



► Polycom®  
RMX® 1500/2000/4000  
XML API Overview

### **Trademark Information**

Polycom®, the Polycom “Triangles” logo, and the names and marks associated with Polycom’s products are trademarks and/or service marks of Polycom, Inc., and are registered and/or common-law marks in the United States and various other countries. All other trademarks are the property of their respective owners.

### **Patent Information**

The accompanying product is protected by one or more U.S. and foreign patents and/or pending patent applications held by Polycom, Inc.

© 2011 Polycom, Inc. All rights reserved.

Polycom, Inc.  
4750 Willow Road  
Pleasanton, CA 94588-2708  
USA

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Polycom, Inc. Under the law, reproducing includes translating into another language or format.

As between the parties, Polycom, Inc., retains title to and ownership of all proprietary rights with respect to the software contained within its products. The software is protected by United States copyright laws and international treaty provision. Therefore, you must treat the software like any other copyrighted material (e.g., a book or sound recording).

Every effort has been made to ensure that the information in this manual is accurate. Polycom, Inc., is not responsible for printing or clerical errors. Information in this document is subject to change without notice.

# Table of Contents

<b>Introduction .....</b>	<b>1</b>
<b>Contents of the RMX XML SDK .....</b>	<b>1</b>
<b>Types of Schemas.....</b>	<b>2</b>
Common Schemas (Prefix common) .....	2
Object Schemas (Prefix obj) .....	2
Transaction Schemas (Prefix trans) .....	2
Response Schemas (Prefix response_trans).....	3
The response_general Schema .....	3
<b>Description of the Individual Schemas.....</b>	<b>4</b>
<b>Working with the RMX XML API .....</b>	<b>11</b>
Communicating with the RMX XML API .....	11
XML Requests and Responses .....	11
The RMX XML API Internal XML Parser.....	12
Logging in to the MCU .....	12
The trans_mcu Schema .....	12
An XML Example .....	24
Keeping the Connection with the MCU Alive.....	26
Working with the Response and Transaction Schemas.....	26
Audio Muting a Participant - An XML Example.....	29
Transaction Processing.....	30
Polling for Data .....	32
Retrieving Channel Information for IP Participants .....	35
The Compression Mechanism.....	35
Sending XML requests in Compressed Format .....	35
Receiving XML responses from the MCU in Compressed Format.....	36



# Introduction

RMX User workstations and other external systems are connected to the MCU via TCP/IP LAN connections using XML over HTTP. These external systems communicate with the MCU using the RMX XML API as a low level API.

The RMX XML API works as an internal web site on the MCU. The web site receives XML pages and sends the answer back in XML format. All requests and responses have schema formats. The syntax used for the XML API schemas is according to the W3C recommendation implemented in MS version 4 MSXML.

Copies of all the RMX XML API schemas are supplied in the **Schemas** folder in the RMX SDK. The XML API schemas are text files, and you can open them with any editor you want. Internet Explorer can be used to view the files in a convenient format.

This guide provides basic instructions for working with the RMX XML API. For more detailed information, refer to the RMX XML API Reference Guide. Contents of the RMX XML SDK

The RMX XML SDK consists of the following items:

- Copies of all of the RMX XML schemas (located in the **Schemas** folder).
- The Polycom XML Tracer application (located in the Tracer folder).
- Sample Scripts that can be used to communicate with an LDAP database (located in the External\_Database\_LDAP folder)
- Documentation (located in the Docs folder)

The **Docs** Folder contains the following documents:

- RMX XML API Release Notes V7.6
- RMX XML API Overview V7.6 (this document)
- Polycom RMX XML API V7.6 Reference Guide (.chm file)
- MGC to RMX 2000 XML API Conferencing Comparison
- RMX 2000 External Database API Programmer's Guide
- Polycom XML Tracer User Guide V1\_52
- Readme file

# Types of Schemas

The schemas are divided to four categories, each with a different prefix, as follows:

## Common Schemas (Prefix common)

The common schemas are like h files in C, and are included in other schemas. They contain definitions of elements that are common to multiple schemas. The common schemas are as follows:

- **common\_obj** - This contains elements that are used by multiple object schemas.
- **common\_obj\_ip\_span** - This contains elements that are used by multiple IP related schemas.
- **common\_trans** - This contains elements that are used by multiple transaction schemas.
- **common\_trans\_obj** - This contains elements that are used by multiple transaction and object schemas.

## Object Schemas (Prefix obj)

The object schemas describe the format and parameters of objects in the MCU, for example, conference or participant.

## Transaction Schemas (Prefix trans)

The transaction schemas are used to retrieve the data of objects in the MCU (get requests), or to act on objects in the MCU (set requests). For example, the schema **trans\_res\_1** defines the XML format to create a conference, Entry Queue, SIP Factory, Meeting Room or Profile on the MCU, to update the properties of an Entry Queue, SIP Factory, Meeting Room or Profile, to define the default Entry Queue, and to remove the definition of the default Entry Queue.

The transaction schemas often contain other types of schemas, such as object schemas and common schemas. For example, the **trans\_res\_1** schema includes the **obj\_reservation** schema, because it has to include the conference data when it is sent to start a conference.

The transaction schemas have a consistent pattern that demonstrates object oriented methodology, as follows:

1. The root element is the schema name.
2. The next element is the TRANS\_COMMON\_PARAMS element, which is similar to a base class. This element mainly describes the MCU on which the action is to operate, and the synchronization method to be used.
3. The next element is the ACTION element.
4. After this there are details of the individual actions that the transaction can make. For example, the actions in trans\_res\_1 are START (to create a conference, Entry Queue, SIP Factory, Meeting Room or Profile), UPDATE (to update the properties of an Entry Queue, SIP Factory, Meeting Room or Profile), SET\_DEFAULT\_EQ (to define the default Entry Queue) and CANCEL\_DEFAULT\_EQ (to remove the definition of the

default Entry Queue). For each action, there is a description of the parameters that must be specified in order to initiate the action.

## Response Schemas (Prefix **response\_trans**)

The response schemas describe the XML format returned by the XML API for the sent transactions.

The response schemas have a consistent pattern, as follows:

1. The root element is the schema name.
2. The next element is the return status for the transaction.
3. The next element is the ACTION element.
4. After this is a choice of the actions that this response answers, corresponding to the actions in the **trans\_xxx** schema. Under each action there is a description of the returned data.

In general, each **trans\_xxx** schema has a corresponding **response\_trans\_xxx** schema. However, although due to internal performance considerations the **trans\_res** and **trans\_conf** schemas were each split into two schemas (**trans\_res\_1** and **trans\_res\_2**, and **trans\_conf\_1** and **trans\_conf\_2**, respectively), there is only one response schema for each pair of transactions, (**response\_trans\_res** and **response\_trans\_conf**, respectively).

## The **response\_general** Schema

The **response\_general** schema is returned when there is no response with data available, for example, when an invalid schema is sent.

Usually the **response\_general** schema only contains a status which explains why the requested response was not sent. However, the schema optionally contains a **TRANS\_TOKEN** element, which is used for asynchronous transactions for which the full response is not returned automatically, and an **ACTION\_TYPE** element which identifies the type of request that was sent.

# Description of the Individual Schemas

The RMX 1500/2000/ 4000 XML API developer's kit version 7.6 contains the following 140 schemas:

Schema	Description
<b>common_obj</b>	Contains elements that are used by multiple object schemas.
<b>common_obj_ip_span</b>	Contains elements that are used by multiple IP related schemas.
<b>common_trans</b>	Contains elements that are used by multiple transaction schemas.
<b>common_trans_obj</b>	Contains elements that are used by multiple transaction and object schemas.
<b>obj_active_alarms_list</b>	Holds a list of system alerts.
<b>obj_active_directory</b>	Holds information about the Active Directory. (Applicable from versions 7.5.0.J and 7.6)
<b>obj_audit_file_summary_list</b>	Holds a list of auditor file summaries.
<b>obj_av_msg_service</b>	Holds information about Conference and Entry Queue IVR Services.
<b>obj_cards_list</b>	Holds information about cards.
<b>obj_cdr_full</b>	Holds complete information about a CDR.
<b>obj_cdr_summary_list</b>	Holds summary information about a CDR.
<b>obj_certificate_summary</b>	Holds the information of a security certificate used by the RMX. (Applicable from versions 7.5.0.J and 7.6)
<b>obj_certificate_summary_list</b>	Holds a list of the information of a security certificate used by the RMX. (Applicable from versions 7.5.0.J and 7.6)
<b>obj_cfg</b>	Holds a list of system configuration flag names and values.
<b>obj_collect_info</b>	Contains elements used for the Information Collector. (Applicable from version 7.2)
<b>obj_conf_summary_list</b>	Holds summary information about conferences.
<b>obj_conference</b>	Holds detailed information about a conference.
<b>obj_connection</b>	Holds connection information for a user who is connected to the MCU.
<b>obj_connections_list</b>	Holds connection information for all users who are connected to the MCU.
<b>obj_directory</b>	Holds the contents of a directory, that is, a list of the names of the files and sub-directories in the directory.
<b>obj_dongle_configuration</b>	Mostly for internal use only, but also holds the number of licensed ports.
<b>obj_dynamic_ip_service</b>	Holds the dynamically changing parameters of an IP Service. The dynamically changing parameters are the parameters that are displayed by the Signaling Monitor.



Schema	Description
<b>obj_ethernet_settings</b>	Holds information about a LAN port of the RMX 4000 and the RMX 1500/2000 when an RTM LAN card is installed. (Applicable from version 5.0)
<b>obj_ethernet_settings_list</b>	Holds information about all LAN ports of the RMX 4000 and the RMX 1500/2000 when an RTM LAN card is installed. (Applicable from version 5.0)
<b>obj_exchange_cfg</b>	Holds information about the connection to the Microsoft Exchange Server. (Applicable from version 6.0)
<b>obj_faults_list</b>	Holds a list of faults.
<b>obj_force</b>	Holds information about the arrangement of video windows on the endpoint screens, and information about which participants are forced to appear in specific windows.
<b>obj_full_ip_service</b>	Holds full information about an IP Network Service, that is, both the general parameters and the dynamically changing parameters.
<b>obj_full_ip_service_list</b>	Holds full information about all IP Network Services, that is, both the general parameters and the dynamically changing parameters.
<b>obj_hot_backup</b>	Contains the Hot Backup configuration parameters. (Applicable in version 4.6/4.7.x and from version 7.2)
<b>obj_install_phase</b>	Contains the parameters of a single software installation phase. (Applicable from version 6.0)
<b>obj_install_phases_list</b>	Contains a list of phases that are part of the software installation procedure. (Applicable from version 6.0)
<b>obj_ip_service</b>	Holds information about general parameters of an IP or Management Network Service.
<b>obj_ip_service_list</b>	Holds information about all IP or Management Network Services.
<b>obj_isdn_srv</b>	Holds information about an ISDN/PSTN Network Service.
<b>obj_isdn_srv_list</b>	Holds information about all ISDN/PSTN Network Services.
<b>obj_lecture_mode</b>	Holds Lecture Mode information.
<b>obj_licensing_configuration</b>	Holds license information.
<b>obj_logger</b>	Contains information about the Logger Configuration function (Applicable from version 7.6)
<b>obj_log_file_summary_list</b>	Holds a list of log file summaries.
<b>obj_mcu_time</b>	Holds the MCU time.
<b>obj_ongoing_party</b>	Holds information about a participant in a conference.
<b>obj_oper_list</b>	Holds information about all MCU users.
<b>obj_operator</b>	Holds information about an MCU user.
<b>obj_party</b>	Holds information about a participant.

Schema	Description
<b>obj_recording_links_list</b>	Holds a list of recording links.
<b>obj_repeated</b>	Contains a list of recurrent reservations values. (Applicable from version 4.0)
<b>obj_res_summary_list</b>	Holds summary information about Meeting Rooms, Entry Queues, SIP Factories and Profiles.
<b>obj_reservation</b>	Holds detailed information about a conference, Meeting Room, Entry Queue, SIP Factory or Profile.
<b>obj_resolutions_set</b>	Contains the decision array of video resolutions that are used for video conferences. (Applicable from version 7.0)
<b>obj_rsrc_report</b>	Holds a resource report for an RMX.
<b>obj_rtm_isdn_span</b>	Holds information about an ISDN/PSTN span.
<b>obj_rtm_isdn_span_list</b>	Holds information about all ISDN/PSTN spans.
<b>obj_service</b>	Holds a subset of the information about an ISDN/PSTN Network Service.
<b>obj_snmp</b>	Holds SNMP (Simple Network Management Protocol) details.
<b>obj_tcp_dump</b>	Contains elements used for the TCP dump Network Traffic Capture interface. (Applicable from version 7.2)
<b>response_general</b>	Returned when there is no response with data available, for example, when an invalid schema is sent. Usually the response_general schema only contains a status which explains why the requested response was not sent. However, the schema optionally contains a TRANS_TOKEN element, which is used for asynchronous transactions for which the full response is not returned automatically.
<b>response_trans_active_alarms_list</b>	Contains the response to the trans_active_alarms_list schema, which is used to retrieve system alerts.
<b>response_trans_audit_file_summary_list</b>	Contains the response to the trans_audit_file_summary_list schema, which is used to retrieve a list of auditor file summaries.
<b>response_trans_av_msg_service</b>	Contains the response to the trans_av_msg_service schema, which is used to manage a Conference or Entry Queue IVR Service.
<b>response_trans_av_msg_service_list</b>	Contains the response to the trans_av_msg_service_list schema, which is used to retrieve information about Conference and Entry Queue IVR Services.
<b>response_trans_card</b>	Contains the response to the trans_card schema, which is used to manage a card.
<b>response_trans_cards_list</b>	Contains the response to the trans_cards_list schema, which is used to retrieve details of all the cards in the MCU.
<b>response_trans_cdr_full</b>	Contains the response to the trans_cdr_full schema, which is used to retrieve the complete CDR information from the MCU.
<b>response_trans_cdr_list</b>	Contains the response to the trans_cdr_list schema, which is used to retrieve a list of CDR summaries from the MCU.
<b>response_trans_certificate</b>	Contains the response to the trans_certificate schema, which is used to send certificates to the RMX.

Schema	Description
<b>response_trans_certificate_list</b>	Contains the response to the trans_certificate_list schema, which is used to retrieve a list of certificates from the MCU. (Applicable from version 7.5.0.J and 7.6)
<b>response_trans_certificate_request</b>	Contains the response to the trans_certificate_request schema, which is used to create certificate requests.
<b>response_trans_cfg</b>	Contains the response to the trans_cfg schema, which is used to manage system configuration flags.
<b>response_trans_conf</b>	Contains the response to the trans_conf_1 or trans_conf_2 schemas, which are used to perform operations on conferences.
<b>response_trans_conf_list</b>	Contains the response to the trans_conf_list schema, which is used to retrieve a list of conference summaries.
<b>response_trans_connections_list</b>	Contains the response to the trans_connections_list schema, which is used to retrieve connection information about all users who are currently connected to the MCU.
<b>response_trans_ethernet_settings</b>	Contains the response to the trans_ethernet_settings schema, which is used to manage the LAN ports of the RMX 4000 and the RMX 1500/2000 when an RTM LAN card is installed. (Applicable from version 5.0)
<b>response_trans_ethernet_settings_list</b>	Contains the response to the trans_ethernet_settings_list schema, which is used to retrieve details of the LAN ports of the RMX 4000 and the RMX 1500/2000 when an RTM LAN card is installed. (Applicable from version 5.0)
<b>response_trans_faults_list</b>	Contains the response to the trans_faults_list schema, which is used to retrieve the full faults list.
<b>response_trans_faults_list_short</b>	Contains the response to the trans_faults_list_short schema, which is used to retrieve an abbreviated faults list, that is, a faults list without the asserts.
<b>response_trans_hot_backup</b>	Contains the response to the trans_hot_backup schema, which is used to retrieve and update the hot backup configuration. (Applicable in version 4.6/4.7.x and from version 7.2)
<b>response_trans_ip_service</b>	Contains the response to the trans_ip_service schema, which is used to manage an IP and Management Network Service.
<b>response_trans_ip_service_list</b>	Contains the response to the trans_ip_service_list schema, which is used to retrieve details of IP and Management Network Services.
<b>response_trans_isdn_phone</b>	Contains the response to the trans_isdn_phone schema, which is used to manage ISDN/PSTN phone number ranges.
<b>response_trans_isdn_service</b>	Contains the response to the trans_isdn_service schema, which is used to manage ISDN/PSTN Network Services.
<b>response_trans_isdn_service_list</b>	Contains the response to the trans_isdn_service_list schema, which is used to retrieve details of all ISDN/PSTN Network Services.
<b>response_trans_log_file_list</b>	Contains the response to the trans_log_file_list schema which is used to retrieve a list of log file summaries.
<b>response_trans_logger</b>	Contains the response to the trans_logger schema, which is used to manage the Logger Configuration utility. (Applicable from version 7.6)

Schema	Description
<b>response_trans_mcu</b>	Contains the response to the trans_mcu schema, which is used to log in and out of the MCU, to retrieve information from the MCU, and to perform configuration and file operations on the MCU.
<b>response_trans_oper_list</b>	Contains the response to the trans_oper_list schema, which is used to retrieve information about all defined RMX users.
<b>response_trans_operator</b>	Contains the response to the trans_operator schema, which is used to manage an RMX user.
<b>response_trans_party</b>	Contains the response to the trans_party schema, which is used to retrieve information about a participant in a conference.
<b>response_trans_recording_links_list</b>	Contains the response to the trans_recording_links_list schema, which is used to manage recording links.
<b>response_trans_res</b>	Contains the response to the trans_res_1 or trans_res_2 schemas which are used to set up conferences, and to set up and manage Meeting Rooms, Entry Queues, SIP Factories and Profiles.
<b>response_trans_res_list</b>	Contains the response to the trans_res_list schema which is used to retrieve a list of Meeting Room, Entry Queue, SIP Factory and Profile summaries.
<b>response_trans_rsrc_report</b>	Contains the response to the trans_rsrc_report schema, which is used to retrieve a resource report for an RMX.
<b>response_trans_rtm_isdn_span</b>	Contains the response to the trans_rtm_isdn_span schema which is used to manage an ISDN/PSTN span.
<b>response_trans_rtm_isdn_span_list</b>	Contains the response to the trans_rtm_isdn_span_list which is used to retrieve details of ISDN/PSTN spans.
<b>response_trans_snmp</b>	Contains the response to the trans_snmp schema, which is used to manage the SNMP (Simple Network Management Protocol) configuration.
<b>response_trans_tcp_dump</b>	Contains the response to the trans_tcp_dump schema, which is used to for the TCP dump Network Traffic Capture interface. (Applicable from version 7.2)
<b>trans_active_alarms_list</b>	Used to retrieve system alerts.
<b>trans_audit_file_summary_list</b>	Used to retrieve a list of auditor file summaries.
<b>trans_av_msg_service</b>	Used to manage a Conference and Entry Queue IVR Service.
<b>trans_av_msg_service_list</b>	Used to retrieve information about Conference and Entry Queue IVR Services.
<b>trans_card</b>	Used to manage a card.
<b>trans_cards_list</b>	Used to retrieve details of all the cards in the MCU.
<b>trans_cdr_full</b>	Used to retrieve the complete CDR information from the MCU.
<b>trans_cdr_list</b>	Used to retrieve a list of CDR summaries from the MCU.
<b>trans_certificate</b>	Used to send certificates to the RMX.
<b>trans_certificate_list</b>	Used for Server Certificate monitoring. (Applicable in version 7.5.0.J and from version 7.6)
<b>trans_certificate_request</b>	Used to create certificate requests.
<b>trans_cfg</b>	Used to manage system configuration flags.

Schema	Description
<b>trans_conf_1</b>	The trans_conf_1 and trans_conf_2 schemas are used to manage conferences.
<b>trans_conf_2</b>	The trans_conf_1 and trans_conf_2 schemas are used to manage conferences.
<b>trans_conf_list</b>	Used to retrieve a list of conference summaries.
<b>trans_connections_list</b>	Used to retrieve connection information about all RMX users who are currently connected to the MCU.
<b>trans_ethernet_settings</b>	Used to manage the LAN ports of the RMX 4000 and the RMX 1500/2000 when an RTM LAN card is installed. (Applicable from version 5.0)
<b>trans_ethernet_settings_list</b>	Used to retrieve details of all LAN ports of the RMX 4000 and the RMX 1500/2000 when an RTM LAN card is installed. (Applicable from version 5.0)
<b>trans_faults_list</b>	Used to retrieve a full faults list.
<b>trans_faults_list_short</b>	Used to retrieve an abbreviated faults list, that is, a faults list without the asserts.
<b>trans_hot_backup</b>	Used to retrieve the hot backup configuration information and enables you to update the hot backup configuration. (Applicable from version 7.2)
<b>trans_ip_service</b>	Used to manage an IP and Management Network Service.
<b>trans_ip_service_list</b>	Used to retrieve details of IP and Management Network Services.
<b>trans_isdn_phone</b>	Used to manage ISDN/PSTN Network Service phone number ranges.
<b>trans_isdn_service</b>	Used to manage an ISDN/PSTN Network Service.
<b>trans_isdn_service_list</b>	Used to retrieve details of ISDN/PSTN Network Services.
<b>trans_log_file_list</b>	Used to retrieve a list of log file summaries.
<b>trans_logger</b>	Used to manage the Logger Configuration utility. (Applicable from version 7.6)
<b>trans_mcu</b>	Used to log in and out of the MCU, to retrieve information from the MCU, and to perform configuration and file operations on the MCU.
<b>trans_oper_list</b>	Used to retrieve information about all defined RMX users.
<b>trans_operator</b>	Used to manage an RMX user.
<b>trans_party</b>	Used to retrieve information about a participant in a conference.
<b>trans_recording_links_list</b>	Used to manage recording links.
<b>trans_res_1</b>	The trans_res_1 and trans_res_2 schemas are used to set up conferences, and to set up and manage Meeting Rooms, Entry Queues, SIP Factories and Profiles.
<b>trans_res_2</b>	The trans_res_1 and trans_res_2 schemas are used to set up conferences, and to set up and manage Meeting Rooms, Entry Queues, SIP Factories and Profiles.
<b>trans_res_list</b>	Used to retrieve a list of Meeting Room, Entry Queue, SIP Factory and Profile summaries.

Schema	Description
<b>trans_rsrc_report</b>	Used to retrieve a resource report for an RMX.
<b>trans_rtm_isdn_span</b>	Used to manage an ISDN/PSTN span.
<b>trans_rtm_isdn_span_list</b>	Used to retrieve details of ISDN/PSTN spans.
<b>trans_snmp</b>	Used to manage the SNMP (Simple Network Management Protocol) configuration.
<b>trans_tcp_dump</b>	Contains the requests for the TCP dump Network Traffic Capture interface. (Applicable from version 7.2)

**Note:** In addition to the schemas listed above, the RMX XML API developer's kit contains the following schemas that are for internal use only, but are included in the XML API developer's kit because they are referenced by other schemas:

- obj\_conf\_full\_list
- obj\_gateway
- obj\_mcu\_lan\_configuration
- obj\_mcu\_memory\_state
- obj\_performance\_monitoring
- obj\_repeated
- obj\_span

These schemas are located in the **Internal** folder.

# Working with the RMX XML API

## Communicating with the RMX XML API

The RMX XML API can be used with any environment that can post HTTP streams. You need to open an HTTP connection against each MCU with which you want to work.

The client application sends an XML string to the internal HTTP server in the MCU.

A module in the internal web site performs validation checks on the XML string.

If the XML string fails the validation, the MCU sends an error status to the client application via the internal web site.

If the XML string is OK, it is translated to C++ and sent to the MCU. The MCU then sends a response to the client application.

## XML Requests and Responses

All XML API requests have an input parameter which contains an XML string. The XML string can be specified in various formats, for example, char\*, BSTR or MSDOM. This input parameter will be referred to as the REQUEST\_XML.

All XML API responses return an output parameter which contains an XML string in the same format as the input parameter. The output parameter will be referred as the RESPONSE\_XML.

### Example of the procedure for working with the RMX XML API from Internet Explorer using Java Script

1. Create an HTTP connection object, as follows:

```
var httpConnection = new ActiveXObject("Microsoft.XMLHTTP");
```

2. Open an HTTP connection, pointing it to the URL of the XTransactionSubmit.asp which is located in the MCU, as follows:

```
httpConnection.Open("POST", "MCU IP Address/XTransactionSubmit.  
asp", false);
```

3. Post the XML to the MCU, as follows:

```
httpConnection.Send(REQUEST_XML);
```

4. **httpConnection.responseText** will hold the XML\_RESPONSE string.

## The RMX XML API Internal XML Parser

The RMX XML API uses an internal XML parser. Like the Microsoft XML parser, this internal parser uses certain characters as control characters. For example, if you include the greater than (>) character in an XML element, the internal parser will interpret this character as an end of element symbol, and an error will occur. To avoid this problem you need to replace these special characters in the XML stream with the appropriate entity reference, as detailed in the table below:

Special Character	Entity Reference
&	&amp;
<	&lt;
>	&gt;
'	&apos;
"	&quot;

When the RMX XML API internal parser encounters an entity reference in an XML stream, it converts the entity reference to the appropriate special character. Similarly, when the internal parser returns an XML stream, the stream will contain entity references instead of the equivalent special characters. You can then convert these entity references back to the appropriate special character.

## Logging in to the MCU

You need to log in to the MCU for each socket you connect through, and cannot connect through a new socket using the old token obtained when logging in through a different socket. To connect through a different socket, you must log in again using XML.

The RMX XML API can currently only be used with port 80. The preferred port is now hard-coded, and cannot be changed in system.cfg.

## The trans\_mcu Schema

The first thing you have to do to before sending any transactions to the MCU is to log in to the MCU. This is done according to the **trans\_mcu** schema shown below:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
```

```
<xsd:include schemaLocation="common_trans.xsd"/>
<xsd:include schemaLocation="obj_mcu_lan_configuration.xsd"/>
<xsd:include schemaLocation="obj_mcu_time.xsd"/>
<xsd:include schemaLocation="obj_exchange_cfg.xsd"/>
<xsd:include schemaLocation="obj_resolutions_set.xsd"/>
<xsd:include schemaLocation="obj_collect_info.xsd"/>
<xsd:include schemaLocation="obj_active_directory.xsd"/>
```



```
<!-- In most transaction schemas, details of the choices are to be found at the
beginning of the schema. These are the ACTIONS (functions) that can be performed. -
->

<xsd:element name="LOGIN" type="LoginRequestContent"/>

<!-- The LoginRequestContent type is defined in common_trans.xsd and is explained
later.-->

<xsd:element name="LOGOUT" type="LogoutContent"/>
<xsd:element name="GET_STATE"/>
<xsd:element name="RMX_GET_STATE_EX" type="RmxGetStateExContent"/>
<xsd:element name="GET_MEMORY_STATE"/>
<xsd:element name="GET_LAN_CONFIGURATION"/>
<xsd:element name="GET_TIME"/>
<xsd:element name="GET_CFS"/>
<xsd:element name="RESET"/>
<xsd:element name="GET_RECORDING_JUNCTION_LIST"/>
<xsd:element name="FILE_UPDATED_STRING" type="xsd:string"/>
<xsd:element name="PATH" type="xsd:string"/>
<xsd:element name="NEW_NAME" type="xsd:string"/>
<xsd:element name="GET_DIRECTORY" type="DirectoryPath"/>
<xsd:element name="GET_DIRECTORY_RECURSIVE" type="DirectoryPath"/>
<xsd:element name="GET_VIRTUAL_DIRECTORY" type="DirectoryPath"/>
<xsd:element name="GET_VIRTUAL_DIRECTORY_RECURSIVE" type="DirectoryPath"/>
<xsd:element name="REMOVE_DIRECTORY" type="DirectoryPath"/>
<xsd:element name="REMOVE_DIRECTORY_CONTENT" type="DirectoryPath"/>
<xsd:element name="BEGIN_RECEIVING_VERSION"/>
<xsd:element name="FINISHED_TRANSFER_VERSION"/>
<xsd:element name="UPDATE_KEY_CODE" type="KeyCodeContent"/>
<xsd:element name="FLUSH"/>
<xsd:element name="STOP_ALL_MEDIA_RECORDING"/>
<xsd:element name="RESTORE_TYPE" type="RestoreType"/>
<xsd:element name="COLLECT_INFO" type="CollectInfoContent"/>
<xsd:element name="GET_COLLECT_INFO_SETTINGS"/>
<xsd:element name="ABORT_COLLECT_INFO"/>
<xsd:element name="GET_INSTALLATION_STATUS"/>
<xsd:element name="TURN_SSH" type="TurnSSHContent"/>
<xsd:element name="GET_PORT_CONFIGURATION"/>
<xsd:element name="SET_PORT_CONFIGURATION" type="PortConfigurationIndex"/>
<xsd:element name="GET_ENHANCED_PORT_CONFIGURATION"/>
<xsd:element name="GET_CHECK_ENHANCED_PORT_CONFIGURATION"
type="EnhancedPortNumConfiguration"/>
<xsd:element name="SET_ENHANCED_PORT_CONFIGURATION"
type="EnhancedPortNumConfiguration"/>
<xsd:element name="AUDIO_NUM_PORTS_CONFIG" type="xsd:integer"/>
<xsd:element name="CIF_NUM_PORTS_CONFIG" type="xsd:integer"/>
<xsd:element name="SD_NUM_PORTS_CONFIG" type="xsd:integer"/>
```

```

<xsd:element name="HD720_NUM_PORTS_CONFIG" type="xsd:integer"/>
<xsd:element name="HD1080_NUM_PORTS_CONFIG" type="xsd:integer"/>
<xsd:element name="INSTALL_PREVIOUS_VERSION"
type="InstallPreviousVersionContent"/>
<xsd:element name="VERSION_TYPE" type="VersionType"/>
<xsd:element name="GET_ALLOCATION_MODE"/>
<xsd:element name="SET_ALLOCATION_MODE" type="SelectedAllocationModeContent"/>
<xsd:element name="SELECTED_ALLOCATION_MODE" type="AllocationModeType"/>
<xsd:element name="BACKUP_CONFIG_START"/>
<xsd:element name="BACKUP_CONFIG_FINISH"/>
<xsd:element name="RESTORE_CONFIG_START"/>
<xsd:element name="RESTORE_CONFIG_FINISH" type="RestoreCfgFinish"/>
<xsd:element name="REASON" type="LogoutReasonType" default="normal"/>
<xsd:element name="SET_PING" type="SetPingContent"/>
<xsd:element name="GET_PING" type="GetPingContent"/>
<xsd:element name="GET_MCU_EXCHANGE_CONFIG_PARAMS"/>
<xsd:element name="SET_MCU_EXCHANGE_CONFIG_PARAMS"
type="SetMcuExchangeConfigParams"/>
<xsd:element name="GET_LAST_SET_MCU_EXCHANGE_CONFIG_INDICATION"/>
<xsd:element name="GET_RESOLUTIONS_SET"/>
<xsd:element name="GET_ACTIVE_DIRECTORY_CONFIGURATION"/>
<xsd:element name="SET_ACTIVE_DIRECTORY_CONFIGURATION"
type="SetActiveDirectoryConfiguration"/>
<xsd:element name="GET_AD_SERVER_AVAILABILITY_STATUS"/>
<xsd:element name="GET_LAST_UPDATE_KEY_CODE_INDICATION"/

<xsd:complexType name="RestoreCfgFinish">
    <xsd:sequence>
        <xsd:element name="RESTORE_CONFIG_FILE" type="xsd:string"/>
        <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
namespace="##other"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="VersionType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="fallback"/>
        <xsd:enumeration value="factory"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="InstallPreviousVersionContent">
    <xsd:sequence>
        <xsd:element ref="VERSION_TYPE"/>

```

```
<xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
namespace="###other"/>
</xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="RestoreType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="standard"/>
    <xsd:enumeration value="extensive"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="KeyCodeContent">
  <xsd:sequence>
    <xsd:element name="KEY_CODE" type="xsd:string"/>
    <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
namespace="###other"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CollectInfoContent">
  <xsd:sequence>
    <xsd:element ref="START_TIME"/>
    <xsd:element ref="END_TIME"/>
    <xsd:element ref="COLLECT_INFO_LIST"/>
    <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
namespace="###other"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="DirectoryPath">
  <xsd:sequence>
    <xsd:element ref="PATH"/>
    <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
namespace="###other"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TurnSSHContent">
  <xsd:sequence>
    <xsd:element ref="ON"/>
    <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
namespace="###other"/>
  </xsd:sequence>
</xsd:complexType>
```

```

<!-- The root elements in the transaction schemas have the same name as the schema
name. -->
<xsd:element name="TRANS_MCU">
<xsd:complexType>
    <xsd:sequence>
        <!-- The first element in all transaction schemas is the TRANS_COMMON_PARAMS
        element. This complex element is defined in common_trans.xsd and is explained
        later.-->
        <xsd:element ref="TRANS_COMMON_PARAMS"/>
        <!-- In all transaction schemas a choice of the ACTIONS group or the ACTION
        element comes next. The ACTION element contains the ACTIONS group which
        identifies which operation you want to perform. In this case the choices are
        LOGIN, LOGOUT, GET_STATE and so on, as defined further down in the schema. -->
        <xsd:choice>
            <xsd:group ref="ACTIONS"/>
            <xsd:element ref="ACTION"/>
        </xsd:choice>
        <xsd:any namespace="##other" processContents="skip" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="SET_LAN_CONFIGURATION">
<xsd:complexType>
    <xsd:sequence>
        <xsd:element ref="MCU_LAN_CONFIGURATION"/>
        <xsd:any namespace="##other" processContents="skip" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="SET_TIME">
<xsd:complexType>
    <xsd:sequence>
        <xsd:element ref="MCU_TIME"/>
        <xsd:any namespace="##other" processContents="skip" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>

```

```
<xsd:element name="SHIFT_RESERVATIONS_TIME">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="SHIFT_RESERVATIONS"/>
      <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
        namespace="##other"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="FILE_UPDATED">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="FILE_UPDATED_STRING"/>
      <xsd:any namespace="##other" processContents="skip" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="CREATE_DIRECTORY">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="PATH"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="RENAME">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="PATH"/>
      <xsd:element ref="NEW_NAME"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```

<xsd:element name="SET_RESTORE_TYPE">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="RESTORE_TYPE"/>
      <xsd:any namespace="##other" processContents="skip" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="PortConfigurationIndex">
  <xsd:sequence>
    <xsd:element ref="SELECTED_ID"/>
    <xsd:any namespace="##other" processContents="skip" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="EnhancedPortNumConfiguration">
  <xsd:sequence>
    <xsd:element ref="AUDIO_NUM_PORTS_CONFIG"/>
    <xsd:element ref="CIF_NUM_PORTS_CONFIG"/>
    <xsd:element ref="SD_NUM_PORTS_CONFIG"/>
    <xsd:element ref="HD720_NUM_PORTS_CONFIG"/>
    <xsd:element ref="HD1080_NUM_PORTS_CONFIG"/>
    <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
      namespace="##other"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SelectedAllocationModeContent">
  <xsd:sequence>
    <xsd:element ref="SELECTED_ALLOCATION_MODE"/>
    <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
      namespace="##other"/>
  </xsd:sequence>
</xsd:complexType>

```

```
<xsd:simpleType name="LogoutReasonType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="normal"/>
    <xsd:enumeration value="session_expired"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="LogoutContent">
  <xsd:sequence>
    <xsd:element ref="REASON"/>
    <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
      namespace="###other"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SetPingContent">
  <xsd:sequence>
    <xsd:element ref="PING"/>
    <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
      namespace="###other"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="GetPingContent">
  <xsd:sequence>
    <xsd:element ref="PING_ID"/>
    <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
      namespace="###other"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SetMcuExchangeConfigParams">
  <xsd:sequence>
    <xsd:element ref="MCU_EXCHANGE_CONFIG_PARAMS"/>
    <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
      namespace="###other"/>
  </xsd:sequence>
</xsd:complexType>
```

```

<xsd:complexType name="RmxGetStateExContent">
  <xsd:sequence>
    <xsd:element ref="CLIENT_IP" minOccurs="0"/>
    <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
      namespace="##other"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="SET_RESOLUTIONS_SET">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="SET_RESOLUTIONS_PARAMS"/>
      <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
        namespace="##other"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="SetActiveDirectoryConfiguration">
  <xsd:sequence>
    <xsd:element ref="ACTIVE_DIRECTORY_CONFIG_PARAMS"/>
    <xsd:any processContents="skip" minOccurs="0" maxOccurs="unbounded"
      namespace="##other"/>
  </xsd:sequence>
</xsd:complexType>

<!-- The list of ACTIONS to choose from. -->
<xsd:group name="ACTIONS">
  <xsd:choice>
    <xsd:element ref="LOGIN"/>
    <xsd:element ref="LOGOUT"/>
    <xsd:element ref="GET_STATE"/>
    <xsd:element ref="GET_MEMORY_STATE"/> <!--not supported-->
    <xsd:element ref="RESET"/>
    <xsd:element ref="GET_LAN_CONFIGURATION"/> <!--not supported-->
    <xsd:element ref="SET_LAN_CONFIGURATION"/> <!--not supported-->
    <xsd:element ref="GET_TIME"/>
    <xsd:element ref="SET_TIME"/>
    <xsd:element ref="GET_DIRECTORY"/>
    <xsd:element ref="GET_DIRECTORY_RECURSIVE"/> <!--not supported-->
    <xsd:element ref="GET_VIRTUAL_DIRECTORY"/>
    <xsd:element ref="GET_VIRTUAL_DIRECTORY_RECURSIVE"/>
    <xsd:element ref="FILE_UPDATED"/> <!--not supported-->
    <xsd:element ref="CREATE_DIRECTORY"/>
  </xsd:choice>
</xsd:group>

```



```
<xsd:element ref="RENAME"/>
<xsd:element ref="REMOVE_DIRECTORY"/>
<xsd:element ref="BEGIN_RECEIVING_VERSION"/>
<xsd:element ref="FINISHED_TRANSFER_VERSION"/>
<xsd:element ref="UPDATE_KEY_CODE"/>
<xsd:element ref="FLUSH"/>
<xsd:element ref="GET_CFS"/>
<xsd:element ref="STOP_ALL_MEDIA_RECORDING"/>
<xsd:element ref="GET_RECORDING_JUNCTION_LIST"/>
<xsd:element ref="SET_RESTORE_TYPE"/>
<xsd:element ref="COLLECT_INFO"/>
<xsd:element ref="GET_COLLECT_INFO_SETTINGS"/>
<xsd:element ref="ABORT_COLLECT_INFO"/>
<xsd:element ref="GET_INSTALLATION_STATUS"/>
<xsd:element ref="TURN_SSH"/>
<xsd:element ref="REMOVE_DIRECTORY_CONTENT"/>
<xsd:element ref="GET_PORT_CONFIGURATION"/>
<xsd:element ref="SET_PORT_CONFIGURATION"/>
<xsd:element ref="GET_ENHANCED_PORT_CONFIGURATION"/>
<xsd:element ref="GET_CHECK_ENHANCED_PORT_CONFIGURATION"/>
<xsd:element ref="SET_ENHANCED_PORT_CONFIGURATION"/>
<xsd:element ref="INSTALL_PREVIOUS_VERSION"/>
<xsd:element ref="GET_ALLOCATION_MODE"/>
<xsd:element ref="SET_ALLOCATION_MODE"/>
<xsd:element ref="BACKUP_CONFIG_START"/>
<xsd:element ref="BACKUP_CONFIG_FINISH"/>
<xsd:element ref="RESTORE_CONFIG_START"/>
<xsd:element ref="RESTORE_CONFIG_FINISH"/>
<xsd:element ref="SET_PING"/>
<xsd:element ref="GET_PING"/>
<xsd:element ref="RMX_GET_STATE_EX"/>
<xsd:element ref="GET_MCU_EXCHANGE_CONFIG_PARAMS"/>
<xsd:element ref="SET_MCU_EXCHANGE_CONFIG_PARAMS"/>
<xsd:element ref="GET_LAST_SET_MCU_EXCHANGE_CONFIG_INDICATION"/>
<xsd:element ref="GET_RESOLUTIONS_SET"/>
<xsd:element ref="SET_RESOLUTIONS_SET"/>
<xsd:element ref="GET_ACTIVE_DIRECTORY_CONFIGURATION"/>
<xsd:element ref="SET_ACTIVE_DIRECTORY_CONFIGURATION"/>
```

```

        <xsd:element ref="GET_AD_SERVER_AVAILABILITY_STATUS"/>
        <xsd:element ref="GET_LAST_UPDATE_KEY_CODE_INDICATION"/>
        <xsd:element ref="SHIFT_RESERVATIONS_TIME"/>
    </xsd:choice>
</xsd:group>

```

```

<xsd:element name="ACTION">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:group ref="ACTIONS"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

The LoginRequestContent type and the MCU\_IP and LISTEN\_PORT elements are defined in the common\_trans schema.

```

<xsd:complexType name="LoginRequestContent">
  <xsd:sequence>
    <!-- The MCU IP & port number see below. -->
    <xsd:element ref="MCU_IP"/>
    <!-- The user name and password to log in to the MCU. -->
    <xsd:element ref="USER_NAME"/>
    <xsd:element ref="PASSWORD"/>
    <xsd:element ref="STATION_NAME" minOccurs="0"/>
    <xsd:element ref="COMPRESSION" minOccurs="0"/>
    <xsd:element ref="CONFERENCE_RECORDER" minOccurs="0"/> <!--not supported-->
    <xsd:element ref="NEW_PASSWORD" minOccurs="0"/> <!--not supported-->
    <xsd:any namespace="##other" processContents="skip" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="MCU_IP" type="McuContent"/>

<xsd:complexType name="McuContent">
  <xsd:sequence>
    <xsd:element ref="IP"/> <!-- The IP of the MCU. -->
    <xsd:element ref="LISTEN_PORT" minOccurs="0"/> <!--relevant only
      when connecting-->
    <xsd:element ref="HOST_NAME" minOccurs="0"/>
    <xsd:any namespace="##other" processContents="skip" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

```

```

<xsd:element name="LISTEN_PORT" type="xsd:integer"/>

<!-- The TRANS_COMMON_PARAMS element.
As stated earlier, each trans_xxx schema starts with the TRANS_COMMON_PARAMS
element, which is defined in the common_trans schema.
Below is the definition of TRANS_COMMON_PARAMS. -->

<xsd:element name="TRANS_COMMON_PARAMS" type="TransCommonParamsContent"/>
.
<xsd:complexType name="TransCommonParamsContent">
    <xsd:sequence>
        <!-- After logging in, you will get two tokens that you will have to send in
each transaction. However, in the login transaction itself, these tokens
have no meaning. -->

        <!-- The MCU_TOKEN is the identifier of the MCU. -->
        <xsd:element ref="MCU_TOKEN"/>

        <!-- The MCU_USER_TOKEN identifies the logger to the MCU for security
purposes. -->
        <xsd:element ref="MCU_USER_TOKEN"/>

        <!-- The synchronization method. -->
        <xsd:group ref="SYNC_CHOICE" minOccurs="0"/>
        <xsd:element ref="MESSAGE_ID" minOccurs="0"/>

        <xsd:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<!-- The SYNC_CHOICE element below is optional. However, if your client works
asynchronously, and you want to use tokens or pointers to help identify
transaction responses, then you should specify the ASYNC value for the SYNC_CHOICE
element. Then you can use the YOUR_TOKEN1 or YOUR_TOKEN2 elements that are
contained in the ASYNC element to save a token, or pointer, or any 32 bit data. --
>

<xsd:group name="SYNC_CHOICE">
    <xsd:choice>
        <xsd:element ref="ASYNC"/>
        <xsd:element ref="SYNC"/>
    </xsd:choice>
</xsd:group>

```

```
<!-- Below is the full definition of the ASYNC element. -->
<xsd:element name="ASYNC" type="AsyncContent"/>
<xsd:complexType name="AsyncContent">
  <xsd:sequence>
    <!-- The first three elements, ASYNC_METHOD, URL and PORT_NUMBER are
    obsolete. -->
    <xsd:element ref="ASYNC_METHOD"/>
    <xsd:element ref="URL" minOccurs="0"/>
    <xsd:element ref="PORT_NUMBER" minOccurs="0"/>
    <!-- In YOUR_TOKEN1 save your token or pointer or any 32 bit data that
    will return to you. -->
    <xsd:element ref="YOUR_TOKEN1" minOccurs="0"/>
    <!-- In YOUR_TOKEN2 save your token or pointer or any 32 bit data that
    will return to <xsd:element ref="YOUR_TOKEN2" minOccurs="0"/>
    <xsd:any namespace="##other" processContents="skip" minOccurs="0"
    maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

## An XML Example

The following example shows the login XML sent to the MCU, and the corresponding response XML.

```
<TRANS_MCU><!-- The root is the transaction name. -->
  <TRANS_COMMON_PARAMS>
    <!-- In the login transaction the next two parameters have no meaning. Just
    send 0. -->
    <MCU_TOKEN>0</MCU_TOKEN>
    <MCU_USER_TOKEN>0</MCU_USER_TOKEN>
    <MESSAGE_ID>1</MESSAGE_ID>
  </TRANS_COMMON_PARAMS>
  <!-- The next element is the action (Login), and below it are the action's
  parameters, in this case, the Login parameters. -->
  <ACTION>
    <LOGIN>
      <!-- The next parameters are the MCU IP and the port number. -->
      <MCU_IP>
        <IP>172.22.189.154</IP>
        <LISTEN_PORT>80</LISTEN_PORT>
        <HOST_NAME/>
      </MCU_IP>
      <!-- The user and password strings. -->
      <USER_NAME>POLYCOM</USER_NAME>
      <PASSWORD>POLYCOM</PASSWORD>
      <STATION_NAME>EMA.F3-JUDITHS</STATION_NAME>
```

```
<COMPRESSION>true</COMPRESSION>
</LOGIN>
</ACTION>
</TRANS_MCU>
```

The response will be according to the **response\_trans\_mcu** schema:

```
<RESPONSE_TRANS_MCU>
<!-- The status of the action, STATUS_OK (success). -->
  <RETURN_STATUS>
    <ID>0</ID>
    <DESCRIPTION>STATUS_OK</DESCRIPTION>
    <YOUR_TOKEN1>0</YOUR_TOKEN1>
    <YOUR_TOKEN2>0</YOUR_TOKEN2>
    <MESSAGE_ID>1</MESSAGE_ID>
    <DESCRIPTION_EX/>
  </RETURN_STATUS>
  <ACTION>
    <!-- The sent action -->
    <LOGIN>
      <!-- The two tokens that you should keep and send with all further
      transactions. -->
      <MCU_TOKEN>537</MCU_TOKEN>
      <MCU_USER_TOKEN>537</MCU_USER_TOKEN>
      <VERSION_LIST><!-- The version numbers.-->
        <MCU_VERSION>
          <MAIN>1</MAIN>
          <MAJOR>1</MAJOR>
          <MINOR>0</MINOR>
          <INTERNAL>70</INTERNAL>
          <PRIVATE_DESCRIPTION>vasily_2007-01-25_14-
          36</PRIVATE_DESCRIPTION>
        </MCU_VERSION>
        <MCMS_VERSION>
          <MAIN>0</MAIN>
          <MAJOR>0</MAJOR>
          <MINOR>0</MINOR>
          <INTERNAL>0</INTERNAL>
          <PRIVATE_DESCRIPTION/>
        </MCMS_VERSION>
      </VERSION_LIST>
      <AUTHORIZATION_GROUP>administrator</AUTHORIZATION_GROUP>
      <API_NUMBER>2000</API_NUMBER>
      <PRODUCT_TYPE>Rmx_2000</PRODUCT_TYPE>
```

```
<HTTP_PORT>80</HTTP_PORT>
</LOGIN>
</ACTION>
</RESPONSE_TRANS_MCU>
```

## Keeping the Connection with the MCU Alive

If any XML API call is not performed within an interval of two minutes, the MCU will disconnect the client, and your token numbers (MCU\_TOKEN and MCU\_USER\_TOKEN) will no longer be valid.

If your application interacts frequently with the XML API, it is highly recommended to keep the connection alive, and use the tokens that you receive from the initial login request for all transactions. Do not log in and log out for each request, as this can cause problems with the RMX XML API and the RMX client.

To keep the connection alive, make your application send an XML page every 30 seconds (at least). For example, you can send the following XML transaction to get the MCU state:

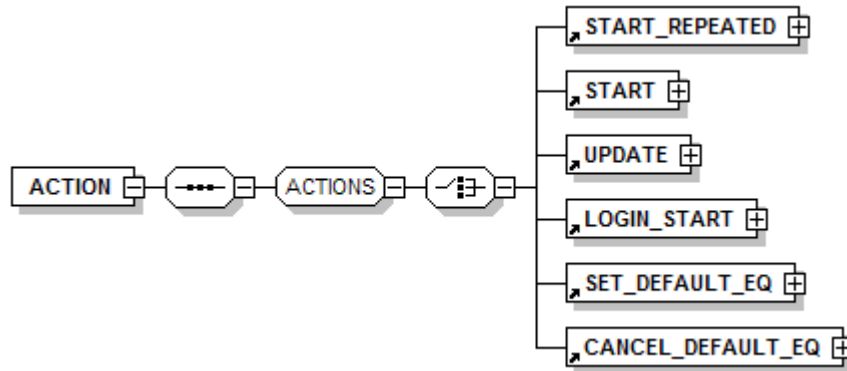
```
<!-- The root is the transaction name. -->
<TRANS_MCU>
  <TRANS_COMMON_PARAMS>
    <MCU_TOKEN>26</MCU_TOKEN>
    <MCU_USER_TOKEN>26</MCU_USER_TOKEN>
    <MESSAGE_ID>147</MESSAGE_ID>
  </TRANS_COMMON_PARAMS>
  <ACTION>
    <GET_STATE/>
  </ACTION>
</TRANS_MCU>
```

## Working with the Response and Transaction Schemas

The response and transaction schemas generally contain an element named **ACTION**. This element contains a group named **ACTIONS**, which contains a choice of action elements. Each action element identifies an action that can be requested using the schema, or to which a response is being returned by the schema.

We recommend that when using the response and transaction schemas, you use the **ACTION** element and the **ACTIONS** group, although for compatibility with the XAP you can use the **ACTION** group as in the past.

For example, in the **trans\_res\_1** schema, the **ACTION** element, contains the **ACTIONS** group, which contains the **START\_REPEATED** (*not supported*), **START**, **UPDATE**, **LOGIN\_START** (*not supported*), **SET\_DEFAULT\_EQ** and **CANCEL\_DEFAULT\_EQ** elements, as shown below:

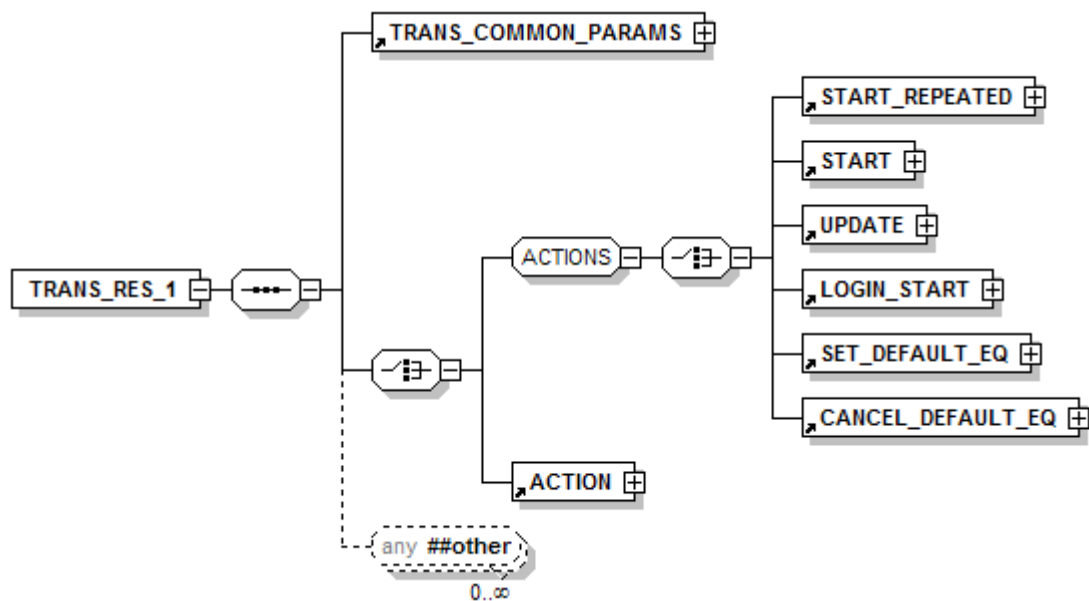


```

<xsd:element name="ACTION">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:group ref="ACTIONS"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:group name="ACTIONS">
  <xsd:choice>
    <xsd:element ref="START_REPEATED"/>    <!--not supported-->
    <xsd:element ref="START"/>
    <xsd:element ref="UPDATE"/>
    <xsd:element ref="LOGIN_START"/>    <!--not supported-->
    <xsd:element ref="SET_DEFAULT_EQ"/>
    <xsd:element ref="CANCEL_DEFAULT_EQ"/>
  </xsd:choice>
</xsd:group>

```

The **TRANS\_RES\_1** element contains a choice of the **ACTIONS** group and the **ACTION** element, as shown below:





```
<xsd:element name="TRANS_RES_1">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="TRANS_COMMON_PARAMS"/>
      <xsd:choice>
        <xsd:group ref="ACTIONS"/>
        <xsd:element ref="ACTION"/>
      </xsd:choice>
      <xsd:any namespace="##other" processContents="skip" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

## Audio Muting a Participant - An XML Example

This example shows the XML that must be sent in order to audio mute a party according to the **trans\_conf\_2** schema, and the response that will be received.

```
<TRANS_CONF_2>
  <TRANS_COMMON_PARAMS>
    <MCU_TOKEN>1005</MCU_TOKEN>
    <MCU_USER_TOKEN>1005</MCU_USER_TOKEN>
    <ASYNCR>
      <YOUR_TOKEN1>0</YOUR_TOKEN1>
      <YOUR_TOKEN2>0</YOUR_TOKEN2>
    </ASYNCR>
    <MESSAGE_ID>85</MESSAGE_ID>
  </TRANS_COMMON_PARAMS>
  <ACTION>
    <SET_AUDIO_VIDEO_MUTE>
      <ID>2</ID>
      <AUDIO_MUTE>true</AUDIO_MUTE>
      <VIDEO_MUTE>false</VIDEO_MUTE>
      <PARTY_ID>0</PARTY_ID>
    </SET_AUDIO_VIDEO_MUTE>
  </ACTION>
</TRANS_CONF_2>
```

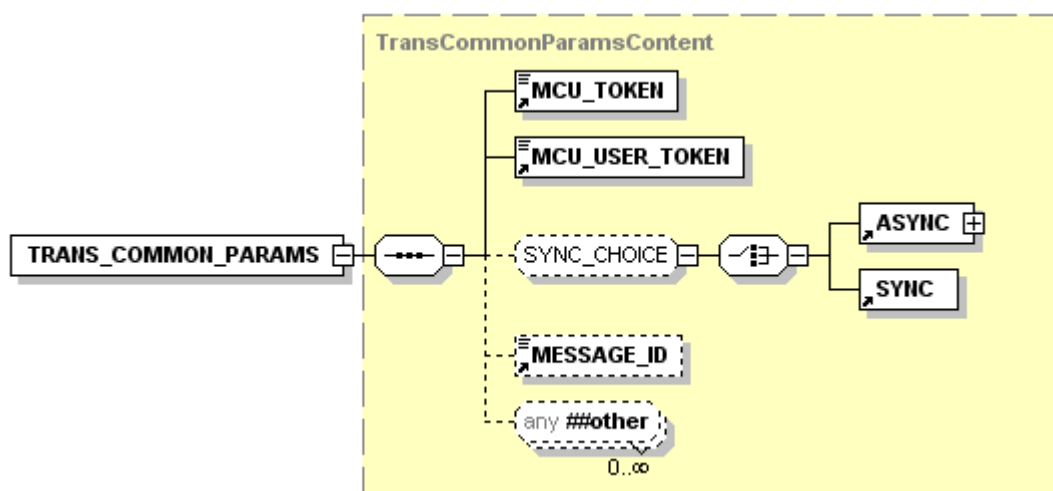
The response will be as follows, according to the **response\_trans\_conf** schema:

```
<RESPONSE_TRANS_CONF>
  <RETURN_STATUS>
    <ID>1</ID>
    <DESCRIPTION>IN_PROGRESS</DESCRIPTION>
    <YOUR_TOKEN1>0</YOUR_TOKEN1>
    <YOUR_TOKEN2>0</YOUR_TOKEN2>
    <MESSAGE_ID>85</MESSAGE_ID>
    <DESCRIPTION_EX/>
  </RETURN_STATUS>
  <ACTION>
    <SET_AUDIO_VIDEO_MUTE/>
  </ACTION>
</RESPONSE_TRANS_CONF>
```

## Transaction Processing

The client is responsible for managing transactions. The MCU itself returns responses immediately. The RMX XML API provides two tokens which can be used to identify the transaction response. These tokens are returned in the **YOUR\_TOKEN1** and **YOUR\_TOKEN2** elements which are contained in the **ASYNC** element in the **SYNC\_CHOICE** group in the **TRANS\_COMMON\_PARAMS** element in the **common\_trans** schema.

The **SYNC\_CHOICE** group is optional, and only needs to be specified if you want to use the contents of the **YOUR\_TOKEN1** and **YOUR\_TOKEN2** elements, and in this case should be specified with the **ASYNC** value. The **SYNC** value is obsolete.

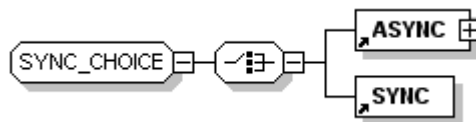


```

<xsd:element name="TRANS_COMMON_PARAMS" type="TransCommonParamsContent"/>
<xsd:complexType name="TransCommonParamsContent">
  <xsd:sequence>
    <xsd:element ref="MCU_TOKEN"/>
    <xsd:element ref="MCU_USER_TOKEN"/>
    <xsd:group ref="SYNC_CHOICE" minOccurs="0"/>
    <xsd:element ref="MESSAGE_ID" minOccurs="0"/>
    <xsd:any namespace="##other" processContents="skip" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

```

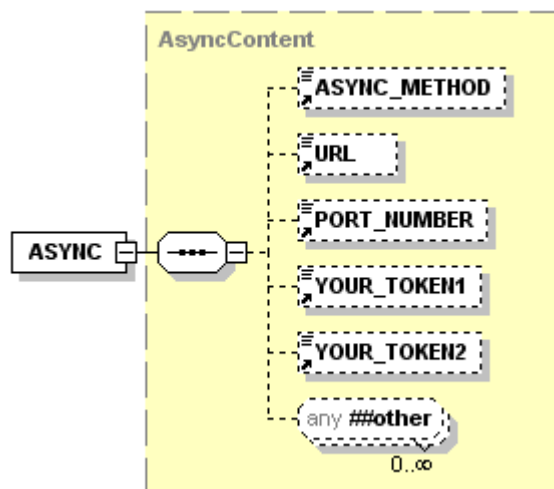
If you wish to use user tokens or pointers to help identify transaction responses, you should specify the **ASYNC** value for the **SYNC\_CHOICE** group. Then when sending a transaction, you can use **YOUR\_TOKEN1** or **YOUR\_TOKEN2** to save a token or pointer or any 32 bit data.



```

<xsd:group name="SYNC_CHOICE">
  <xsd:choice>
    <xsd:element ref="ASYNC"/>
    <xsd:element ref="SYNC"/>
  </xsd:choice>
</xsd:group>

```



```
<xsd:complexType name="AsyncContent">
  <!-- The ASYNC_METHOD, URL and PORT_NUMBER elements are obsolete. The YOUR_TOKEN1
  and YOUR_TOKEN2 elements can be used to save tokens, pointers or any 32 bit data
  to be used to identify transaction responses when the client application is
  working in asynchronous mode. -->
  <xsd:sequence>
    <xsd:element ref="ASYNC_METHOD" minOccurs="0"/>
    <xsd:element ref="URL" minOccurs="0"/>
    <xsd:element ref="PORT_NUMBER" minOccurs="0"/>
    <xsd:element ref="YOUR_TOKEN1" minOccurs="0"/>
    <xsd:element ref="YOUR_TOKEN2" minOccurs="0"/>
    <xsd:any namespace="##other" processContents="skip" minOccurs="0"
    maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

## Polling for Data

The XML API cannot push requested data when data is changed. Therefore when you want to monitor objects like conferences you have to poll for them. It is recommended to poll for data every 30 seconds using the GET\_STATE transaction from the **trans\_mcu** schema.

The XML API provides a mechanism to make the polling more efficient. When you query for an object, the RESPONSE\_XML contains an object token, OBJ\_TOKEN. If you send this token in your next request, then the RESPONSE\_XML will include any members in the object that were changed since the previous call, and a new object token. If no changes were made, you will get the same token and no data. The RESPONSE\_XML also contains a flag indicating if there any changes were made.

For example, if a participant was added to a conference after you made a get request, then in the next request, provided that you send the correct object token, you will only get the data of the new participant. If participants were removed, you will get a list of the deleted participants.

For any get request, if you want to get the full data, send **-1** in the object token.

The following is an XML example for retrieving the Meeting Room list:

---

**Note:** The Meeting Room list contains Meeting Room, Entry Queue and SIP Factory summary information.

---

## The initial REQUEST\_XML

```
<TRANS_RES_LIST>
  <TRANS_COMMON_PARAMS>
    <MCU_TOKEN>3</MCU_TOKEN>
    <MCU_USER_TOKEN>3</MCU_USER_TOKEN>
    <MESSAGE_ID>143</MESSAGE_ID>
  </TRANS_COMMON_PARAMS>
  <ACTION>
    <GET_MEETING_ROOM_LIST>
      <OBJ_TOKEN>-1</OBJ_TOKEN>
      <!-- The object token is -1, in order to get the full data. -->
    </GET_MEETING_ROOM_LIST>
  </ACTION>
</TRANS_RES_LIST>
```

## The RESPONSE\_XML (one Meeting Room in the list)

```
<RESPONSE_TRANS_RES_LIST>
  <RETURN_STATUS>
    <ID>0</ID>
    <DESCRIPTION>STATUS_OK</DESCRIPTION>
    <YOUR_TOKEN1>0</YOUR_TOKEN1>
    <YOUR_TOKEN2>0</YOUR_TOKEN2>
    <MESSAGE_ID>143</MESSAGE_ID>
    <DESCRIPTION_EX/>
  </RETURN_STATUS>
  <ACTION>
    <GET_MEETING_ROOM_LIST>
      <MEETING_ROOM_SUMMARY_LS>
        <OBJ_TOKEN>12</OBJ_TOKEN>
        <!-- The returned object token is 1. This will be used in the next get. -->
        <CHANGED>true</CHANGED>
        <DELETED_RES_LIST/>
        <DEFAULT_EQ_NAME/>
        <MEETING_ROOM_SUMMARY>
          <NAME>Maple_Room</NAME>
          <ID>0</ID>
          <RES_CHANGE>new</RES_CHANGE>
          <DURATION>
            <HOUR>2</HOUR>
            <MINUTE>0</MINUTE>
            <SECOND>0</SECOND>
          </DURATION>
        </MEETING_ROOM_SUMMARY>
      </MEETING_ROOM_SUMMARY_LS>
    </GET_MEETING_ROOM_LIST>
  </ACTION>
</RESPONSE_TRANS_RES_LIST>
```

```

<MEET_ME_PHONE />
<MR_STATE>passive</MR_STATE>
<ENTRY_QUEUE>false</ENTRY_QUEUE>
<ENTRY_PASSWORD>0512</ENTRY_PASSWORD>
<PASSWORD>2968</PASSWORD>
<NUMERIC_ID>1000</NUMERIC_ID>
<NUM_PARTIES>0</NUM_PARTIES>
<NUM_UNDEFINED_PARTIES>0</NUM_UNDEFINED_PARTIES>
<DIAL_IN_H323_SRV_PREFIX_LIST>
  <DIAL_IN_H323_SRV_PREFIX>
    <NAME>Default IP Service</NAME>
    <PREFIX>9431</PREFIX>
  </DIAL_IN_H323_SRV_PREFIX>
</DIAL_IN_H323_SRV_PREFIX_LIST>
<ENCRYPTION>false</ENCRYPTION>
<SIP_FACTORY>false</SIP_FACTORY>
<AD_HOC_PROFILE_ID>1</AD_HOC_PROFILE_ID>
<DISPLAY_NAME>Maple_Room</DISPLAY_NAME>
<IS_TELEPRESENCE_MODE>false</IS_TELEPRESENCE_MODE>
</MEETING_ROOM_SUMMARY>
</MEETING_ROOM_SUMMARY_LS>
</GET_MEETING_ROOM_LIST >
</ACTION>
</RESPONSE_TRANS_RES_LIST>

```

### The next REQUEST\_XML

```

<TRANS_RES_LIST>
  <TRANS_COMMON_PARAMS>
    <MCU_TOKEN>3</MCU_TOKEN>
    <MCU_USER_TOKEN>3</MCU_USER_TOKEN>
    <MESSAGE_ID>177</MESSAGE_ID>
  </TRANS_COMMON_PARAMS>
  <ACTION>
    <GET_MEETING_ROOM_LIST>
      <OBJ_TOKEN>12</OBJ_TOKEN>
      <!-- The object token is 1 as returned in the previous received object, in order
      to get differential data. -->
    </GET_MEETING_ROOM_LIST>
  </ACTION>
</TRANS_RES_LIST>

```

**If no changes happened in the list, the RESPONSE\_XML will be:**

```

<RESPONSE_TRANS_RES_LIST>
  <RETURN_STATUS>

```

```

<ID>0</ID>
<DESCRIPTION>STATUS_OK</DESCRIPTION>
<YOUR_TOKEN1>0</YOUR_TOKEN1>
<YOUR_TOKEN2>0</YOUR_TOKEN2>
<MESSAGE_ID>177</MESSAGE_ID>
<DESCRIPTION_EX/>
</RETURN_STATUS>
<ACTION>
  <GET_MEETING_ROOM_LIST>
    <MEETING_ROOM_SUMMARY_LS>
      <OBJ_TOKEN>12</OBJ_TOKEN>
      <!-- The returned object token is the same as in the previous
      get request, since no changes were made. -->
      <CHANGED>false</CHANGED>
    </MEETING_ROOM_SUMMARY_LS>
  </GET_MEETING_ROOM_LIST>
</ACTION>
</RESPONSE_TRANS_RES_LIST>

```

## Retrieving Channel Information for IP Participants

Detailed channel information for IP participants is returned in the **IP\_MONITOR\_CHANNELS** element, which is contained in the **ONGOING\_PARTY** element in the **obj\_ongoing\_party** schema. Since this element contains a large amount of information, its contents are only retrieved in response to the **trans\_party GET** transaction, and not in response to the **trans\_conf\_2 GET** transaction.

## The Compression Mechanism

To speed up transmission times, the RMX XML API provides an option that enables client applications to send HTTP XML strings to the MCU in compressed format, and to receive RMX XML strings from the MCU in compressed format. This is the recommended method of work for all XML strings that are larger than 10 KB.

---

**Note** To maintain backwards compatibility with the XAP, you can work with the RMX XML API without using compression. However, working with compression is strongly recommended.

---

## Sending XML requests in Compressed Format

It is recommended that all XML request strings larger than 10 KB are compressed before being sent over the HTTP connection to the MCU.

### To send an XML request string in compressed format:

Compress the XML string using the OpenSource **zlib** component.

Include the following header in the header file sent with the HTTP:

```
Content-Encoding: zip
```

Send the header file and compressed contents to the MCU.

## Receiving XML responses from the MCU in Compressed Format

When you log in to an MCU, you can use the **COMPRESSION** element to specify whether or not you want to receive responses in compressed mode. If you request to receive responses in compressed mode, then all XML response strings that are larger than 10 KB will be sent over the HTTP to the client in compressed format. XML response strings that are smaller than 10 KB will not be compressed.

All compressed response strings will be sent with a header file which includes the following header:

```
Content-Encoding: zip
```

If the header file contains the header `Content-Encoding: zip`, then decompress the XML string using the OpenSource **zlib** component.

### The **COMPRESSION** element

The **COMPRESSION** element is located in the **common\_trans** schema and is an optional component of the **LOGIN** element in the **trans\_mcu** schema. The **COMPRESSION** element is used to indicate whether or not responses should be compressed. By default there is no compression of responses.

The **COMPRESSION** element is defined as follows:

```
<xsd:element name="COMPRESSION" type="xsd:boolean"/>
```

The **LOGIN** element is defined as follows:

```
<xsd:element name="LOGIN" type="LoginRequestContent"/>
<xsd:complexType name="LoginRequestContent">
  <xsd:sequence>
    <xsd:element ref="MCU_IP"/>
    <xsd:element ref="USER_NAME"/>
    <xsd:element ref="PASSWORD"/>
    <xsd:element ref="STATION_NAME" minOccurs="0"/>
    <xsd:element ref="COMPRESSION" minOccurs="0"/>
    <xsd:element ref="CONFERENCE_RECORDER" minOccurs="0"/>
    <xsd:element ref="NEW_PASSWORD" minOccurs="0"/>
    <xsd:any namespace="##other" processContents="skip" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```



The following is an example of the XML string needed to log in to the MCU when you want to receive all XML streams bigger than 10 KB in compressed mode:

```
<TRANS_MCU>
  <TRANS_COMMON_PARAMS>
    <MCU_TOKEN>0</MCU_TOKEN>
    <MCU_USER_TOKEN>0</MCU_USER_TOKEN>
    <ASync>
      <YOUR_TOKEN1>0</YOUR_TOKEN1>
      <YOUR_TOKEN2>0</YOUR_TOKEN2>
    </ASync>
    <MESSAGE_ID>0</MESSAGE_ID>
  </TRANS_COMMON_PARAMS>
  <ACTION>
    <LOGIN>
      <MCU_IP>
        <IP>127.0.0.1</IP>
        <LISTEN_PORT>80</LISTEN_PORT>
        <HOST_NAME/>
      </MCU_IP>
      <USER_NAME>POLYCOM</USER_NAME>
      <PASSWORD>POLYCOM</PASSWORD>
      <STATION_NAME>EMMA.F3-JUDITHS</STATION_NAME>
      <COMPRESSION>true</COMPRESSION>
    </LOGIN>
  </ACTION>
</TRANS_MCU>
```