DCN SWSMD
Streaming Meeting Data

BOSCH

en | Interface Manual

Bosch Security Systems | 2013 July

Table of Contents

1. Introduction **4**
1.1 Purpose 4
1.2 Scope 4
1.3 Definitions, Acronyms and Abbreviations 4

2. Streaming Meeting Data (DCN-SWSMD) **5**
2.1 Issues to know: 5

3. Configuration **6**
3.1 Description 6
3.2 Registering a TcpIpActivityTraceListener(s) 6
3.3 Defining the TcpIpActivityTraceListener 7
3.3.1 IPAddress 7
3.3.2 ConnectionPort 7
3.3.3 AllowedClients 7
3.3.4 TopicFilter 7
3.3.5 ActivityTypeFilter 8
3.3.6 MaxQueuedItems 9
3.4 Multiple TcpIpActivityTraceListeners 9

4. Activities **10**
4.1 Activity details 10
4.2 Activity containers 13
4.2.1 Translation from activity data into XML 17

5. Creating a customized client **19**
5.1 Connecting with the TcpIpActivityTraceListener 19
5.2 Receiving the data 19
5.3 Deserializing the received activities 20
5.4 Lists 22

Appendix A. ADDITIONAL INFO **24**
A.1. Enumerations 24
A.2. Example XML Strings 26
A.2.1. SeatUpdated 26
A.2.2. InterpretationTranslationStarted 26
A.2.3. MeetingStarted 27
A.2.4. MeetingStopped 28
A.2.5. VotingStopped 28
A.2.6. MicrophoneTestStopped 28

1. INTRODUCTION

1.1 Purpose

This document is the DCN-SWSMD user manual. The user manual describes how to configure the DCN NG server to send the activities and how to create a Client that can receive these activities.

1.2 Scope

This document provides information for the customer that wants to create a Client application showing meeting data.

This document is intended for 3rd party software developers.

The code examples within this document are based on C# from framework 3.5.

1.3 Definitions, Acronyms and Abbreviations

Activity	An activity that happens in the server, e.g. MeetingStarted
Topic	A Topic defines a category in which a group of related activities is located.
C#	An object oriented programming language developed by Microsoft as part of the .NET initiative.
TcpIpActivityTraceListener	Standard Windows activity listener

2. STREAMING MEETING DATA (DCN-SWSMD)

Streaming Meeting Data can be used to create customizable videoscreens contain meeting data.

Streaming Meeting Data unlike other modules does not require any user action to operate; it is a software stream from the Bosch DCN Conference Software Server over an Ethernet connection.

The stream consists of XML data which can be received by any number of custom made PC software clients.

2.1 Issues to know:

- **License Key**
  The stream is only available when the License key in the CCU contains DCN-SWSMD.
  When the license key in the CCU does not contain DCN-SWSMD then activities are not queued.
- **Configuration**
  By default the DCN-SWSMD functionality is enabled and does not need any configuration settings.
  - **Access Control**
    Access control for custom made clients is done via a list of allowed IP addresses. When a client tries to connect from an IP address which is not present in the list then the connection is rejected.
    This list of allowed IP addresses is optional and all clients are accepted when this list is not set.
  - **Filtering**
    Filter options are available to define which type of activities are streamed. Filtering is optional and when no filtering is configured all activities are streamed.
  - **Queues**
    - The size of the queue in which the activities are placed before they are streamed can be defined for both the connected (at least one client is connected) and the disconnected state.
    - The activity queue will hold queued activities even when no clients are connected. When a client connects it will cause the queue to start sending the queued activities after which they are removed from the queue.
    - The activity queue has a limited size. Queuing activities into a full queue will cause the oldest activities to be removed. The activities which are removed from the queue will be logged as a warning into the event log.
- **Performance**
  On a typical system, running without any problems, meeting data is communicated to the connected clients within 1 second after a change is initiated. This response time does not include the processing time of the external client(s).
- **Bidirectional**
  The stream is not bidirectional, any data send from a custom made client to Bosch DCN Conference Software Server is ignored.

3. CONFIGURATION

3.1 Description

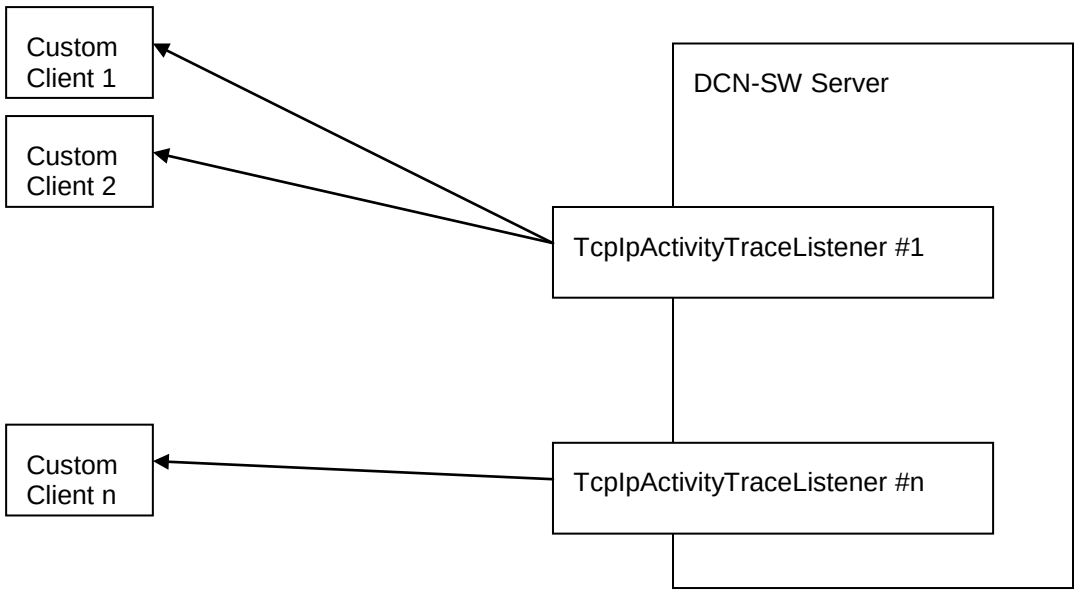
By default the DCN-SWSMD functionality is enabled and does not need any configuration settings; if advanced settings are not needed this chapter can be skipped.

DCN-SWSMD can be used to receive activities from the Bosch DCN Conference Software Server. As an example it is possible to receive an activity when a meeting is started. With this activity the sessions for the meeting, the participants, etc. will also be sent.

The DCN-SWSMD functionality is only available when the License key in the CCU or NCO contains DCN-SWSMD. Without this correct License key no activities are communicated with the connected client(s) via TCP/IP.

The configuration of the DCN-SWSMD is done via the configuration file of the server (..\Program Files\Bosch\Digital Congress Network\DCN-SW\Server.exe.config). In this file multiple activity loggers can be defined, each with their own filter settings.

The following describes a possible setup with multiple TcpIpActivityTraceListeners running at the server side. These listeners are responsible for waiting for custom clients that want to connect. It is possible that multiple custom clients are connected with the same TcpIpActivityTraceListener. The communication between server and custom clients is only one way as depicted by the arrow.



3.2 Registering a TcpIpActivityTraceListener(s)

The configuration of the DCN-SWSMD is done via the configuration file of the server (..\Program Files\Bosch\Digital Congress Network\DCN-SW\Server.exe.config). In this file multiple activity loggers can be defined, each with their own filter settings.

An activity logger can be set up to listen at a certain TCP/IP port for custom clients that want to connect. The filters are used to pass through only specific data.The following example is taken from the server configuration file where it was added within the <sources> region:

```
<source name="ActivityCategory" switchName="ActivitySwitch"
switchType="System.Diagnostics.SourceSwitch">
    <listeners>
        <remove name="Default"/>
        <add name="[ListenerName]"/>
    </listeners>
</source>
```

Where `[ListenerName]` can be any name for the TcpIpActivityTraceListener

3.3 Defining the TcpIpActivityTraceListener

The server configuration file needs to be extended so that the following is added within the `<sharedlisteners>` region:

```
<add name="[ListenerName]"
     type="Bosch.Dcn.Ecpc.Server.Services.TraceListeners.TcpIpActivityTraceListener,
    Bosch.Dcn.Ecpc.Server.Services"
     IpAddress="192.168.1.1"
     ConnectionPort="20000"
     AllowedClients="BDAZ1063, localhost, 192.168.10.3"
     TopicFilter="Meeting, Session"
     ActivityTypeFilter="MeetingStarted, SessionStarted"
     MaxQueuedItems="50, 100"/>
```

Where the `[ListenerName]` is set to the `[ListenerName]` as configured in the paragraph “Registering a TcpIpActivityTraceListener”.

A new activity logger is created with the name `[ListenerName]`.
The type defines the type and the namespace of the trace listener that is instantiated when an activity is traced.

3.3.1 IPAddress

This defines the IP address on which the external clients should connect. This is used when multiple IP addresses are available on the computer on which the server is running.

This field is not mandatory. If it is not set then default the activity logger will accept the connection of external clients on all IP addresses available on the server.

3.3.2 ConnectionPort

This defines the port at which this specific activity logger should listen for new clients. This field is mandatory. This specific activity logger will not start when this is not set.

3.3.3 AllowedClients

This can be used to only allow certain clients to connect. These clients can be defined by entering the computer name or by entering IP addresses. This field is not mandatory, if it is not present then default all external clients are allowed to connect.

Usage

```
AllowedClients="[Client][, [Client]]"
```

Where `[Client]` can either be a host name or a valid IP address.

3.3.4 TopicFilter

This allows filtering the activities at the topic level. It is possible to filter on multiple topics.This field is not mandatory; when it is not set then default no filtering takes place and all activities are send to the custom clients.

NOTE: This is only true when the ActivityTypeFilter is not set. Else the activities will also be filtered by the activity type.

Usage

```
TopicFilter="[Topic][, [Topic]]"
```

Where `[Topic]` can contain one of the following values:

- System
- Meeting
- Session
- Discussion

- Participant
- Seat
- Voting
- Interpretation
- ServiceCall
- Booth
- Desk
- TestSystem

3.3.5 ActivityTypeFilter

This allows filtering the activities based on their type. It is possible to set up the filtering so that only one specific activity is send to the connected clients. Pay attention that in most cases it does not make sense to combine the TopicFilter and the ActivityTypeFilter, use either none or only one.

This field is not mandatory, when it is not set then default no filtering takes place on activity types.

Usage

```
ActivityTypeFilter="[ActivityType][, [ActivityType]]"
```

Where `[ActivityType]` can contain one of the following values:

- SystemStarted
- SystemStopped
- MeetingDataUpdated
- MeetingStarted
- MeetingStopped
- ParticipantAdded
- AttendanceRegistrationStarted
- AttendanceRegistrationStopped
- SessionDataUpdated
- SessionStarted
- SessionStopped
- SessionSuspended
- SessionResumed
- DiscussionDataUpdated
- RequestListUpdated
- ResponseListUpdated
- ActiveListUpdated
- ActiveResponseListUpdated
- SpecialEquipmentListUpdated
- VotingDataUpdated
- VotingStarted
- VotingStopped
- VotingOnHold
- VotingResumed
- VotingSelected
- VotingInterimResults
- SeatAdded
- SeatUpdated
- SeatRemoved
- SeatPriorityButtonActivated
- SeatPriorityButtonDeactivated
- ParticipantUpdated
- InterpretationTranslationStarted
- InterpretationTranslationStopped
- ServiceCallStarted
- ServiceCallIsBeingServiced

- ServiceCallHandled
- ServiceCallCanceled
- BoothInUse
- BoothNotInUse
- DeskAdded
- DeskUpdated
- DeskRemoved
- ChannelTestStarted
- ChannelTestStopped
- MicrophoneTestStarted
- MicrophoneTestEnded
- MicrophoneTestCanceled
- MicrophoneTestFailed

3.3.6 MaxQueuedItems

The activities are placed in a queue when they occur. After that they are handled sequentially onto a separate thread so that the oldest in the queue is handled first. This ensures that in busy periods where a lot of activities occur the activities are still send in the correct sequence to the clients. However to prevent the queue to fill up with too much activities it is possible to set the maximum number of activities that are queued. When an activity is queued into a full queue it will cause the oldest activity in the queue to be removed. This activity will not be sent to the external clients, it will however be logged in the Windows event log.

When setting the MaxQueuedItems one should pass 2 numbers. The first number represent the queue size when no clients are connected and the second number is used when at least 1 external client is connected.

If the activity logger is running without any connected external clients then the activities that occur are queued and stored. The connection of an external client triggers the activity logger and all queued activities are then sent to the external client.

NOTE: If the queued activities are sent to the external client then they are removed from the queue. So new external clients will only receive new activities.

This field is not mandatory, when it is set, it replaces the default queue size of 50.500 activities.

Usage

```
MaxQueuedItems="MaxItemsInDisconnectedMode, MaxItemsInConnectedMode"
```

Where `[MaxItemsInDisconnectedMode]` can contain one of the following values: 0 – 1000
Where `[MaxItemsInConnectedMode]` can contain one of the following values: 0 - 1000

3.4 Multiple TcpIpActivityTraceListeners

It is possible to define more then one TcpIpActivityTraceListener. The advantage is that each trace listener can have their own specific filter settings. To do this create and register another TcpIpActivityTraceListener. Note that the name of the new created listener must be different then the first defined listener.

4. ACTIVITIES

4.1 Activity details

The activities are divided in categories; we call these categories ‘Topics’. The following table describes the topics with the corresponding activities.

Topic	Activity	Description
System	SystemStarted	This is send when the DCN-SW server is started and it is send only once.
	SystemStopped	This is send when the DCN-SW server is stopped and it is send only once.
Meeting	MeetingDataUpdated	This is send when the meeting data of the running meeting is updated. The concerning data is the subject, the description and the date/time at which the meeting is scheduled.
	MeetingStarted	This is send when a meeting is started. The available sessions and participants for this meeting are sent.
	MeetingStopped	This is send when a meeting is stopped. Only the meeting identifier is sent.
	ParticipantAdded	This is send when a participant is added to the running meeting.
	AttendanceRegistrationStarted	This is send when attendance registration is started .
	AttendanceRegistrationStopped	This is send when attendance registration is stopped .
Session	SessionDataUpdated	This is send when the data of the running session is updated. Only the identifier and the session data container are sent.
	SessionStarted	This is send when a session is started. The information regarding the session including the votings is send.
	SessionStopped	This is send when a session is stopped. Only the session identifier is sent.
	SessionSuspended	This is send when a session is suspended. Only the session identifier is sent.
	SessionResumed	This is send when a session is resumed. Only the session identifier is sent.
Discussion	DiscussionDataUpdated	This is send when the data of the running discussion is updated. Only the identifier and the discussion data are sent.
	RequestListUpdated	This is send when the request list is updated. Only the discussion identifier and the list are sent.
	ResponseListUpdated	This is send when the response list is updated. Only the discussion identifier and the list are sent.
	ActiveListUpdated	This is send when the active list is updated. Only the discussion identifier and the list are sent.
	ActiveResponseListUpdated	This is send when the active response list is updated. Only the discussion identifier and the list are sent.
	SpecialEquipmentListUpdated	This is send when the special equipment list is updated. Only the discussion identifier and the list are sent.
Voting	VotingDataUpdated	This is send when the data of a voting is updated. Only the identifier and the voting data container are sent.
	VotingStarted	This is send when the voting is started. The identifier, the voting data and the possible answers are sent.

Topic	Activity	Description
	VotingOnHold	This is send when the running voting is held. Only the voting identifier and the voting data are sent.
	VotingResumed	This is send when the held voting is resumed. Only the voting identifier and the voting data are sent.
	VotingStopped	This is send when the running voting is stopped. Only the voting identifier and the voting data are sent.
	VotingSelected	This is send when another voting is selected. Only the voting identifier and the voting data are sent.
	VotingInterimResult	This is send when interim voting results are enabled. These interim results contain the changed values compared to the previous VotingInterimResult activities.
Seat	SeatAdded	This is send when a seat is added on the DCN-SW server.
	SeatUpdated	This is send when the data regarding a seat is updated.
	SeatRemoved	This is send when a seat is removed from the DCN-SW server.
	SeatPriorityButtonActivated	This is send when the priority button is pressed on a chairman unit. It includes the seat data and the seated participant.
	SeatPriorityButtonDeactivated	This is send when the priority button is released on a chairman unit. It includes the seat data and the seated participant.
Participant	ParticipantUpdated	This is send when the data regarding a participant is updated. It includes the participant data, the seat and the group if available.
Interpretation	InterpretationTranslationStarted	This is send when the translation is started.
	InterpretationTranslationStopped	This is send when the translation is stopped.
ServiceCall	ServiceCallStarted	This is send when the usher is called.
	ServiceCallIsBeingServiced	This is send when the usher starts servicing the call.
	ServiceCallHandled	This is send when the usher has handled the call.
	ServiceCallCanceled	This is send when the call is canceled.
Booth	BoothInUse	This is send when the booth is in use.
	BoothNotInUse	This is send when the booth is no longer in use.
Desk	DeskAdded	This is send when a desk is added on the DCN-SW server.
	DeskUpdated	This is send when the data regarding a desk is updated.
	DeskRemoved	This is send when a desk is removed on the DCN-SW server.
TestSystem	ChannelTestStarted	This is send when the channel test is started.
	ChannelTestStopped	This is send when the channel test is stopped.
	MicrophoneTestStarted	This is send when the microphone test is started.
	MicrophoneTestEnded	This is send when the microphone test is ended.
	MicrophoneTestCanceled	This is send when the microphone test is canceled.

Topic	Activity	Description
	MicrophoneTestFailed	This is send when the microphone test is failed. Microphone test is considered as failed when user does not get any response from CCU regarding the test results.

The following table describes the activities and the corresponding data in more detail. To describe which data is send or not, is described in the data column as follows:

<Seat> → All items within seat are filled in if possible. If no participant is seated then the <Participant> field within seat is not filled in.

<Seat(Id)> → Only the identifier within seat is filled in. The rest of the fields are not set even if a participant is seated.

Activity	Data
SystemStarted	<ActivityInfo>
SystemStopped	<ActivityInfo>
MeetingDataUpdated	<ActivityInfo><Meeting(Id, <MeetingData>)>
MeetingStarted	<ActivityInfo><Meeting>
MeetingStopped	<ActivityInfo><Meeting(Id)>
ParticipantAdded	<ActivityInfo><Meeting(Id, <Participant>)>
AttendanceRegistrationStarted	<ActivityInfo><Meeting(Id)>
AttendanceRegistrationStopped	<ActivityInfo><Meeting(Id)>
SessionDataUpdated	<ActivityInfo><Session(Id, <SessionData>)>
SessionStarted	<ActivityInfo><Session>
SessionStopped	<ActivityInfo><Session(Id)>
SessionSuspended	<ActivityInfo><Session(Id)>
SessionResumed	<ActivityInfo><Session(Id)>
DiscussionDataUpdated	<ActivityInfo><Discussion(Id,<DiscussionData>)>
RequestListUpdated	<ActivityInfo><Discussion(Id, <RequestList>)>
ResponseListUpdated	<ActivityInfo><Discussion(Id, <ResponseList>)>
ActiveListUpdated	<ActivityInfo><Discussion(Id, <ActiveList>)>
ActiveResponseListUpdated	<ActivityInfo><Discussion(Id, <ActiveResponseList>)>
SpecialEquipmentListUpdated	<ActivityInfo><Discussion(Id, <SpecialEquipmentList>)>
VotingDataUpdated	<ActivityInfo><Voting(Id, <VotingData>)>
VotingStarted	<ActivityInfo><Voting(Id)>
VotingOnHold	<ActivityInfo><Voting(Id)>
VotingResumed	<ActivityInfo><Voting(Id)>
VotingStopped	<ActivityInfo><Voting>
VotingSelected	<ActivityInfo><Voting(Id, <VotingData>)>
VotingInterimResult	<ActivityInfo><Voting>
SeatAdded	<ActivityInfo><Seat>
SeatUpdated	<ActivityInfo><Seat>
SeatRemoved	<ActivityInfo><Seat(Id)>
SeatPriorityButtonActivated	<ActivityInfo><Seat>
SeatPriorityButtonDeactivated	<ActivityInfo><Seat>
ParticipantUpdated	<ActivityInfo><Participant>
InterpretationTranslationStarted	<ActivityInfo><Desk>
InterpretationTranslationStopped	<ActivityInfo><Desk>
ServiceCallStarted	<ActivityInfo><ServiceCall(Id, <Seat>)>
ServiceCallIsBeingServiced	<ActivityInfo><ServiceCall(Id, <Seat>)>
ServiceCallHandled	<ActivityInfo><ServiceCall(Id, <Seat>)>
ServiceCallCanceled	<ActivityInfo><ServiceCall(Id, <Seat>)>
BoothInUse	<ActivityInfo><Booth(BoothNumber)>
BoothNotInUse	<ActivityInfo><Booth(BoothNumber)>
DeskAdded	<ActivityInfo><Desk>
DeskUpdated	<ActivityInfo><Desk>
DeskRemoved	<ActivityInfo><Desk(Id)>
ChannelTestStarted	<ActivityInfo>
ChannelTestStopped	<ActivityInfo>
MicrophoneTestStarted	<ActivityInfo>
MicrophoneTestEnded	<ActivityInfo><MicrophoneTestResults>
MicrophoneTestCanceled	<ActivityInfo>
MicrophoneTestFailed	<ActivityInfo>

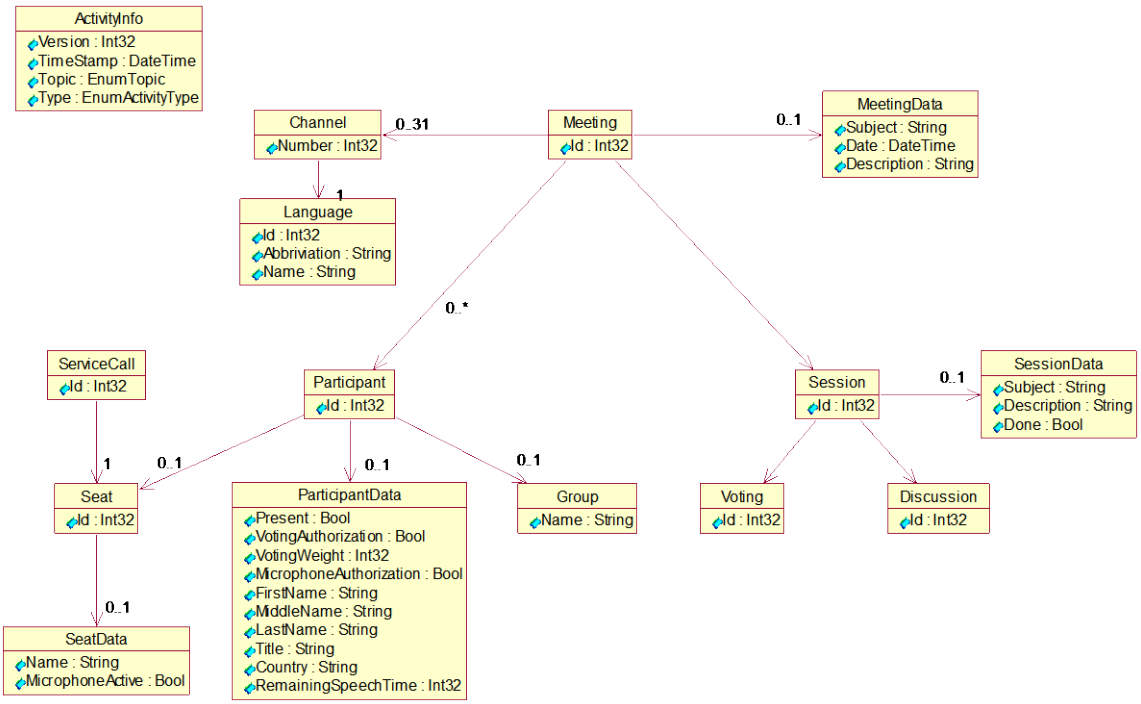
4.2 Activity containers

This paragraph describes containers that are used to transfer the activity data onto the connected custom clients. Some data within these containers is optional and will not always be available in the received XML string. Optional fields are embraced with the [ ] characters. It is possible that these optional fields are not present in the XML string when an activity occurs.

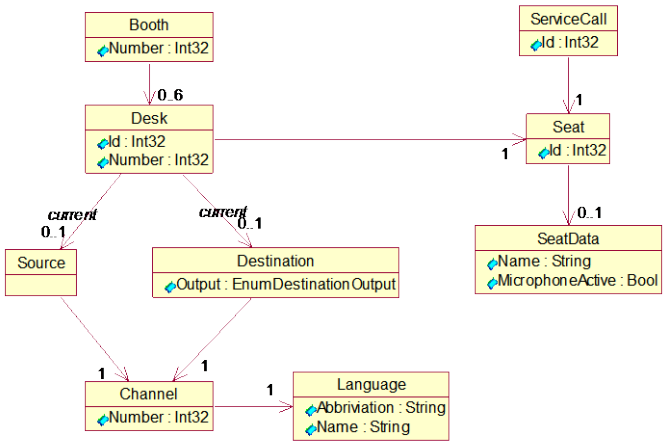
A container may contain references to other containers. These referenced containers are embraces with the <> characters.

When there can be reference to a list of containers of the same type we use the 'List of' before the container.

The next diagram shows the relations of the containers for the meeting activity:



The next diagram shows the relations of the containers for the interpretation activity:



The next tables describe the containers in more detail. The first column contains the attributes in the container and the type while the second column describes the attribute in more detail.

The following types are recognized as attributes:

- **Int32**
  An integer of 32 bits which can contain a number in the range from -2147483648 to 2147843647. This is serialized as a string into the XML string.

- **DateTime**
  This type contains a time on a certain date. This date/time is serialized into the XML string in the format “2008-08-27T13:00:51.2126576+02:00” where 13:00:51 is according the UTC time zone. The +02:00 indicates the zone offset, so the actual time is 15:00:51.

- **EnumTopic**
  A custom enumerator defined in A.1.This is serialized as a string in the XML string.

- **EnumActivityType**
  A custom enumerator defined in A.1. This is serialized as a string in the XML string.

- **EnumDestinationOutput**
  A custom enumerator defined in A.1. This is serialized as a string in the XML string.

- **Bool**
  Indicates that a flag is either true or false. This is serialized as a string into the XML string as either “True” or “False”.

- **String**
  Directly serialized into the XML string.

- **StopWatchState**
  A Custom enumerator defined for Group Speech time. This is serialized as a string in the XML string.

<ActivityInfo>	
Version (Int32)	The version of the data definition for this activity. This will only change when updates take place on the data definition during development.
Timestamp (DateTime)	The date/time at which this activity occurred.
Topic (string from EnumTopic)	Defines the topic of this activity.
Type (string from EnumActivityType)	Defines the specific type of the activity.

<Meeting>	
Id (Int32)	The unique identifier for this meeting.
[<MeetingData>]	The data for the meeting.
List of <Session>]	The corresponding sessions for this meeting.
List of <Participant>]	The participants for this meeting.
[List of <Channel>]	The channels for this meeting.
[List of <Booth>]	The booths with their desks for this meeting.

<MeetingData>	
Subject (String)	The subject of this meeting.
DateTime (DateTime)	The date/time at which this meeting should start.
Description (String)	The description for this meeting.

<Session>	
Id (Int32)	Unique identifier for this session.
[<SessionData>]	The data that belongs to this session.
[<Discussion>]	The discussion that runs within this session.
[List of <Voting>]	The votings for this session.
[List of <Groups>]	This contains a list of groups.
[List of <Particpants>}	The participants in this list.

<SessionData>	
Subject (String)	The subject of this session.
Description (String)	The description of this session.
Done (bool)	Indicates if this session is done.

<Discussion>	
Id (Int32)	The unique identifier for this discussion.

<Discussion>	
[<DiscussionData>]	The data for this discussion.
[<RequestList>]	This contains the participants who are currently requesting to speak.
[<ResponseList>]	This contains the participants who are currently requesting to respond to an active speaker.
[<ActiveList>]	This contains the participants who are currently speaking.
[<ActiveResponseList>]	This contains the participants who are currently responding to a previously active speaker.
[<SpecialEquipmentList>]	This contains a list of special equipment (e.g. notebooks).
[List of <ActiveGroups>]	This contains a list of Active groups.

<DiscussionData>	
NumberOfActiveMicrophones (Int32)	The number of microphones which are currently active.

<RequestList>	
List of <Participant>	The participants in this list.

<ResponseList>	
List of <Participant>	The participants in this list.

<ActiveList>	
List of <Participant>	The participants in this list.

<ActiveResponseList>	
List of <Participant>	The participants in this list.

<SpecialEquipmentList>	
List of <Participant>	The participants in this list.

<Voting>	
Id (Int32)	The unique identifier for this voting.
[<VotingData>]	The data which belongs to this voting.
[<VotingResults>]	The results that belong to this voting.
[List of <VotingAnswers>]	The possible answers for this voting.

<VotingData>	
Name (String)	The name for this voting.
Subject (String)	The subject of this voting.
VotingType (String)	The type of this voting.
RemainingVotingTime (Int32)	The remaining voting time in seconds. Contains -1 when no end time is present.

<voting results>	
[List of <VotingIndividualResult>]	A list that contains the individual results for each participant for this voting.
[<VotingTotalResults>]	The total results for this voting.
[List of <VotingGroupResult>]	A list that contains the individual results for each group for this voting.

<VotingIndividualResult>	
<Participant>	The participant which is responsiblefor these voting results.
AnswerId (Int32)	The identifier for the answer given for this voting by this participant. This identifier can be mapped on the answers present in the <voting> container.

<VotingTotalResults>	
Approved (bool)	Voting is approved (or not) based on the settings for quorum and majority.
RequiredQuorum (Int32)	The required quorum.
ActualQuorum (Int32)	The actual quorum.
MaximumQuorum (Int32)	The maximum quorum.

<VotingTotalResults>	
RequiredMajority (Int32)	The required majority.
ActualMajority (Int32)	The actual majority.
MaximumMajority (Int32)	The maximum majority.
NumberOfAuthorizedPresentParticipants (Int32)	The number of authorized participants which are present for the voting.
NumberOfAuthorizedPresentParticipantsWithoutVote (Int32)	The number of authorized participants which are present and did not vote.
List of <VotingAnswerResult>	The list of answers and the related information for this voting.

<VotingGroupResult>	
List of <VotingAnswerResult>	The list of answers given within this group.
<Group>	The group responsible for these results.
NumberOfAuthorizedPresentParticipants (Int32)	The number of authorized participants which were present within this group for the voting.
NumberOfAuthorizedParticipantsWithout Vote (Int32)	The number of authorized participants which were present and did not vote.

<VotingAnswerResult>	
NumberOfCasts (Int32)	The number of casts for this answer.
Percentage (double)	The percentage of participants that voted this answer.
AnswerId	The identifier of the answer for which the results are meant. This identifier can be mapped on the answers present in the <voting> container.

<Answer>	
Id (Int32)	The unique identifier for this answer.
AnswerText (String)	The description for this answer.
LegendText (String)	The translated text for this answer (Currently not yet available).
Correct (bool)	Indicates that this answer is the correct one (or not).
Score (Int32)	When multiple answers are correct then this score will indicate which answer has the preference.

<Seat>	
Id (Int32)	The unique identifier for this seat.
[<SeatData>]	The data which belongs to this seat.
[<Participant>]	The participant which is seated at this seat.

<SeatData>	
Name (String)	The name of this seat
MicrophoneActive (bool)	Indicates if the microphone is active or not.
SeatType (Sting)	The type of this seat. • Chairman • Delegate • Interpreter • Operator • Unknown (unassigned seat)

<Group>	
Name (String)	The name for this group
RemainingGroupSpeechTime(Int32)	The remaining speech time for this group in seconds. Contains -1 when no end time is present.
StopWatchState(String from enum[StopWatchState])	Indicates the state of the stopwatch of the group.

<Participant>	
Id (Int32)	The unique identifier for this participant.
[<ParticipantData>]	The data which belongs to this participant.
[<Seat>]	The seat at which this participant is seated.
[<Group>]	The group in which this participant is located.

<ParticipantData>	
Present (bool)	Indicates that the participant is (or is not) present for the meeting.
VotingAuthorisation (bool)	Indicates that the participant is (or is not) authorized to vote.

VotingWeight (Int32)	The voting weight of the participant.
MicrophoneAuthorisation (bool)	Indicates that the participant is (or is not) authorized to use the microphone.
FirstName (string)	The first name of the participant.
MiddleName (string)	The middle name of the participant.
LastName (string)	The last name of the participant.
Title (string)	The title of the participant.
Country (string)	The country of the participant.
RemainingSpeechTime (Int32)	The remaining speech time for this participant in seconds. Contains -1 when no end time is present (In case the microphone mode is voice activated or Push-To-Talk this value is always -1).
SpeechTimerOnHold (bool)	Indicates that the speech timer is (or is not) on hold.

<ServiceCall>	
Id (Int32)	The unique identifier for this service call.
[<Seat>]	The seat at which this service call is initiated.

<Booth>	
Number (Int32)	The number of the booth.
[List of <Desk>]	The desk suited in this booth.

<Channel>	
Number (int32)	The channel number in the system.
[<Language>]	The language which belongs to this channel.

<Language>	
Abbreviation (string)	The abbreviation of the language.
Name (string)	The name of the language.

<Desk>	
Id (Int32)	The unique identifier for this desk.
Number (Int32)	The desk number.
[<Booth>]	Booth information of the desk.
[<Seat>]	Seat information of the desk
[<Source>]	The source channel used for translation.
[<Destination>]	The destination channel used for translation.

<Source>	
[<Channel>]	The channel used as source at the desk.

<Destination>	
Output (EnumDestinationOutput)	The desk output used for the translations.
[<Channel>]	The channel used for the destination.

<MicrophoneTestResults>	
List of <MicrophoneTestResult>	The corresponding microphone test result.

<MicrophoneTestResult>	
<Seat>	The seat which is tested.
Passed (bool)	Test is passed (or not) and the microphone is works correctly.

4.2.1 Translation from activity data into XML

The described activity data is translated into XML by using the following guideline:

- The data inside a container is added as attributes; e.g. <Seat id=”1”> where id is the attribute
- References to other containers are added as elements;
  e.g. :

```
<Seat>
      <SeatData Name=”Seat1” MicrophoneActive=”True”/>
</Seat>
```

Where `SeatData` is the element.

5. CREATING A CUSTOMIZED CLIENT

This describes the necessary actions to create a custom client in Microsoft Visual C# to receive and decode the XML stream of the **Bosch DCN Conference Software Server**. See also the DCN-SWSMD example code on the DVD.

5.1 Connecting with the TcpIpActivityTraceListener

Normally the default configuration does not need any changes.

If you did make changes to the Configuration you need to restart the DCN-SW server.

The default port to connect to is 20000.

If you do not use .Net framework to create the custom client it is recommended to use Microsoft XML library to deserialize the XML stream.

The following C# code can be used to connect with the TcpIpActivityTraceListener:

```
Socket clientSocket = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);

try
{
    // 20000 is the value that was set for ConnectionPort when defining the
    // TcpIpActivityTraceListener. The IP address that is used can only be
    // used in the situation that the client and server are
    // running on the same computer.
IPEndPoint ipEndPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 20000);
clientSocket.Connect(ipEndPoint);
}
catch (SocketException e)
{
    // A problem occurred … Check what happened.
}
finally
{
clientSocket.Close();
}
```

A connection should have been made successfully when the server was restarted and the configuration was successful.

5.2 Receiving the data

The data concerning an activity is received as follows:

- **Header**
  The header is binary and contains the topic that is contained in the activity and the length of the message that follows

- **Message**
  This is a string in which the activity is streamed as XML

The format of the header is as follows:

Content	Size in bytes	Description
`Topic`	4	The topic of the activity that is present in the message, this is in little-endian format. This is an Int32 and can be mapped onto the enum EnumTopic as described in Additional Info.
`Message length`	4	The length of the message that is located in the received data after this header, this is in little-endian format.

After the connection is succesfully made with the TcpIpActivityTraceListener one could do the following to retrieve the complete message:

```
While (!quit)
{
```

```
    // For each message a header is sent that contains 8 bytes.
    if (clientSocket.Available > 8)
    {
        // First we read the header.
        byte[] header = new byte[8];
        int numberOfReceivedBytes = clientSocket.Receive(header, 8, 0);

        if (numberOfReceivedBytes != 8)
            throw new InternalOperationException("Header receival problem");

        // The header contains 2 integers, the first 4 bytes contains the topic
        // and the remaining 4 bytes contain the length of the message that follows.
        Int32 topic = BitConverter.ToInt32(header, 0);
        Int32 messageLength = BitConverter.ToInt32(header, 4);
        byte[] buffer = new byte[messageLength];
        byte[] receivedData = new byte[messageLength];
        Int32 totalNumberOfBytesReceived = 0;
        StringBuilder receivedString = new StringBuilder();

        // It is possible that we do not receive the complete message in
        // one read. So expect multiple reads.
        while (totalNumberOfBytesReceived < messageLength)
        {
            numberOfReceivedBytes = clientSocket.Receive(buffer,
                messageLength - totalNumberOfBytesReceived, 0);

            // Add the received data to the buffer.
            Array.Copy(buffer, 0, receivedData, totalNumberOfBytesReceived,
                numberOfReceivedBytes);

            totalNumberOfBytesReceived += numberOfReceivedBytes;
        }
        receivedString.Append(System.Text.Encoding.Unicode.GetString(receivedData));
    }
}
```

After the call “receivedString.Append(…)” is executed the received XML string is present in the StringBuilder.

5.3 Deserializing the received activities

The easiest way to convert the XML string, received in the previous paragraph, into a collection of objects is to use the XML deserializing mechanism from the Framework. To do this, one should define classes with public properties that correspond to the properties as defined in Activities . Also ensure that the defined classes that are used to deserialize the XML data have a parameterless constructor since this is necessary for the deserialization process.

The following activities can be send from the DCN-SW server when Streaming Meeting Data is enabled:

SystemActivity, MeetingActivity, SessionActivity, DiscussionActivity, VotingActivity, SeatActivity, ParticipantActivity, InterpretationActivity, ServiceCallActivity, BoothActivity, DeskActivity, TestSystemActivity

These activities have some corresponding properties, so the best way to deserialize these is to have a generic base class that contains these properties. The next code fragment describes a possible implementation for this base class:

```
[XmlRootAttribute(ElementName = "Activity", IsNullable = false)]
public abstract class Activity
{
    protected Activity(){}

    /// <summary>The version of this activity. This can be used to check if the
    /// activity matches the expected version.</summary>
    [XmlAttribute]
    public Int32 Version { get; set; }

    /// <summary>The date/time at which the activity occurred.</summary>
    [XmlAttribute]
    public DateTime TimeStamp { get; set; }

    /// <summary>The topic of this activity.</summary>
    [XmlAttribute]
    public EnumTopic Topic { get; set; }

    /// <summary>The type of this activity.</summary>
    [XmlAttribute]
    public EnumActivityType Type { get; set; }
}
```

The specific activities are derived from this base class. As an example the participant activity is displayed:

```
[XmlRootAttribute(ElementName = "ParticipantActivity", IsNullable = false)]
public class ParticipantActivity : Activity
{
    public ParticipantActivity() {}

    public ParticipantContainer Participant { get; set ; }
}
```

The participant activity has one extra property next to the ones defined in the base activity. This is the participant container which contains the data regarding the participant.

The ParticipantContainer should be defined as follows:

```
[XmlRootAttribute(ElementName = "ParticipantContainer", IsNullable = false)]
public class ParticipantContainer
{
    public ParticipantContainer() {}

    /// <summary>The unique identifier for this participant.</summary>
    [XmlAttribute]
    public Int32 Id { get; set }

    /// <summary>The data for this participant.</summary>
    public ParticipantDataContainer ParticipantData { get; set }

    /// <summary>The seat at which this participant is located.</summary>
    public SeatContainer Seat { get; set }

    /// <summary>The group in which this participant is located.</summary>
    public GroupContainer Group { get; set }
}
```

The ParticipantDataContainer should be defined as follows:

```
[XmlRootAttribute(ElementName = "ParticipantDataContainer", IsNullable = false)]
public class ParticipantDataContainer
{
    public ParticipantDataContainer() {}

    /// <summary>Indicates that the current participant is present or not.</summary>
    [XmlAttribute] public bool Present { get; set; }
```

```
    /// <summary>The voting weight for this participant.</summary>
    [XmlAttribute] public Int32 VotingWeight { get; set; }

    /// <summary>Is this participant authorized to vote.</summary>
    [XmlAttribute] public bool VotingAuthorisation { get; set; }

    /// <summary>Is this participant authorized to use the microphone.</summary>
    [XmlAttribute] public bool MicrophoneAuthorisation { get; set; }

    /// <summary>The first name of this participant.</summary>
    [XmlAttribute] public string FirstName { get; set; }

    /// <summary>The middle name of this participant.</summary>
    [XmlAttribute] public string MiddleName { get; set; }

    /// <summary>The last name of this participant.</summary>
    [XmlAttribute] public string LastName { get; set; }

    /// <summary>The title of this participant.</summary>
    [XmlAttribute] public string Title { get; set; }

    /// <summary>The country set for this participant.</summary>
    [XmlAttribute] public string Country { get; set; }

    /// <summary>The remaining speech time in seconds for this
participant.</summary>
    [XmlAttribute] public Int32 RemainingSpeechTime { get; set; }
}
```

After defining the StationContainer and the GroupContainer one is ready to deserialize the ParticipantActivity.

In one of the previous examples we had the “receivedString” that contained the XML string and the header that was send in front of it.

The message header that is received when an activity is received contains the topic of the activity. When this topic indicates that the ParticipantActivity is present in the message we can deserialize it. For this we need the next 2 methods:

```
/// <summary>Converts the String to UTF8 Byte array and is used in
deserialization.</summary>
private static Byte[] StringToUTF8ByteArray(string pXmlString)
{
    UTF8Encoding encoding = new UTF8Encoding();
    byte[] byteArray = encoding.GetBytes(pXmlString);
    return byteArray;
}

/// <summary>Reconstruct an object from an XML string</summary>
public T DeserializeObject<T>(string xml)
{
    XmlSerializer xs = new XmlSerializer(typeof(T));
    MemoryStream memoryStream = new MemoryStream(StringToUTF8ByteArray(xml));
    XmlTextWriter xmlTextWriter = new XmlTextWriter(memoryStream, Encoding.UTF8);
    return (T)xs.Deserialize(memoryStream);
}
```

The deserializing of the ParticipantActivity can be done as follows:

```
ParticipantActivity participantActivity =
    DeserializeObject<ParticipantActivity>(receivedString.ToString());
```

After this statement the participantActivity will contain the data regarding this activity. The related objects, like the StationContainer will also be set if that data is available in the XML string. This same mechanism could be used for all the other activities.

This process should be duplicated for the remaining activities when these also need to be deserialized.

5.4 Lists

In some occasions a list of items is send within a container. The way to handle this will be explained by describing the meeting container since this contains a list of ParticipantContainers.

```
[XmlRootAttribute(ElementName = "MeetingContainer", IsNullable = false)]
public class MeetingContainer
{
    ...

    /// <summary>The collection of participants within this meeting.</summary>
    public List<ParticipantContainer> Participants { get; set; }

    ...
}
```

APPENDIX A. ADDITIONAL INFO

A.1. Enumerations

The enumerations EnumTopic and EnumActivityType should be defined as follows:

```
public enum EnumTopic
{
    System = 0,
    Meeting = 1,
    Session = 2,
    Discussion = 3,
    Participant = 4,
    Seat = 5,
    Voting = 6
    Interpretation = 7,
    ServiceCall = 8,
    Booth = 9,
    Desk = 10
    TestSystem = 11
}
```

```
public enum EnumActivityType
{
    // System topic
    SystemStarted,
    SystemStopped,

    // Meeting topic
    MeetingDataUpdated,
    MeetingStarted,
    MeetingStopped,

    AttendanceRegistrationStarted,
    AttendanceRegistrationStopped,

    ParticipantAdded,

    // Session topic
    SessionDataUpdated,
    SessionStarted,
    SessionStopped,
    SessionSuspended,
    SessionResumed,

    // Discussion topic
    DiscussionDataUpdated,
    RequestListUpdated,
    ResponseListUpdated,
    ActiveListUpdated,
    ActiveResponseListUpdated,
    SpecialEquipmentListUpdated,

    // Seat topic
    SeatAdded,
    SeatUpdated,
    SeatRemoved,
    SeatPriorityButtonActivated,
    SeatPriorityButtonDeactivated,

    // Participant topic
    ParticipantUpdated,

    // Voting topic
    VotingDataUpdated,
    VotingStarted,
    VotingStopped,
    VotingOnHold,
    VotingResumed,
    VotingSelected,
    VotingInterimResult,
```

```
    // Interpretation topic
    InterpretationTranslationStarted,
    InterpretationTranslationStopped,

    // ServiceCall topic
    ServiceCallStarted,
    ServiceCallIsBeingServiced,
    ServiceCallHandled,

    // Booth topic
    BoothInUse,
    BoothNotInUse,

    // Desk topic
    DeskAdded
}
```

```
public enum EnumDestinationOutput
{
    A = 0,
    B = 1
}
```

```
public enum StopWatchState
{
    /// <summary>Idle state of the stopwatch</summary>
    STOPWATCH_IDLE = 0,
    /// <summary>Running state of the stopwatch</summary>
    STOPWATCH_RUNNING,
    /// <summary>Interrupted state of the stopwatch</summary>
    STOPWATCH_INTERRUPTED
}
```

A.2. Example XML Strings

To make things more understandable a few examples of XML strings are displayed here.

A.2.1. SeatUpdated

When an update occurs on the seat then a string will be received in which the topic is set to Seat and the ActivityType is set to SeatUpdated. Based on the table we should expect this:

SeatUpdated	<activity info><seat>

The xml string received:

```
<?xml version="1.0" encoding="utf-8"?>
<SeatActivity xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" Version="1"
    TimeStamp="2008-08-27T13:00:51.2126576+02:00" Topic="Seat" Type="SeatUpdated">
  <Seat Id="6">
    <SeatData Name="0006" MicrophoneActive="true" />
    <Participant Id="7">
      <ParticipantData Present="false" VotingWeight="1" VotingAuthorisation="true"
          MicrophoneAuthorisation="true" FirstName="0007" MiddleName=""
          LastName="" Title="" Country="" RemainingSpeechTime="-1" />
    </Participant>
  </Seat>
</SeatActivity>
```

As one can see the participant node does not contain a tag for the group. This means that this participant was not assigned to a group.

A.2.2. InterpretationTranslationStarted

```
<?xml version="1.0" encoding="utf-8"?>
<InterpretationActivity xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" Version="1"
    TimeStamp="2008-08-27T13:00:51.2126576+02:00" Topic="Interpretation"
    Type="InterpretationTranslationStarted">
  <Desk Number=”1” />
    <Booth Number=”1” />
    </Booth>
    <Seat Id=”25”>
      <SeatData Name=”1:1” MicrophoneActive=”true” />
    </Seat>
    <Source>
      <Channel Number=”0” />
        <Language Abbreviation=”FLR” Name=”Floor” />
      </Channel>
    </Source>
    <Destination Output=”A”>
      <Channel Number=”2” />
        <Language Abbreviation=”NLD” Name=”Dutch” />
      </Channel
    </Destination>
  </Desk>
</InterpretationActivity>
```

A.2.3. MeetingStarted

```
<?xml version="1.0" encoding="utf-8"?>
<MeetingActivity xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" Version="1"
    TimeStamp="2008-08-27T13:33:18.4660052+02:00" Topic="Meeting"
Type="MeetingStarted">
  <Meeting Id="1">
    <MeetingData Subject="DefaultMeeting" DateTime="2008-08-27T13:22:33.097"
        Description="" />
    <Sessions>
      <SessionContainer Id="1">
        <SessionData Subject="Session" Description="Description" Done="false" />
      </SessionContainer>
    </Sessions>
    <Participants>
      <ParticipantContainer Id="2">
        <ParticipantData Present="false" VotingWeight="1" VotingAuthorisation="true"
            MicrophoneAuthorisation="true" FirstName="Carl" MiddleName="the"
            LastName="Coder" Title="Sir" Country="Spain" RemainingSpeechTime="-1" />
        <Seat Id="3">
          <SeatData Name="0003" MicrophoneActive="false" />
        </Seat>
        <Group Name="-" />
      </ParticipantContainer>
      <!-- ... The next participants are located here ... -->
    </Participants>
    <Channels>
      <Channel Number=”1” />
        <Language Abbreviation=”ENG” Name=”English” />
      </Channel>
      <Channel Number=”2” />
        <Language Abbreviation=”NLD” Name=”Dutch” />
      </Channel
    </Channels>
    <Booths>
      <Booth Number=”1” />
        <Desks>
          <Desk Number=”1” />
            <Seat Id=”25”>
              <SeatData Name=”1:1” MicrophoneActive=”false” />
            </Seat>
          </Desk>
        </Desks>
      </Booth>
    </Booths>
  </Meeting>
</MeetingActivity>
```

A.2.4. MeetingStopped

```
<?xml version="1.0" encoding="utf-8"?>
<MeetingActivity xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" Version="1"
    TimeStamp="2008-08-27T13:17:26.0182056+02:00" Topic="Meeting"
Type="MeetingStopped">
  <Meeting Id="1" />
</MeetingActivity>
```

A.2.5. VotingStopped

```
<?xml version="1.0" encoding="utf-8"?>
<VotingActivity xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" Version="1"
    TimeStamp="2008-08-27T13:50:36.0042872+02:00" Topic="Voting"
Type="VotingStopped">
  <Voting Id="0">
    <VotingResults>
      <IndividualResults>
        <VotingIndividualResultContainer AnswerId="1">
          <Participant Id="2">
            <ParticipantData Present="false" VotingWeight="1"
VotingAuthorisation="true"
                MicrophoneAuthorisation="true" FirstName="Carl" MiddleName="the"
                LastName="Coder" Title="Sir" Country="Spain" RemainingSpeechTime="-1"
/>
            <Seat Id="3">
              <SeatData Name="0003" MicrophoneActive="false" />
            </Seat>
            <Group Name="-" />
          </Participant>
        </VotingIndividualResultContainer>
      </IndividualResults>
      <VotingTotalResults Approved="true" RequiredQuorum="0" ActualQuorum="0"
          MaximumQuorum="1" RequiredMajority="0" ActualMajority="0"
MaximumMajority="1"
          NumberOfAuthorizedPresentParticipants="1"
          NumberOfAuthorizedPresentParticipantsWithoutVote="0">
        <VotingAnswerResults>
          <VotingAnswerResultContainer AnswerId=”1” NumberOfCasts="1"
Percentage="100"/>
          <VotingAnswerResultContainer AnswerId=”2” NumberOfCasts="0"
Percentage="0"/>
        </VotingAnswerResults>
      </VotingTotalResults>
      <VotingGroupResults>
        <VotingGroupResultContainer NumberOfAuthorizedPresentParticipants="0"
            NumberOfAuthorizedPresentParticipantsWithoutVote="0">
          <VotingAnswerResults>
            <VotingAnswerResultContainer AnswerId=”1” NumberOfCasts="1"
Percentage="100"/>
            <VotingAnswerResultContainer AnswerId=”2” NumberOfCasts="0"
Percentage="0"/>
          </VotingAnswerResults>
          <Group Name="-" />
        </VotingGroupResultContainer>
      </VotingGroupResults>
    </VotingResults>
  </Voting>
</VotingActivity>
```

A.2.6. MicrophoneTestStopped

```
<?xml version="1.0" encoding="utf-8"?>
<TestSystemActivity xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" Version="1" TimeStamp="2011-02-
14T10:38:29.1421121+01:00" Topic="TestSystem" Type="MicrophoneTestEnded">
<MicrophoneTestResults>
   <MicrophoneTestResultContainer Passed="true">
      <Seat>
         <SeatContainer Id="16">
            <SeatData Name="0002"/>
         </SeatContainer>
```

```
            <SeatContainer Id="17">
                <SeatData Name="1:1"/>
            </SeatContainer>
            <SeatContainer Id="18">
                <SeatData Name="1:2"/>
            </SeatContainer>
        </Seat>
    </MicrophoneTestResultContainer>
    <MicrophoneTestResultContainer Passed="false">
        <Seat>
            <SeatContainer Id="19">
                <SeatData Name="0003"/>
            </SeatContainer>
            <SeatContainer Id="20">
                <SeatData Name="2:1"/>
            </SeatContainer>
        </Seat>
    </MicrophoneTestResultContainer>
</MicrophoneTestResults>
</TestSystemActivity>
```