

Related Event types:

- **OIDET_CALLENDV2**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE**

13.5.32 getCallEndReason

Get function for the reason why the call was stopped or aborted.

Function **getCallEndReason ()** As TOICallEndReason

figure 13.99: getCallEndReason

Return value:

- **endReason**
  Reason for the call to end.

Related Event types:

- **OIDET_CALLENDV2**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE**

13.5.33 getNrConfiguredCstExtensions

Get Function for the number of configured call-station extensions.

Function **getNrConfiguredCstExtensions**( ) As Long

figure 13.100: getNrConfiguredCstExtensions

Return value:

- **extCount**
  The number of configured extensions for this call-station.

Related Event types:

- **OIDET_CALLSTATIONEXTENSION**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_ DIAGEVENT_TYPE**

13.5.34 getNrDetectedCstExtensions

Get Function for the number of detected call-station extensions.

Function **getNrDetectedCstExtensions**( ) As Long

figure 13.101: getNrDetectedCstExtensions

Return value:

- **extCount**
  The number of detected extensions for this call-station.

Related Event types:

- **OIDET_CALLSTATIONEXTENSION**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_ DIAGEVENT_TYPE**

13.5.35 getErrorCode

Get function for the error code generated by the CobraNet module inside the CobraNet unit or OMNEO module in the OMNEO unit.

Function **getErrorCode**( ) As Long

figure 13.102: getErrorCode

Return value:

- **error**
  CobraNet/OMNEO module error code.

Related Event types:

- OIDET_COBRANETINTERFACE
- OIDET_COBRANETNETWORK
- OIDET_OMNEOINTERFACE
- OIDET_OMNEONETWORK

Error codes:

- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE

13.5.36 getExpectedConfigVersion

Get function for the expected configuration version.

Function **getExpectedConfigVersion**( ) As String

figure 13.103: getExpectedConfigVersion

Return value:

- **version**
  version in text representation.

Related Event types:

- OIDET_CONFIGURATIONVERSION

Error codes:

- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_ DIAGEVENT_TYPE

13.5.37 getLoadedConfigVersion

Get function for the loaded configuration version.

Function **getLoadedConfigVersion**( ) As String

figure 13.104: getLoadedConfigVersion

Return value:

- **version**
  version in text representation.

Related Event types:

- OIDET_CONFIGURATIONVERSION

Error codes:

- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_ DIAGEVENT_TYPE

13.5.38 isEepromMemoryError

Get function for the identification whether the memory error is related to EEPROM memory.

figure 13.105: isEepromMemoryError

Return value:

- **eepromError**
  EEPROM error indicator.

Related Event types:

- **OIDET_MEMORYERROR**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.39 isFlashMemoryError

Get function for the identification whether the memory error is related to Flash memory.

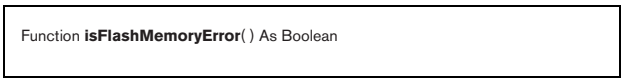

figure 13.106: isFlashMemoryError

Return value:

- **flashError**
  Flash error indicator.

Related Event types:

- **OIDET_MEMORYERROR**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE**

13.5.40 getMissingMessages

Get function for the missing prerecorded message names.

figure 13.107: getMissingMessages

Return value:

- **messages**
  List of missing names of prerecorded messages. The messages parameter is formatted as a comma separated set of message names.

Related Event types:

- **OIDET_PRERECORDEDMESSAGESNAMES**

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE

13.5.41 getChipType

Get function for the type of processor chip, which has been restarted.

```
Function getChipType( ) As TOIChipType
```

figure 13.108: getChipType

Return value:
- chipType
  Type of the processor chip. See section 13.2.17 for the definitions of the chip-types.

Related Event types:
- OIDET_UNITRESET

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE

13.5.42 getErrorDescription

Get function for the error description passed with a user-injected fault.

```
Function getErrorDescription( ) As String
```

figure 13.109: getErrorDescription

Return value:
- description
  Text representation of the user-injected fault.

Related Event types:
- OIDET_USERINJECTEDFAULT

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE

13.5.43 getDiagZoneNames

Get function for the zone names passed with a zone line fault.

```
Function getDiagZoneNames() As String
```

figure 13.110: getDiagZoneNames

Return value:
- zoneNames
  Zone names which are configured to input contact that are reported.

Related Event types:
- OIDET_ZONELINEFAULT

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE

13.5.44 getAmplifierNr

Get function for the related amplifier inside a booster

Function **getAmplifierNr**( ) As Long

figure 13.111: getAmplifierNr

Return value:

- **amplifier**
  Amplifier number (first amplifier gets number 0).

Related Event types:

- **OIDET_WLSBOARD**
- **OIDET_WLSBOARDMAINSPARE**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_ DIAGEVENT_TYPE**

13.5.45 getNrWls2Slaves

Get function for the number of WLS2 slave boards reporting the supervision fault. The number indicates the range, which can be used within the functions **getWls2SlaveAddress** and **getWls2SlaveName**.

Function **getNrWls2Slaves**( ) As Long

figure 13.112: getNrWls2Slaves

Return value:

- **slaveCount**
  Number of slave boards.

Related Event types:

- **OIDET_ENDOFLINESUPERVISION**
- **OIDET_LOUDSPEAKERSUPERVISION**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_ DIAGEVENT_TYPE**

13.5.46 getWls2SlaveAddress

Get function for the WLS2 slave board address.

Function **getWls2SlaveAddress**(idx As Long) As Long

figure 13.113: getWls2SlaveAddress

Parameters:

- **idx**
  The index in the set of addresses. Valid range for the index is 0 .. (**getNrWls2Slaves**() - 1).

Return value:

- **address**
  The address of the WLS2 slave board generating the supervision fault.

Related Event types:

- **OIDET_ENDOFLINESUPERVISION**
- **OIDET_LOUDSPEAKERSUPERVISION**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_ DIAGEVENT_TYPE**

13.5.47 getWls2SlaveName

Get Function for the WLS2 slave board name.

Function **getWls2SlaveName**(idx As Long) As String

figure 13.114: getWls2SlaveName

Parameters:

- **idx**
  The index in the set of addresses. Valid range for the index is 0 .. (**getNrWls2Slaves**() - 1).

Return value:

- **name**
  Configured name of the WLS2 slave board generating the supervision fault.

Related Event types:
- OIDET_ENDOFLINESUPERVISION
- OIDET_LOUDSPEAKERSUPERVISION

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE

13.5.48 getCurrentHWVersion

Get function for the current (detected) HW version of the unit (originator) where the diagnostic event is related.

Function **getCurrentHWVersion**( ) As String

figure 13.115: getCurrentHWVersion

Return value:
- **hwVersion**
  Hardware version in text representation.

Related Event types:
- OIDET_INCOMPATIBLEHWVERSION

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE

13.5.49 getMinimalHWVersion

Get function for the minimal (detected) HW version of the unit (originator) where the diagnostic event is related. If the minimal HW version in not full filled, then the current SW version cannot operate with that hardware.

Function **getMinimalHWVersion**( ) As String

figure 13.116: getMinimalHWVersion

Return value:
- **hwVersion**
  Hardware version in text representation.

Related Event types:
OIDET_INCOMPATIBLEHWVERSION

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE

13.5.50 getCurrentFwVersion

Get function for the current (detected) FW version of the unit (originator) where the diagnostic event is related.

Function **getCurrentFwVersion**() As String

figure 13.117: getCurrentFWVersion

Return value:
- **fwVersion**
  Firmware version in text representation.

Related Event types:
- OIDET_INCOMPATIBLEHWVERSION

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE

13.5.51 getRequiredFwVersion

Get function for the required (detected) FW version of the unit (originator) where the diagnostic event is related. If the minimal FW version in not fulfilled, then the current SW version cannot operate with that firmware.

```
Function getRequiredFwVersion() As String
```

figure 13.118: getRequiredFWVersion

Return value:
- **fwVersion**
  Firmware version in text representation.

Related Event types:
- OIDET_INCOMPATIBLEHWVERSION

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAG EVENT_TYPE

13.5.52 getUnreachedResourceNames

Get function for the unreached resources during a stacked call. The function returns a comma separated string containing the resource names.

```
Function getUnreachedResourceNames() As String
```

figure 13.119: getUnreachedResourceNames

Return value:
- **resources**
  List of names of zone groups, zones and/or control outputs. A comma separates each name in the routing list.

Related Event types:
- OIDET_ CALLTIMEOUTV2

Error codes:
- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_DIAGEVENT_TYPE

13.6 IEventOriginator Methods

13.6.1 getOriginatorType

Get function for the originator type.

Function **getOriginatorType**( ) As TOIEventOriginatorType

figure 13.120: getOriginatorType

Return value:

- **orgType**
  Type of the originator. See section 13.2.14 for the definitions of the originator types.

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE**

13.6.2 getUnitName

Get function of the originator's unit name

Function **getUnitName**( ) As String

figure 13.121: getUnitName

Return value:

- **unitname**
  Name of the unit triggering the event.

Related Originator types:

- **OIEOT_UNITEVENTORIGINATOR**
- **OIEOT_CONTROLINPUTORIGINATOR**
- **OIEOT_AUDIOOUTPUTEVENTORIGINATOR**
- **OIEOT_AUDIOINPUTEVENTORIGINATOR**
- **OIEOT_UNITMENUEVENTORIGINATOR**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE**

13.6.3 getUnitSerialNr

Get function for the unit's serial number.

Function **getUnitSerialNr**( ) As Long

figure 13.122: getUnitSerialNr

Return value:

- **serialNumber**
  Serial number of the unit.

Related Originator types:

- **OIEOT_UNITEVENTORIGINATOR**
- **OIEOT_CONTROLINPUTEVENTORIGINATOR**
- **OIEOT_AUDIOOUTPUTEVENTORIGINATOR**
- **OIEOT_AUDIOINPUTEVENTORIGINATOR**
- **OIEOT_UNITMENUEVENTORIGINATOR**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE**

13.6.4 getUnitType

Get function for the type of the unit.

```
Function getUnitType( ) As TOIUnitType
```

figure 13.123: getUnitType

Return value:

- **unitType**
  Type identification of the unit. See section 13.2.16 for the definitions of the unit-type.

Related Originator types:

- OIEOT_UNITEVENTORIGINATOR
- OIEOT_CONTROLINPUTEVENTORIGINATOR
- OIEOT_AUDIOOUTPUTEVENTORIGINATOR
- OIEOT_AUDIOINPUTEVENTORIGINATOR
- OIEOT_UNITMENUEVENTORIGINATOR

Error codes:

- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE

13.6.5 getTcpIpDeviceName

Get function for the TCP/IP device name of the TCP/IP originator.

```
Function getTcpIpDeviceName( ) As String
```

figure 13.124: getTcpIpDeviceName

Return value:

- **devName**
  The device name of the TCP/IP connection (if available).

Related Originator types:

- OIEOT_OPENINTERFACEEVENTORIGINATOR

Error codes:

- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE

13.6.6 getIpAddress

Get function for the IP-Address of the TCP/IP originator.

```
Function getIpAddress( ) As Long
```

figure 13.125: getIpAddress

Return value:

- **ipAddress**
  The IP-Address of the TCP/IP connection.

Related Originator types:

- OIEOT_OPENINTERFACEEVENTORIGINATOR

Error codes:

- OIERROR_OK
- OIERROR_NO_CONNECTION
- OIERROR_INVALID_PARAMETERS
- OIERROR_ALREADY_BUSY
- OIERROR_INTERNAL_ERROR
- OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE

13.6.7 getPortNumber

Get Function for the port number of the TCP/IP originator.

Function **getPortNumber**( ) As Long

figure 13.126: getPortNumber

Return value:

- **port**
  The port number of the TCP/IP connection.

Related Originator types:

- **OIEOT_OPENINTERFACEEVENTORIGINATOR**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE**

13.6.8 getUserName

Get function for the connection user name of the TCP/IP connection.

Function **getUserName**( ) As String

figure 13.127: getUserName

Return value:

- **name**
  The user name of the TCP/IP connect as used to login.

Related Originator types:

- **OIEOT_OPENINTERFACEEVENTORIGINATOR**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE**

13.6.9 getInputContactName

Get function for the contact name of the event triggering input.

Function **getInputContactName**( ) As String

figure 13.128: getInputContactName

Return value:

- **name**
  Configured input contact name.

Related Originator types:

- **OIEOT_CONTROLINPUTEVENTORIGINATOR**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE**

13.6.10 getAudioOutputName

Get function for the audio output triggering the event.

figure 13.129: getAudioOutputName

Return value:

- **name**
  Configured audio output name.

Related Originator types:

- **OIEOT_AUDIOOUTPUTEVENTORIGINATOR**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE**

13.6.11 getAudioInputName

Get function for the audio input name triggering the event.

figure 13.130: getAudioInputName

Return value:

- **name**
  Configured name of the audio input.

Related Originator types:

- **OIEOT_AUDIOINPUTEVENTORIGINATOR**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE**

13.6.12 getUserId

Get function for the user ID which has logged in on the numeric keypad.

figure 13.131: getUserId

Return value:

- **userId**
  The user ID of the user which has logged in on the numeric keypad.

Related Originator types:

- **OIEOT_USEREVENTORIGINATOR**

Error codes:

- **OIERROR_OK**
- **OIERROR_NO_CONNECTION**
- **OIERROR_INVALID_PARAMETERS**
- **OIERROR_ALREADY_BUSY**
- **OIERROR_INTERNAL_ERROR**
- **OIERROR_FUNCTION_NOT_SUPPORTED_BY_EVENT_ORIGINATOR_TYPE**

14 Example

14.1 Interface usage

In the example code below a simple Visual Basic application is pre-coded, containing a single form, where a subset of the functions mentioned above are present. This example could help you as a starting point for application development based on the Open Interface COM-server.

The interface method used in the example is based on the connection point method as described in section 12.3.

Most of the fields used in the form have self-explaining names.

14.1.1 Form Layout

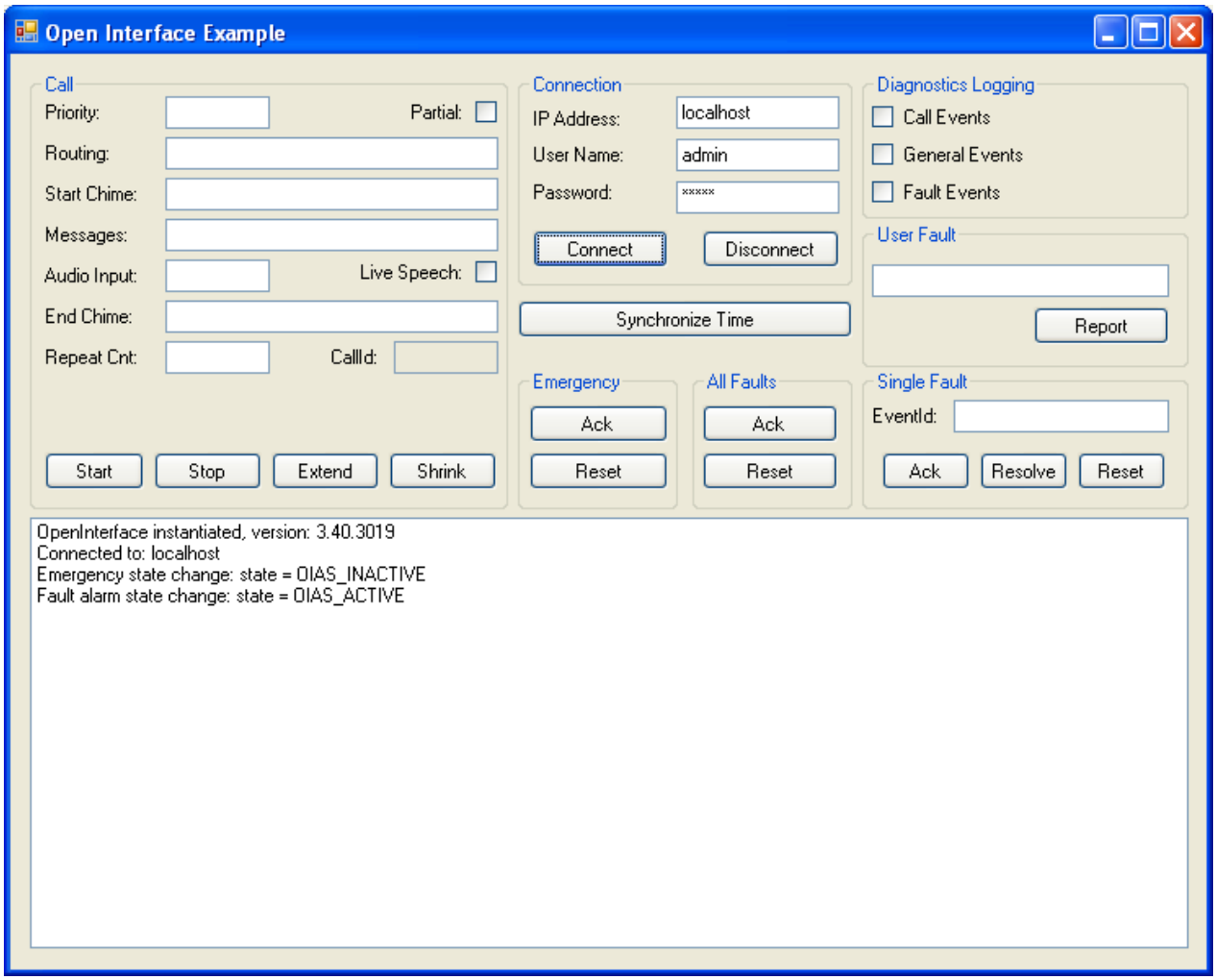

figure 14.1: Open interface example

14.1.2 Form code

```
' Copyright Robert Bosch GmbH, 2010.
' Bosch Security Systems B.V.
' BU Communications Systems, The Netherlands
'
'  Project            : Praesideo
'  Module             : OILVbExample
'  File name  $Workfile:   OILVbExample.vb  $
'             $Revision:   1.4  $
'           Last $Author:   pvg2bda  $
'  Creation Date      : 7 May 2010
'  First Author       : Patrick van Gelder
'
'  Description        : Example showing the use of the Praesideo
'                       OpenInterface using VB.NET.

Imports PRAESIDEOOPENINTERFACECOMSERVERLib
Imports Microsoft.VisualBasic

''' <summary>
''' This class implements an example with which it is possible to
''' execute some actions in a Praesideo system using the OpenInterface.
''' 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 Praesideo COM library.
''' There is only limited error checking.
''' </summary>
''' <remarks>This is only an example! Do not use it in real systems.</remarks>
Public Class OILVbExample

    ' Object reference definition to the Praesideo Interface
    Private WithEvents PraesideoOI As PraesideoOpenInterface_V0340

    ' Declare delegates for callbacks from the OpenInterface.
    ' When you click on e.g. a button in this example a call will be done to
    ' the COM library using the so called 'User Interface Thread'. This can
    ' result in a callback from the COM library. This callback is executed
    ' by a different thread owned by the COM library. This thread is not allowed
    ' to update the user interface and can interfere with the other thread if
    ' it tries to do so. Futhermore these callbacks should take very little
    ' time as the COM library can not do anything else when it is executing the
    ' callbacks. The solution for these issues is using 'Delegates'.
    ' See e.g. http://msdn.microsoft.com/en-us/library/74wy9422(VS.90).aspx
    ' for an explanation of delegates.
    Private Delegate Sub CallStateDelegate(ByVal callId As Integer, ByVal state As TOICallState)
    Private Delegate Sub ConnectionBrokenDelegate()
    Private Delegate Sub DiagEventDelegate(ByVal eventId As Integer, _
                                           ByVal action As TOIActionType, _
                                           ByVal pDiagEvent As Object)
    Private Delegate Sub AlarmStateDelegate(ByVal state As TOIAlarmState)
    Private Delegate Sub ResourceStateDelegate(ByVal resources As String, _
                                               ByVal state As TOIResourceState, _
                                               ByVal priority As Integer, _
                                               ByVal callId As Integer)

    ''' <summary>

    ''' Procedure called on the form loaded event.
    ''' It instantiates the OpenInterface COM interface.
    ''' </summary>
    ''' <param name="sender">Indicates who raised the event. (not used)</param>
    ''' <param name="e">Parameters of the event. (not used)</param>
    Private Sub OILVbExample_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Try
            ' Instantiate the OpenInterface Com-Server interface
            PraesideoOI = New PraesideoOpenInterface_V0340

            Dim versionOpenInterface As String
            versionOpenInterface = PraesideoOI.getVersion
            LogText("OpenInterface instantiated, version: " + versionOpenInterface)

        Catch ex As Exception
            LogText("COM interface reports Error: " + ex.Message)
        End Try
    End Sub
```

```
''' <summary>
''' Procedure that is called when the form is unloaded.
''' </summary>
''' <param name="sender">Indicates who raised the event. (not used)</param>
''' <param name="e">Parameters of the event. (not used)</param>
Private Sub OILVbExample_FormClosed(ByVal sender As System.Object, ByVal e As System.Windows.Forms.FormClosedEventArgs) Handles MyBase.FormClosed
    ' Clear the reference to the COM library.
    PraesideoOI = Nothing
End Sub
''' <summary>
''' Procedure that is called when the call start button is clicked.
''' It will start a call using the parameters defined by the user.
''' </summary>
''' <param name="sender">Indicates who raised the event. (not used)</param>
''' <param name="e">Parameters of the event. (not used)</param>
Private Sub btnCallStart_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCallStart.Click
    Try
        Dim outputHandling As TOICallOutputHandling
        Dim stackingTimeout As Long
        Dim preMonitorDest As String
        Dim liveSpeechAtt As Long
        Dim startChimeAtt As Long
        Dim endChimeAtt As Long
        Dim messageAtt As Long

        ' Determine if it is a partial or non partial call. Callstacking
        ' is not supported in this example.
        If chkPartial.Checked Then
            outputHandling = TOICallOutputHandling.OICOH_PARTIAL
        Else
            outputHandling = TOICallOutputHandling.OICOH_NON_PARTIAL
        End If
        stackingTimeout = 0
        preMonitorDest = String.Empty
        ' Attenuations are always 0 in this example.
        liveSpeechAtt = 0
        startChimeAtt = 0
        endChimeAtt = 0
        messageAtt = 0

        ' Subscribe to the resource updates of the routing.
        ' PraesideoOI_resourceState will be called when there is an
        ' update.
        PraesideoOI.setSubscriptionResources(True, tbRouting.Text)

        ' Create the call and store the callId.
        tbCallId.Text = PraesideoOI.createCallEx2(tbRouting.Text, _
                        Val(tbPriority.Text), _
                        outputHandling, _
                        TOICallStackingMode.OICSM_WAIT_FOR_EACH, _
                        stackingTimeout, _
                        tbStartChime.Text, _
                        tbEndChime.Text, _
                        chkLiveSpeech.Checked, _
                        tbAudioInput.Text, _
                        tbMessages.Text, _
                        Val(tbRepeatCnt.Text), _
                        TOICallTiming.OICTM_IMMEDIATE, _
                        preMonitorDest, _
                        liveSpeechAtt, _
                        startChimeAtt, _
                        endChimeAtt, _
                        messageAtt)
        ' Start the created call using the generated callId.
        PraesideoOI.startCreatedCall(Val(tbCallId.Text))

    Catch ex As Exception
        LogText("Start call reports Error: " + ex.Message)
    End Try
```

```
End Sub
    ''' <summary>
    ''' Procedure that is called when the call start button is clicked.
    ''' It will stop the running call.
    ''' </summary>
    ''' <param name="sender">Indicates who raised the event. (not used)</param>
    ''' <param name="e">Parameters of the event. (not used)</param>
    ''' <remarks>It is assumed that there is a running call.</remarks>
    Private Sub btnCallStop_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCallStop.Click
        Try
            ' This example has no protection whether the call was already stopped
            PraesideoOI.stopCall(Val(tbCallId.Text))
            tbCallId.Text = TCallId.OI_UNDEFINED_CALLID
        Catch ex As Exception
            LogText("Stop call reports Error: " + ex.Message)
        End Try
    End Sub
    ''' <summary>
    ''' Procedure that is called when the call extend button is clicked.
    ''' It will extend the running call with the routing defined by the user.
    ''' </summary>
    ''' <param name="sender">Indicates who raised the event. (not used)</param>
    ''' <param name="e">Parameters of the event. (not used)</param>
    ''' <remarks>It is assumed that there is a running call.</remarks>
    Private Sub btnCallExtend_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCallExtend.Click
        Try
            PraesideoOI.addToCall(Val(tbCallId.Text), tbRouting.Text)
        Catch ex As Exception
            LogText("Extend call reports Error: " + ex.Message)
        End Try
    End Sub
    ''' <summary>
    ''' Procedure that is called when the call shrink button is clicked.
    ''' It will remove the routing defined by the user from the running call.
    ''' </summary>
    ''' <param name="sender">Indicates who raised the event. (not used)</param>
    ''' <param name="e">Parameters of the event. (not used)</param>
    ''' <remarks>It is assumed that there is a running call.</remarks>
    Private Sub btnShrink_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnShrink.Click
        Try
            PraesideoOI.removeFromCall(Val(tbCallId.Text), tbRouting.Text)
        Catch ex As Exception
            LogText("Shrink call reports Error: " + ex.Message)
        End Try
    End Sub
    ''' <summary>
    ''' Procedure that is called when the connect button is clicked.
    ''' It will connect to the given Preasideo system.
    ''' </summary>
    ''' <param name="sender">Indicates who raised the event. (not used)</param>
    ''' <param name="e">Parameters of the event. (not used)</param>
    Private Sub btnConnect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnConnect.Click
        Try
            ' Portnumber is not used anymore.
            PraesideoOI.connect(tbIPAddress.Text, _
                                0, _
                                tbUserName.Text, _
                                tbPassword.Text)
            PraesideoOI.setSubscriptionEvacAlarm(True)
            PraesideoOI.setSubscriptionFaultAlarm(True)

            LogText("Connected to: " + tbIPAddress.Text)
        Catch ex As Exception
            LogText("Connection failure Error: " + ex.Message)
        End Try
    End Sub
    ''' <summary>
    ''' Procedure that is called when the disconnect button is clicked.
    ''' It will disconnect the current connection.
    ''' </summary>
    ''' <param name="sender">Indicates who raised the event. (not used)</param>
    ''' <param name="e">Parameters of the event. (not used)</param>
    ''' <remarks>It is assumed that there is a connection active.</remarks>
    Private Sub btnDisconnect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnDisconnect.Click
        Try
            LogText("On demand disconnection")
            PraesideoOI.disconnect()
        Catch ex As Exception
            LogText("Disconnect problem Error: " + ex.Message)
        End Try
    End Sub
```

```
''' <summary>
''' Procedure that is called when the Synchronize Time button is clicked.
''' It will set the time of the Praesideo system to the current time of the
''' PC running this example.
''' </summary>
''' <param name="sender">Indicates who raised the event. (not used)</param>
''' <param name="e">Parameters of the event. (not used)</param>
Private Sub btnSynchronizeTime_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnSynchronizeTime.Click
    Dim dtCurrent As Date
    dtCurrent = Now
    Try
        PraesideoOI.setDateAndTime(DateAndTime.Year(dtCurrent), _
                                   DateAndTime.Month(dtCurrent), _
                                   DateAndTime.Day(dtCurrent), _
                                   Hour(dtCurrent), _
                                   Minute(dtCurrent), _
                                   Second(dtCurrent))

        LogText("Praesideo Date/Time set to: " + dtCurrent)
    Catch ex As Exception
        LogText("Setting Current Date/Time Error: " + ex.Message)
    End Try
End Sub
''' <summary>
''' Procedure that is called when the Emergency Ack button is clicked.
''' It will acknowledge the evacuation state of the system.
''' </summary>
''' <param name="sender">Indicates who raised the event. (not used)</param>
''' <param name="e">Parameters of the event. (not used)</param>
Private Sub btnEmergencyAck_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnEmergencyAck.Click
    Try
        PraesideoOI.ackEvacAlarm()
    Catch ex As Exception
        LogText("Acknowledge emergency reports Error: " + ex.Message)
    End Try
End Sub
''' <summary>
''' Procedure that is called when the Emergency Reset button is clicked.
''' It will reset the evacuation state of the system.
''' </summary>
''' <param name="sender">Indicates who raised the event. (not used)</param>
''' <param name="e">Parameters of the event. (not used)</param>
''' <remarks>The evacuation state will only be reset when it is already
''' acknowledged by the user.</remarks>
Private Sub btnEmergencyReset_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnEmergencyReset.Click
    Try
        PraesideoOI.resetEvacAlarm()
    Catch ex As Exception
        LogText("Reset emergency reports Error: " + ex.Message)
    End Try
End Sub
''' <summary>
''' Procedure that is called when the All Faults Ack button is clicked.
''' It will acknowledge all faults in the system.
''' </summary>
''' <param name="sender">Indicates who raised the event. (not used)</param>
''' <param name="e">Parameters of the event. (not used)</param>
Private Sub btnAllAck_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnAllAck.Click
    Try
        PraesideoOI.ackAllFaults()
    Catch ex As Exception
        LogText("Acknowledge all faults Error: " + ex.Message)
    End Try
End Sub
''' <summary>
''' Procedure that is called when the All Faults Reset button is clicked.
''' It will reset all faults in the system.
''' </summary>
''' <param name="sender">Indicates who raised the event. (not used)</param>
''' <param name="e">Parameters of the event. (not used)</param>
''' <remarks>The faults state will only be reset when they are resolved and
''' acknowledged by the user.</remarks>
Private Sub btnAllReset_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnAllReset.Click
    Try
        PraesideoOI.resetAllFaults()
    Catch ex As Exception
        LogText("Reset all faults Error: " + ex.Message)
    End Try
End Sub
```

```
''' <summary>
  ''' Procedure that is called when the Single Fault Ack button is clicked.
  ''' It will acknowledge a fault of which the id is defined by the user
  ''' in the system.
  ''' </summary>
  ''' <param name="sender">Indicates who raised the event. (not used)</param>
  ''' <param name="e">Parameters of the event. (not used)</param>
  ''' <remarks>It is assumed the user has filled in a valid id.</remarks>
  Private Sub btnSingleAck_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnSingleAck.Click
    Try
      PraesideoOI.ackFault(Val(tbSingleEventId.Text))
    Catch ex As Exception
      LogText("Acknowledge event: " + tbSingleEventId.Text + _
          " Error: " + ex.Message)
    End Try
  End Sub
  ''' <summary>
  ''' Procedure that is called when the Single Fault Resolve button is clicked.
  ''' It will resolve a fault of which the id is defined by the user
  ''' in the system.
  ''' </summary>
  ''' <param name="sender">Indicates who raised the event. (not used)</param>
  ''' <param name="e">Parameters of the event. (not used)</param>
  ''' <remarks>It is assumed the user has filled in a valid id.</remarks>
  Private Sub btnSingleResolve_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnSingleResolve.Click
    Try
      PraesideoOI.resolveFault(Val(tbSingleEventId.Text))
    Catch ex As Exception
      LogText("Resolve event: " + tbSingleEventId.Text + _
          " Error: " + ex.Message)
    End Try
  End Sub
  ''' <summary>
  ''' Procedure that is called when the Single Fault Reset button is clicked.
  ''' It will reset a fault of which the id is defined by the user
  ''' in the system.
  ''' </summary>
  ''' <param name="sender">Indicates who raised the event. (not used)</param>
  ''' <param name="e">Parameters of the event. (not used)</param>
  ''' <remarks>It is assumed the user has filled in a valid id.
  ''' The fault state will only be reset if it is resolved and
  ''' acknowledged by the user.</remarks>
  Private Sub btnSingleReset_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnSingleReset.Click
    Try
      PraesideoOI.resetFault(Val(tbSingleEventId.Text))
    Catch ex As Exception
      LogText("Reset event: " + tbSingleEventId.Text + _
          " Error: " + ex.Message)
    End Try
  End Sub
  ''' <summary>
  ''' Procedure that is called when the Report Fault button is clicked.
  ''' It will generate a fault with the user defined text in the system.
  ''' </summary>
  ''' <param name="sender">Indicates who raised the event. (not used)</param>
  ''' <param name="e">Parameters of the event. (not used)</param>
  Private Sub btUserFaultReport_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btUserFaultReport.Click
    Try
      PraesideoOI.reportFault(tbUserFault.Text)
    Catch ex As Exception
      LogText("Report Fault: " + tbUserFault.Text + _
          " Error: " + ex.Message)
    End Try
  End Sub
  ''' <summary>
  ''' Procedure that is called when the Diagnostics Logging Call Event
  ''' checkbox is changed.
  ''' It will (un)subscribe the example from call event updates.
  ''' </summary>
  ''' <param name="sender">Indicates who raised the event. (not used)</param>
  ''' <param name="e">Parameters of the event. (not used)</param>
  Private Sub chkDiagCallEvents_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles chkDiagCallEvents1.CheckedChanged
    Try
      PraesideoOI.setSubscriptionEvents(chkDiagCallEvents1.Checked, TOIDiagEventGroup.OIDEG_CALLEVENTGROUP)
    Catch ex As Exception
      LogText("Call Event Subscription Error: " + ex.Message)
    End Try
  End Sub
```

```
''' <summary>
  ''' Procedure that is called when the Diagnostics Logging General Event
  ''' checkbox is changed.
  ''' It will (un)subscribe the example from general event updates.
  ''' </summary>
  ''' <param name="sender">Indicates who raised the event. (not used)</param>
  ''' <param name="e">Parameters of the event. (not used)</param>
  Private Sub chkDiagGeneral_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles chkDiagGeneral1.CheckedChanged
    Try
      PraesideoOI.setSubscriptionEvents(chkDiagGeneral1.Checked, TOIDiagEventGroup.OIDEG_GENERALEVENTGROUP)
    Catch ex As Exception
      LogText("General Event Subscription Error: " + ex.Message)
    End Try
  End Sub
  ''' <summary>
  ''' Procedure that is called when the Diagnostics Logging Fault Event
  ''' checkbox is changed.
  ''' It will (un)subscribe the example from fault event updates.
  ''' </summary>
  ''' <param name="sender">Indicates who raised the event. (not used)</param>
  ''' <param name="e">Parameters of the event. (not used)</param>
  Private Sub chkDiagFault_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles chkDiagFault1.CheckedChanged
    Try
      PraesideoOI.setSubscriptionEvents(chkDiagFault1.Checked, TOIDiagEventGroup.OIDEG_FAULTEVENTGROUP)
    Catch ex As Exception
      LogText("Fault Event Subscription Error: " + ex.Message)
    End Try
  End Sub
  ''' <summary>
  ''' Procedure that is called by the OpenInterface when the callstate changes.
  ''' </summary>
  ''' <param name="callId">The id of the call that changed state.</param>
  ''' <param name="state">The new state of the call.</param>
  Private Sub PraesideoOI_callState(ByVal callId As Integer, ByVal state As TOICallState) Handles PraesideoOI.callState
    ' Indicates whether or not an invoke is required.
    If InvokeRequired Then
      ' Invoke is required because this procedure is not called by the thread of
      ' the user interface. Use BeginInvoke because it is not desirable to
      ' wait for the result. This will call this procedure but now in the context
      ' of the user interface thread.
      BeginInvoke(New CallStateDelegate(AddressOf PraesideoOI_callState), callId, state)
    Else
      ' Called by the thread of the user interface so it is possible to execute.
      LogText("CallState report: callId = " + callId.ToString() + _
          " state = " + state.ToString())
      If state = TOICallState.OICS_ABORT Or _
          state = TOICallState.OICS_END Then
        tbCallId.Text = TCallId.OI_UNDEFINED_CALLID
      End If
    End If
  End Sub
  ''' <summary>
  ''' Procedure that is called by the OpenInterface when the connection with the
  ''' Praesideo system is broken.
  ''' </summary>
  Private Sub PraesideoOI_connectionBroken() Handles PraesideoOI.connectionBroken
    If InvokeRequired Then
      BeginInvoke(New ConnectionBrokenDelegate(AddressOf PraesideoOI_connectionBroken))
    Else
      LogText("Connect broken reported (remote)")
    End If
  End Sub
  ''' <summary>
  ''' Procedure that is called by the OpenInterface when a diagnostics event is
  ''' logged in the Praesideo system.
  ''' </summary>
  ''' <param name="eventId">The id of the event.</param>
  ''' <param name="action">The action done on the event.</param>
  ''' <param name="DiagEvent">The event that was logged.</param>
  ''' <remarks>Only called when there is a subscription active.</remarks>
  Private Sub PraesideoOI_diagEvent(ByVal eventId As Integer, _
                                    ByVal action As TOIActionType, _
                                    ByVal DiagEvent As Object) Handles PraesideoOI.diagEvent
    If InvokeRequired Then
      BeginInvoke(New DiagEventDelegate(AddressOf PraesideoOI_diagEvent), _
          eventId, action, DiagEvent)
    Else
      LogText("EventId = " + eventId.ToString() + _
          " action = " + action.ToString() + _
          " Event: " + DiagEventToString(DiagEvent))
    End If
  End Sub
```

```
''' <summary>
''' Procedure that is called by the OpenInterface when the Praesideo system
''' evacuation state changes.
''' </summary>
''' <param name="state">The new evacuation state.</param>
''' <remarks>Only called when there is a subscription active.</remarks>
Private Sub PraesideoOI_evacAlarmState(ByVal state As TOIAlarmState) Handles PraesideoOI.evacAlarmState
   If InvokeRequired Then
      BeginInvoke(New AlarmStateDelegate(AddressOf PraesideoOI_evacAlarmState), state)
   Else
      LogText("Emergency state change: state = " + state.ToString())
   End If
End Sub
''' <summary>
''' Procedure that is called by the OpenInterface when the Praesideo system
''' fault state changes.
''' </summary>
''' <param name="state">The new fault state.</param>
''' <remarks>Only called when there is a subscription active.</remarks>
Private Sub PraesideoOI_faultAlarmState(ByVal state As TOIAlarmState) Handles PraesideoOI.faultAlarmState
   If InvokeRequired Then
      BeginInvoke(New AlarmStateDelegate(AddressOf PraesideoOI_faultAlarmState), state)
   Else
      LogText("Fault alarm state change: state = " + state.ToString())
   End If
End Sub
''' <summary>
''' Procedure that is called by the OpenInterface when a resource state changes.
''' </summary>
''' <param name="resources">A list of resources (zones etc).</param>
''' <param name="state">State of the resources.</param>
''' <param name="priority">Priority of the call using the resource.</param>
''' <param name="callId">The callId associated with the resource.</param>
''' <remarks>Only called when there is a subscription active.</remarks>
Private Sub PraesideoOI_resourceState(ByVal resources As String, _
                                      ByVal state As TOIResourceState, _
                                      ByVal priority As Integer, _
                                      ByVal callId As Integer) Handles PraesideoOI.resourceState
   If InvokeRequired Then
      BeginInvoke(New ResourceStateDelegate(AddressOf PraesideoOI_resourceState), _
            resources, state, priority, callId)
   Else
      LogText("Resource state change: resources = '" + resources + _
            "' state = " + state.ToString() + _
            " priority = " + priority.ToString() + _
            " callId = " + callId.ToString())
   End If
End Sub

''' <summary>
''' This function is an example how the diagnostic events can be
''' converted into a string.
''' </summary>
''' <param name="ev">The event to convert.</param>
''' <returns>Event converted to a string representation.</returns>
''' <remarks>In this implementation not all possible
''' diagnostic events are converted into a string. It only shows an
''' example how to deal with the various IDiagEvent_V0340 functions,
''' depending on the type of the event.</remarks>
Public Function DiagEventToString(ByVal ev As IDiagEvent_V0340) As String
   Dim evOrg As IEventOriginator_V0340
   evOrg = ev.getAddEventOriginatorObject()

   Dim result As String
   ' Format the event time and event originator for all events
   result = ev.getEventAddTime() + " # " + OriginatorToString(evOrg) + _
         " | " + ev.getEventId().ToString() + " | "

   ' divide the conversion depending on the group of events.
   Select Case ev.getEventGroup()
      Case TOIDiagEventGroup.OIDEG_CALLEVENTGROUP
         result = result + CallGroupEventDetails(ev)
      Case TOIDiagEventGroup.OIDEG_FAULTEVENTGROUP
         result = result + FaultGroupEventDetails(ev)
      Case TOIDiagEventGroup.OIDEG_GENERALEVENTGROUP
         result = result + GeneralGroupEventDetails(ev)
      Case Else
         DiagEventToString = "Invalid group: " + ev.getEventGroup() + _
            " EventType: " + ev.getEventType()
         Exit Function
   End Select
   DiagEventToString = result
End Function
```

```
    ''' <summary>
    ''' This function converts the originator into a string representation.
    ''' </summary>
    ''' <param name="evOrg">The event originator to convert.</param>
    ''' <returns>Originator converted to a string representation.</returns>
    ''' <remarks>Not all possible originators are supported.</remarks>
    Private Function OriginatorToString(ByVal evOrg As IEventOriginator_V0340) As String
        Select Case evOrg.getOriginatorType
            Case TOIEventOriginatorType.OIEOT_CONTROLINPUTEVENTORIGINATOR, _
                 TOIEventOriginatorType.OIEOT_AUDIOINPUTEVENTORIGINATOR, _
                 TOIEventOriginatorType.OIEOT_UNITMENUEVENTORIGINATOR, _
                 TOIEventOriginatorType.OIEOT_UNITEVENTORIGINATOR, _
                 TOIEventOriginatorType.OIEOT_AUDIOOUTPUTEVENTORIGINATOR
                OriginatorToString = evOrg.getUnitName() & " (SN " & _
                    Hex(evOrg.getUnitSerialNr) & ")"

            Case TOIEventOriginatorType.OIEOT_OPENINTERFACEEVENTORIGINATOR
                Dim i As Integer, ip As Long, ipstr As String
                ip = evOrg.getIpAddress()
                ipstr = Trim$(Str$(ip And 255))
                For i = 1 To 3
                    ip = ip / 256
                    ipstr = Trim$(Str$(ip And 255)) + "." + ipstr
                Next
                OriginatorToString = evOrg.getUserName() + " (IP=" + ipstr + ")"

            Case Else
                OriginatorToString = " Unknown general orginator type: " + _
                    evOrg.getOriginatorType().ToString()
        End Select
    End Function

    ''' <summary>
    ''' This function converts the call group event details into a string
    ''' </summary>
    ''' <param name="ev">Event to get the details from.</param>
    ''' <returns>Call group event details converted to a string representation.</returns>
    ''' <remarks></remarks>
    Private Function CallGroupEventDetails(ByVal ev As IDiagEvent_V0340) As String
        ' The call group events are grouped into individual event types.
        ' The method to be used is equal as shown by the fault group
        CallGroupEventDetails = "CallGroup EventType: " + ev.getEventType().ToString()
    End Function

    ''' <summary>
    ''' This function converts the Fault group event details into a string.
    ''' </summary>
    ''' <param name="ev">Event to get the details from.</param>
    ''' <returns>Fault group event details converted to a string representation.</returns>
    ''' <remarks>Not all possible fault groups are supported.</remarks>
    Private Function FaultGroupEventDetails(ByVal ev As IDiagEvent_V0340) As String
        ' The select case statement should be extended with all possible
        ' Fault diagnostic types
        Select Case ev.getEventType()
            Case TOIDiagEventType.OIDET_CALLSTATIONEXTENSION
                FaultGroupEventDetails = "Keypad mismatch: " + ev.getNrDetectedCstExtensions().ToString() + _
                    " connected - " + ev.getNrConfiguredCstExtensions().ToString() + " configured"
            Case TOIDiagEventType.OIDET_USERINJECTEDFAULT
                FaultGroupEventDetails = ev.getErrorDescription()
            Case Else
                FaultGroupEventDetails = "FaultGroup EventType: " + ev.getEventType().ToString()
        End Select
    End Function

    ''' <summary>
    ''' This function converts the General group event details into a string.
    ''' </summary>
    ''' <param name="ev">Event to get the details from.</param>
    ''' <returns>General group event details converted to a string representation.</returns>
    Private Function GeneralGroupEventDetails(ByVal ev As IDiagEvent_V0340) As String
        ' The general group events are grouped into individual event types.
        ' The method to be used is equal as shown in FaultGroupEventDetails.
        GeneralGroupEventDetails = "GeneralGroup EventType: " + ev.getEventType().ToString()
    End Function

    ''' <summary>
    ''' Logs a text in the logging box.
    ''' </summary>
    ''' <param name="LoggingText">Text to log.</param>
    Sub LogText(ByVal LoggingText As String)
        lbLogging.Items.Add(LoggingText)
        lbLogging.TopIndex = lbLogging.Items.Count - 1
    End Sub

End Class
```

14.2 Connecting using C++

This section gives an example of how to connect to the OpenInterface using C++.
Create a file CpraesideoOpenInterfaceEvents.h:

```
#include <atlbase.h>
extern CComModule _Module;
#include <atlcom.h>
#include <stdio.h>
#include <time.h>

#include "PraesideoOpenInterfaceCOMServer.tlh"

CComModule _Module;

// Definition of CPraesideoOpenInterfaceEvents callbackfunction.
class CPraesideoOpenInterfaceEvents :
   public CComObjectRoot,
   public IDispatchImpl<IPraesideoOpenInterfaceEvents_V0300,
                &(__uuidof(IPraesideoOpenInterfaceEvents_V0300)),
                &(__uuidof(PraesideoOpenInterface_V0310))>
{
public:

   CPraesideoOpenInterfaceEvents() :
      CComObjectRoot(),
      IDispatchImpl<IPraesideoOpenInterfaceEvents_V0300,
                &(__uuidof(IPraesideoOpenInterfaceEvents_V0300)),
                &(__uuidof(PraesideoOpenInterface_V0310))>()
   {}

   BEGIN_COM_MAP(CPraesideoOpenInterfaceEvents)
      COM_INTERFACE_ENTRY(IDispatch)
      COM_INTERFACE_ENTRY(IPraesideoOpenInterfaceEvents)
      COM_INTERFACE_ENTRY(IPraesideoOpenInterfaceEvents_V0210)
      COM_INTERFACE_ENTRY(IPraesideoOpenInterfaceEvents_V0220)
      COM_INTERFACE_ENTRY(IPraesideoOpenInterfaceEvents_V0230)
      COM_INTERFACE_ENTRY(IPraesideoOpenInterfaceEvents_V0300)
   END_COM_MAP()

    // Definition of the callback methods
   virtual HRESULT __stdcall raw_resourceState (
      BSTR resources,
      TOIResourceState state,
      long priority,
      long callId )
   {
      return S_OK;
   }

   virtual HRESULT __stdcall raw_faultAlarmState (
      TOIAlarmState state )
   {
      return S_OK;
   }

   virtual HRESULT __stdcall raw_evacAlarmState (
      TOIAlarmState state )
   {
      return S_OK;
   }

   virtual HRESULT __stdcall raw_callState (
      long callId,
      TOICallState state )
   {
      return S_OK;
   }

   virtual HRESULT __stdcall raw_connectionBroken ( )
   {
      return S_OK;
   }

   virtual HRESULT __stdcall raw_bgmRouting (
      BSTR channel,
      VARIANT_BOOL addition,
      BSTR routing )
   {
      return S_OK;
   }
```

```
    virtual HRESULT __stdcall raw_bgmRouting (
        BSTR channel,
        VARIANT_BOOL addition,
        BSTR routing )
    {
        return S_OK;
    }

    virtual HRESULT __stdcall raw_diagEvent (
        long eventId,
        TOIActionType action,
        IDispatch * pDiagEvent )
    {
        HRESULT hRes(S_OK);
        IDiagEvent_V0300* pIDiagEvent = NULL;

        // We'll retrieve an IDiagEvent_V0300 interface pointer here.
        hRes = pDiagEvent->QueryInterface(IID_IDiagEvent_V0300,
                                  reinterpret_cast<void**>(&pIDiagEvent));

        long                    localEventId;
        TOIDiagEventGroup       eventGroup;
        TOIDiagEventType        eventType;
        DATE                    addTime;
        SYSTEMTIME              stAddTime;
        char                    szAddTime[25];
        long                    serialNr(0);
        strcpy(szAddTime, "????-??-?? ??:??:??");

        if (hRes == S_OK)
        {
            localEventId = pIDiagEvent->getEventId();
            eventGroup = pIDiagEvent->getEventGroup();
            eventType = pIDiagEvent->getEventType();
            addTime = pIDiagEvent->getEventAddTime();
            VariantTimeToSystemTime(addTime, &stAddTime);

            sprintf(szAddTime, "%04d-%02d-%02d %02d:%02d:%02d",
                    (int)stAddTime.wYear,
                    (int)stAddTime.wMonth,
                    (int)stAddTime.wDay,
                    (int)stAddTime.wHour,
                    (int)stAddTime.wMinute,
                    (int)stAddTime.wSecond);
        }

        printf(" diagEvent, action=%d, id=%ld, grp=%d, eventType=%d, addTime=%s, tid=%d counter=%d\n",
                (int)action,
                localEventId,
                (int)eventGroup,
                (int)eventType,
                szAddTime,
                ::GetCurrentThreadId());

        pIDiagEvent->Release();
        return hRes;
    }

    virtual HRESULT __stdcall raw_dummy_V0230 ( )
    {
        return S_OK;
    }

    virtual HRESULT __stdcall raw_dummy_V0300 ( )
    {
        return S_OK;
    }
};
```

Main:

```
#import "PraesideoOpenInterfaceCOMServer.tlb" no_namespace named_guids

#include "stdafx.h"
#include <atlbase.h>
extern CComModule _Module;


#include <atlcom.h>
#include <stdio.h>
#include <time.h>


#include "PraesideoOpenInterfaceCOMServer.tlh"

#include "CpraesideoOpenInterfaceEvents.h"


int main(int argc, char* argv[])
{
   HRESULT hRes = ::CoInitializeEx(NULL, COINIT_MULTITHREADED);

   IPraesideoOpenInterface_V0300* pIPraesideoOpenInterface = NULL;

   CComObject<CPraesideoOpenInterfaceEvents>* pEvents = NULL;

   if (S_OK == hRes)
   {
      static_cast<void>(_Module.Init(NULL, 0));

      hRes = CoCreateInstance(__uuidof(PraesideoOpenInterface_V0300),
                        NULL,
                        CLSCTX_ALL,
                        __uuidof(IPraesideoOpenInterface_V0300),
                        (void **)&pIPraesideoOpenInterface);

   }
   if (S_OK == hRes)
   {
      hRes = CComObject<CPraesideoOpenInterfaceEvents>::CreateInstance(&pEvents);
   }
   if (S_OK == hRes)
   {
      hRes = pIPraesideoOpenInterface->registerClientInterface(pEvents);
   }
   if (S_OK == hRes)
   {
      try
      {

         pIPraesideoOpenInterface->connect(L"10.128.102.199", 12345, L"admin", L"admin");
         pIPraesideoOpenInterface->setSubscriptionEvents(VARIANT_TRUE, OIDEG_CALLEVENTGROUP);
         pIPraesideoOpenInterface->setSubscriptionEvents(VARIANT_TRUE, OIDEG_GENERALEVENTGROUP);
         pIPraesideoOpenInterface->setSubscriptionEvents(VARIANT_TRUE, OIDEG_FAULTEVENTGROUP);

         // ... continue your program here, for test purposes we wait ...
         Sleep(100000);
      }
      catch( ... )
      {
         printf("Failed to connect\n");
      }
   }
   ::CoUninitialize();
 return 0;
}

};
```

Note: Should there be an error with respect to the **CoCreateInstanceEx**, add a define for **_WIN32_DCOM** somewhere (e.g. in **stdafx.h**)

15 Backward version support

Besides the current interfaces the Praesideo Open Interface library also contains other interfaces for backward compatibility for software written for the previous version. In this way the existing VB software can be used after upgrading the NCO software and the Praesideo Open Interface COM-server.

The sections below describe the interfaces for the previous versions and the differences against the current interfaces.

15.1 Praesideo V2.0 Interface(s)

The main interface of the V2.0 Praesideo Open Interface is named **PraesideoOpenInterface** and the events interface is named **IPraesideoOpenInterfaceEvents**.

The **PraesideoOpenInterface** interface implements the following functions:

table 15.1: Implemented functions

Function	Section	Notes
connect	13.3.1	The 'port' parameter must be set to value 12345 for proper operation.
disconnect	13.3.2	
getVersion	13.3.3	
startCall	13.3.8	Deprecated in V3.0
stopCall	13.3.9	
abortCall	13.3.10	
addToCall	13.3.11	
removeFromCall	13.3.12	
ackAllFaults	13.3.15	
resetAllFaults	13.3.16	
ackEvacAlarm	13.3.17	
resetEvacAlarm	13.3.19	
registerClientInterface	13.3.21	
deregisterClientInterface	13.3.22	
setSubscriptionResources	13.3.23	
setSubscriptionResourceFault State	13.3.24	
setSubscriptionEvacAlarm	13.3.26	
setDateAndTime	13.3.20	

The **IPraesideoOpenInterfaceEvents** interface defines the following event functions:

table 15.2: Defined function

Function	Section	Notes
resourceState	13.4.1	The state of a zone group is marked free (OIRS_FREE) if the group is only partial occupied.
faultAlarmState	13.4.3	
evacAlarmState	13.4.4	
callState	13.4.5	The call-state OICS_IDLE is not generated, but converted to state OICS_STARTED.
connectionBroken	13.4.7	

15.2 Praesideo V2.1 Interface(s)

The main interface of the V2.1 Praesideo Open Interface is named **PraesideoOpenInterface_V0210** and the events interface is named **IPraesideoOpenInterfaceEvents_V0210**.

The **PraesideoOpenInterface_V0210** interface implements all the functions of the PraesideoOpenInterface V2.0 interface plus the following functions:

table 15.3: Additional functions

Function	Section	Notes
setBgmRouting	13.3.27	
addBgmRouting	13.3.28	
removeBgmRouting	13.3.29	
setBgmVolume	13.3.31	
incrementBgmVolume	13.3.32	
decrementBgmVolume	13.3.34	
setSubscriptionBgmRouting	13.3.36	

The **IPraesideoOpenInterfaceEvents_V0210** interface defines all the events of the **IPraesideoOpenInterfaceEvents** interface plus the following event functions:

table 15.4: Additional event functions

Function	Section	Notes
bgmRouting	13.4.6	

15.3 Praesideo V2.2 Interface(s)

The main interface of the V2.2 Praesideo Open Interface is named **PraesideoOpenInterface_V220** while the other interfaces (**IDiagEvent_V220**, **IPraesideoOpenInterfaceEvents_V220** and **IEventOriginator_V220**) have the same name as described in the document.

The **PraesideoOpenInterface_V220** interface implements all the functions of the **PraesideoOpenInterface V2.1** interface plus the following functions:

table 15.5: Additional functions (1)

Function	Section	Notes
reportFault	13.3.37	
resolveFault	13.3.38	
ackFault	13.3.39	
resetFault	13.3.40	
setSubscriptionEvents	13.3.43	

The **IPraesideoOpenInterfaceEvents_V0220** interface defines all the events of the **IPraesideoOpenInterfaceEvents V2.1** interface plus the following event functions:

table 15.6: Additional event functions (2)

Function	Section	Notes
diagEvent	13.4.8	Event returns the IDiagEvent interface.

Due to the introduction of the **diagEvent** two new interfaces are added for the Praesideo V2.2 Open Interface; **IDiagEvent** and **IEventOriginator**.

The IDiagEvent_V220 interface implements the following functions:

table 15.7: Additional functions (3)

Function	Section	Notes
getEventType	13.5.1	Includes also the events as described below.
getEventGroup	13.5.2	
getEventId	13.5.3	
getEventState	13.5.4	
getEventAddTime	13.5.5	
getEventAckTime	13.5.6	
getEventResolveTime	13.5.7	
getEventResetTime	13.5.8	
getAddEventOriginator		Deprecated in V2.3
getAcknowledgeEventOriginator		Deprecated in V2.3
getResolveEventOriginator		Deprecated in V2.3
getResetEventOriginator		Deprecated in V2.3
getSviName	13.5.13	
getSpareBoosterName	13.5.15	
getSpareBoosterSerialNr	13.5.16	
getRemovedResourceNames	13.5.17	
getAddedResourceNames	13.5.18	
getCallAudioInputName	13.5.19	
getCallOutputsNames	13.5.20	
getCallStartChimesNames	13.5.21	
getCallEndChimesNames	13.5.22	
getCallMessagesNames	13.5.23	
isCallLiveSpeech	13.5.24	
getCallMessageRepeatCount	13.5.25	
getCallPriority	13.5.26	
getNrConfiguredCstExtensions	13.5.33	
getNrDetectedCstExtensions	13.5.34	
getErrorCode	13.5.35	
getExpectedConfigVersion	13.5.36	
getLoadedConfigVersion	13.5.37	
isEepromMemoryError	13.5.38	
isFlashMemoryError	13.5.39	
getMissingMessages	13.5.40	
getChipType	13.5.41	
getErrorDescription	13.5.42	
getDiagZoneNames	13.5.43	
getAmplifierNr	13.5.44	
getSubjectEventAddTime		Deprecated in V2.3
getSubjectEventType		Deprecated in V2.3
getSubjectEventOriginator		Deprecated in V2.3
getNrWIs2Slaves	13.5.45	
getWIs2SlaveAddress	13.5.46	
getWIs2SlaveName	13.5.47	
getCurrentHWVersion	13.5.48	

table 15.7: Additional functions (3)

getMinimalHWVersion	**13.5.49**

The following Diagnostic Event Types are also sent in the Praesideo V2.20 interface:

table 15.8: Additional functions (4)

TOIDiagEventType	Notes
OIDET_ACKNOWLEDGEMENT	Indicates that the diagnostic event represents an acknowledgement of a specific fault event.
OIDET_RESOLVE	Indicates that the diagnostic event represents a resolve of a specific fault event.
OIDET_RESET	Indicates that the diagnostic event represents a reset of a specific fault event.

The **IEventOriginator_V220** interface implements all the functions as described in section 13.6.

15.4 Praesideo V2.3 Interface(s)

The main interface of the V2.3 Praesideo Open Interface is named **PraesideoOpenInterface_V230** while the other interfaces (**IDiagEvent_V230**, **IPraesideoOpenInterfaceEvents_V230** and **IEventOriginator_V230**) have the same name as described in the document.
The **PraesideoOpenInterface_V230** interface implements all the functions of the V2.2 Praesideo Open Interface.

The **IDiagEvent_V230** interface implements the following functions:

table 15.9: Implemented functions

Function	Section
getAddEventOriginatorObject	13.5.9
getAcknowledgeEventOriginatorObject	13.5.10
getResolveEventOriginatorObject	13.5.11
getResetEventOriginatorObject	13.5.12

The following functions will not be supported anymore in V2.3 (for compatibility reasons they will not be removed from the library):

table 15.10: Unsupported functions

Function
getAddEventOriginator
getAcknowledgeEventOriginator
getResolveEventOriginator
getResetEventOriginator
getSubjectEventAddTime
getSubjectEventType
getSubjectEventOriginator

The following diagnostic event types are also sent in the Praesideo V2.30 interface:

table 15.11: Diagnostic event types

TOIDiagEventType
OIDET_REMOTEPOWERSUPPLY
OIDET_REMOTEBACKUPSUPPLY
OIDET_REMOTECONNECTION

15.5 Praesideo V3.0 Interface(s)

The main interface of the V3.0 Praesideo Open Interface is named **PraesideoOpenInterface_V300** while the other interfaces (**IDiagEvent_V300**, **IPraesideoOpenInterfaceEvents_V300** and **IEventOriginator_V300**) have the same name as described in the document.

The **PraesideoOpenInterface_V300** interface implements all the functions of the **PraesideoOpenInterface V2.3** interface plus the following functions:

table 15.12: Implemented functions

Function	Section	Notes
createCallEx	13.3.6	Deprecated in V3.4
startCreatedCall	13.3.7	

The following function will not be supported anymore in V3.0 (for compatibility reasons it will not be removed from the library):

table 15.13: Unsupported functions

Function
startCall

The following Diagnostic Event Types are also sent in the Praesideo V3.00 interface:

table 15.14: Diagnostic event types

TOIDiagEventType
OIDET_BOOSTERSPARESWITCH2
OIDET_BOOSTERSPARESWITCHRETURN2
OIDET_POWERBACKUPSUPPLY2
OIDET_POWERMAINSSUPPLY2
OIDET_GROUPAFAULT
OIDET_GROUPBFAULT
OIDET_CLASSASWITCHOVER
OIDET_HUNDREDVLINEFAULT
OIDET_WLS2CCBSYNC
OIDET_AMPMISSING

15.6 Praesideo V3.1 Interface(s)

The main interface of the V3.1 Praesideo Open Interface is named **PraesideoOpenInterface_V310** while the other interfaces (**IDiagEvent_V310**, **IPraesideoOpenInterfaceEvents_V310** and **IEventOriginator_V310**) have the same name as described in the document.

The PraesideoOpenInterface_V310 interface implements all the functions of the **PraesideoOpenInterface V3.0** interface plus the following functions:

table 15.15: Implemented functions

Function	Section	Notes
createCallEx	13.3.6	With extended parameters. Deprecated in V3.4

The **PraesideoOpenInterface_V310** interface implements all the definitions of the **PraesideoOpenInterface V3.0** interface plus the following definitions:

table 15.16: Implemented definitions

TOIUnitType
OIUT_CRF
TOICallState
OICS_REPLAY
TOICallEndReason
OICSR_ORIGINATOR
OICSR_RESOURCE_LOST
OICSR_SYSTEM
OICSR_STOPCOMMAND
OICSR_UNKNOWN
TOICallOutputHandling
OICOH_PARTIAL
OICOH_NON_PARTIAL
OICOH_STACKED
TOICallStackingMode
OICSM_WAIT_FOR_ALL
OICSM_WAIT_FOR_EACH
TOICallTiming
OICTM_IMMEDIATE
OICSM_TIME_SHIFTED
OICSM_MONITORED
TOICallStackingTimeout
OICST_INFINITE

The **PraesideoOpenInterface_V310** interface implements all the functions of the **PraesideoOpenInterface V3.0** interface plus the following functions:

table 15.17: Implemented functions

Function	Section	Notes
cancelAll	13.3.13	
cancelLast	13.3.14	
toggleBgmRouting	13.3.30	
incrementBgmChannelVolume	13.3.33	
decrementBgmChannelVolume	13.3.35	
setSubscriptionBgmVolume	13.3.44	
getZoneNames	13.3.47	
getZoneGroupNames	13.3.48	
getMessageNames	13.3.49	
getChimeNames	13.3.50	
getAudioInputNames	13.3.51	
getBgmChannelNames	13.3.52	
getConfigId	13.3.54	

The **IPraesideoOpenInterfaceEvents_V0310** interface defines all the events of the **IPraesideoOpenInterfaceEvents_V0300** interface plus the following event functions:

table 15.18: Implemented events

Function	Section	Notes
bgmVolume	13.4.9	

The **IDiagEvent_V310** interface implements all the diag event of the **IDiagEvent_V300** interface plus the following functions:

table 15.19: Implemented functions

Function	Section	Notes
getCurrentFwVersion	13.5.50	
getRequiredFwVersion	13.5.51	
getCallMacroName	13.5.29	
getCallId	13.5.27	
getOriginalCallId	13.5.28	
getCallStateCompleted	13.5.30	
isCallAborted	13.5.31	
getCallEndReason	13.5.32	

The following Diagnostic Event Types are also sent in the Praesideo V3.10 interface:

table 15.20: Diagnostic event types

TOIDiagEventType
OIDET_CALLCHANGERESOURCEV2
OIDET_CALLENDV2
OIDET_CALLSTARTV2
OIDET_CALLTIMEOUTV2
OIDET_INVALIDFWVERSION
OIDET_INTERNALFAULT

The following Diagnostic Event Types will not be supported anymore in V3.0 (for compatibility reasons they will not be removed from the library):

table 15.21: Unsupported event types

TOIDiagEventType
OIDET_CALLCHANGERESOURCE
OIDET_CALLEND
OIDET_CALLSTART

15.7 Praesideo V3.3 Interface(s)

The main interface of the V3.3 Praesideo Open Interface is named **PraesideoOpenInterface_V330** while the other interfaces (**IDiagEvent_V330**, **IPraesideoOpenInterfaceEvents_V330** and **IEventOriginator_V330**) have the same name as described in the document.

The **PraesideoOpenInterface_V330** interface implements all the functions of the **PraesideoOpenInterface V3.10** interface plus the following functions:

table 15.22: Implemented events

Function	Section	Notes
resetEvacAlarmEx	13.3.18	With extended parameters

The **IPraesideoOpenInterfaceEvents_V0330** Interface defines all the events of the **IPraesideoOpenInterfaceEvents_V0310** interface plus the following event functions:

table 15.23: Implemented events

Function	Section	Notes
setSubscriptionResourceFault State	13.3.24	
resourceFaultState	13.4.2	

The following Diagnostic Event Types are also sent in the Praesideo V3.30 interface:

table 15.24: Diagnostic event types

TOIDiagEventType
OIDET_OPENINTERFACECONNECTFAILED
OIDET_USERLOGIN
OIDET_USERLOGOUT
OIDET_USERLOGINFAILED

15.8 Praesideo V3.4 Interface(s)

The main interface of the V3.4 Praesideo Open Interface is named **PraesideoOpenInterface_V340** while the other interfaces (**IDiagEvent_V340**, **IPraesideoOpenInterfaceEvents_V340** and IEventOriginator_V340) have the same name as described in the document.

The **PraesideoOpenInterface_V340** interface implements all the functions of the **PraesideoOpenInterface V3.30** interface plus the following functions:

table 15.25: Implemented events

Function	Section	Notes
createCallEx2	13.3.5	Create call with extended parameters for audio source attenuation control.

The following function will not be supported anymore in V3.4 (for compatibility reasons it will not be removed from the library):

table 15.26: Implemented events

Function
createCallEx

The following Diagnostic Event Types are also sent in the Praesideo V3.40 interface:

table 15.27: Diagnostic event types

TOIDiagEventType
OIDET_LINEINPUTSUPERVISION
OIDET_WLSBOARDMAINSPARE

15.9 Praesideo V3.6 Interface(s)

The main interface of the V3.6 Praesideo Open Interface is named **PraesideoOpenInterface_V360** while the other interfaces (**IDiagEvent_V360**, **IPraesideoOpenInterfaceEvents_V360** and **IEventOriginator_V360**) have the same name as described in the document.

The **PraesideoOpenInterface_V360** interface implements all the functions of the **PraesideoOpenInterface V3.40** interface plus the following functions:

table 15.28: Implemented events

Function	Section	Notes
activateVirtualControlInput	13.3.41	Activate a virtual control input
deactivateVirtualControlInput	13.3.42	Deactivate a virtual control input
setSubscriptionUnitCount	13.3.45	Set a subscription for changes in the number of connected MOST units.
setSubcriptionVirtualControlInputs	13.3.46	Set a subscription for virtual control input state changes.
getVirtualControlInputNames	13.3.53	Gets the configured virtual control inputs from the NCO,
unitCount	13.4.10	Informs clients that the number of connected MOST units has changed.
virtualControlInputState	13.4.11	Informs clients that one or more virtual control inputs have changed state.
getConfiguredUnits	13.3.55	Gets the configured units from the NCO.
getConnectedUnits	13.3.56	Gets the connected units from the NCO.

The **PraesideoOpenInterface_V360** interface implements all the definitions of the **PraesideoOpenInterface V3.40** interface plus the following definitions:

table 15.29: Diagnostic event types

TOIVirtualControlInputDeactivation
OIVCI_STOP
OIVCI_ABORT

table 15.30: Diagnostic event types

TOIVirtualControlInputState
OIVCIS_ACTIVE
OIVCIS_INACTIVE

The following Diagnostic Event Types are also sent in the Praesideo V3.60 interface:

table 15.31: Diagnostic event types

TOIDiagEventType
OIDET_NOFAULTS
OIDET_BACKUPPOWERMODESTART
OIDET_BACKUPPOWERMODEEND

15.10 Praesideo V4.1 Interface(s)

The main interface of the V4.1 Praesideo Open Interface is named **PraesideoOpenInterface_V410** while the other interfaces (**IDiagEvent_V410**, **IPraesideoOpenInterfaceEvents_V410** and **IEventOriginator_V410**) have the same name as described in the document.

The **PraesideoOpenInterface_V410** interface implements all the definitions of the **PraesideoOpenInterface_V3.60** interface.

The **IPraesideoOpenInterfaceEvents_V0410** interface defines all the events of the **IPraesideoOpenInterfaceEvents_V0360** interface.

The **IDiagEvent_V410** interface implements all the diag event of the **IDiagEvent_V360** interface plus the following function:

table 15.32: Implemented events

Function	Section	Notes
getDiagZoneNames	13.5.43	Get zone names with a zone line fault from the event

The following Diagnostic Event Types is also sent in the Praesideo V4.10 interface:

table 15.33: Diagnostic event types

TOIDiagEventType
OIDET_ZONELINEFAULT

15.11 Praesideo V4.3 Interface(s)

The main interface of the V4.3 Praesideo Open Interface is named **PraesideoOpenInterface_V430** while the other interfaces (**IDiagEvent_V430**, **IPraesideoOpenInterfaceEvents_V430** and **IEventOriginator_V430**) have the same name as described in the document.

The **PraesideoOpenInterface_V430** interface implements all the definitions of the **PraesideoOpenInterface_V4.10** interface.

The **IPraesideoOpenInterfaceEvents_V0430** interface defines all the events of the **IPraesideoOpenInterfaceEvents_V0410** interface.

The following Diagnostic Event Types are also sent in the Praesideo V4.30 interface:

table 15.34: Diagnostic event types

TOIDiagEventType
OIDET_OMNEOINTERFACE
OIDET_OMNEONETWORK
OIDET_AMPFANFAULT

Note
The routing and call content are identified by unique names. These names are entered with the configuration webpages during the installation of the system. Before version 3.1, it was not possible to retrieve these unique names from the system via the Open Interface communication. Instead, some customers have been retrieving configuration information from the NCO via webpage content for use in an Open Interface client application. This function was never supported but no alternative other than manual input was available at that moment in the Open Interface.
However, from version 3.1 onwards retrieving this type of information is available in the Open Interface as supported functions for retrieving zone names, zone group names, message names, chime names, audio input names and BGM channel names.
In version 3.4 of Praesideo the web configuration pages have been redone completely to make the visual presentation and user interface less depending on the browser program and version. As a consequence of this update the data transfer between the browser and server in the NCO has changed. Normally that will not be a problem, but the former method of retrieving configuration information will not work anymore (without making changes) and is strongly advised against. Use the supported Open Interface commands instead.

15.12 Protocol differences between V2.0, V2.1, V2.2, V2.3, V3.0, V3.1, V3.3, V3.4, V3.6 and V4.1

For support to non VB- and COM-clients and additional functionality, the protocol used between the COM-Server and the Praesideo NCO has been changed since V2.0. This section provides a global overview of the protocol changes and how to deal with the different versions.

Differences:
Due to the change of the protocol different ports are used for the communication between the PC and the Praesideo NCO. V2.0 and V2.1 required that TCP/IP port number 12345 is passed in the connect function of the Open Interface. When using V2.2 or V2.3 the port number is obsolete and any value passed will be ignored. The COM-Server always uses TCP/IP port number 9401 for the communication.

Dealing with differences:
When the client system needs to be operating with different Praesideo system versions, the software can follow the following steps to determine the interface and the available functions:

- Create a V2.0 Open Interface Object.
- Query the version of the open interface using the **getVersion** function (see section 13.3.4).
- If the version is V2.0, then use the created object.
- If the version is V2.1, then free the object and create a V2.1 Open Interface Object. Use that object for calling the functions.
- If the version is V2.2, then free the object and create a V2.2 Open Interface Object.
- If the version is V2.3, then free the object and create a V2.3 Open Interface Object.
- If the version is V3.0, then free the object and create a V3.0 Open Interface Object.
- If the version is V3.1, then free the object and create a V3.1 Open Interface Object.
- If the version is V3.3, then free the object and create a V3.3 Open Interface Object.
- If the version is V3.4 or 3.5, then free the object and create a V3.4 Open Interface Object.
- If the version is V3.6, then free the object and create a V3.6 Open Interface Object.

- If the version is V4.1, then free the object and create a V4.1 Open Interface Object.
- Connect to the Praesideo system using the connect function (see section 13.3.1). If you receive an exception then your are dealing with a Praesideo System prior to V2.2. Upgrade the Praesideo System.
- Check the NCO version using the function **getNcoVersion** (see section 13.3.4). The interface version and the NCO version should be equal. If unequal then you are probably dealing with a newer version of the Praesideo System and you should upgrade the Open Interface COM-server.

Note
With the Open Interface Library version V2.2 it is not possible to control and/or monitor older Praesideo Systems (version V2.1 or older), because of internal changes.

- Use the functions and events as defined conform the version dealing with, see section 15.1, 15.2 and 15.3 for the previous versions and section 13.3 and 13.4 for version V2.3.

Diagnostic events that are not defined in a certain version shall not be sent to the client when using an interface of this version. However:

- When connecting a V2.2 interface on the V3.0 library, new (post V2.2) events are not sent to the client application.
- When connecting a V2.2 interface on the V2.3 library, new (post V2.2) events are sent to the client application.

Bosch Security Systems B.V.
Torenallee 49
5617 BA Eindhoven
The Netherlands
www.boschsecurity.com