Avaya Communication Manager API

XML Programmer's Guide

03-300201
Issue 1
May 2004

Notice

Every effort was made to ensure that the information in this document was complete and accurate at the time of printing. However, information is subject to change.

Warranty

Avaya Inc. provides a limited warranty on this product. Refer to your sales agreement to establish the terms of the limited warranty. In addition, Avaya's standard warranty language as well as information regarding support for this product, while under warranty, is available through the following Web site: http://www.avaya.com/support.

Preventing Toll Fraud

"Toll fraud" is the unauthorized use of your telecommunications system by an unauthorized party (for example, a person who is not a corporate employee, agent, subcontractor, or is not working on your company's behalf). Be aware that there may be a risk of toll fraud associated with your system and that, if toll fraud occurs, it can result in substantial additional charges for your telecommunications services.

Avaya Fraud Intervention

If you suspect that you are being victimized by toll fraud and you need technical assistance or support, in the United States and Canada, call the Technical Service Center's Toll Fraud Intervention Hotline at 1-800-643-2353.

Disclaimer

Avaya is not responsible for any modifications, additions or deletions to the original published version of this documentation unless such modifications, additions or deletions were performed by Avaya. Customer and/or End User agree to indemnify and hold harmless Avaya, Avaya's agents, servants and employees against all claims, lawsuits, demands and judgments arising out of, or in connection with, subsequent modifications, additions or deletions to this documentation to the extent made by the Customer or End User.

How to Get Help

For additional support telephone numbers, go to the Avaya support Web site: http://www.avaya.com/support. If you are:

- Within the United States, click the *Escalation Management* link. Then click the appropriate link for the type of support you need.
- Outside the United States, click the *Escalation Management* link. Then click the *International Services* link that includes telephone numbers for the international Centers of Excellence.

Providing Telecommunications Security

Telecommunications security (of voice, data, and/or video communications) is the prevention of any type of intrusion to (that is, either unauthorized or malicious access to or use of) your company's telecommunications equipment by some party.

Your company's "telecommunications equipment" includes both this Avaya product and any other voice/data/video equipment that could be accessed via this Avaya product (that is, "networked equipment").

An "outside party" is anyone who is not a corporate employee, agent, subcontractor, or is not working on your company's behalf. Whereas, a "malicious party" is anyone (including someone who may be otherwise authorized) who accesses your telecommunications equipment with either malicious or mischievous intent.

Such intrusions may be either to/through synchronous (time-multiplexed and/or circuit-based) or asynchronous (character-, message-, or packet-based) equipment or interfaces for reasons of:

- Utilization (of capabilities special to the accessed equipment)
- Theft (such as, of intellectual property, financial assets, or toll facility access)
- Eavesdropping (privacy invasions to humans)
- Mischief (troubling, but apparently innocuous, tampering)
- Harm (such as harmful tampering, data loss or alteration, regardless of motive or intent)

Be aware that there may be a risk of unauthorized intrusions associated with your system and/or its networked equipment. Also realize that, if such an intrusion should occur, it could result in a variety of losses to your company (including but not limited to, human/data privacy, intellectual property, material assets, financial resources, labor costs, and/or legal costs).

Responsibility for Your Company's Telecommunications Security

The final responsibility for securing both this system and its networked equipment rests with you - Avaya's customer system administrator, your telecommunications peers, and your managers. Base the fulfillment of your responsibility on acquired knowledge and resources from a variety of sources including but not limited to:

- Installation documents
- System administration documents
- Security documents
- Hardware-/software-based security tools
- Shared information between you and your peers
- Telecommunications security experts

To prevent intrusions to your telecommunications equipment, you and your peers should carefully program and configure:

- Your Avaya-provided telecommunications systems and their interfaces
- Your Avaya-provided software applications, as well as their underlying hardware/software platforms and interfaces
- Any other equipment networked to your Avaya products

TCP/IP Facilities

Customers may experience differences in product performance, reliability and security depending upon network configurations/design and topologies, even when the product performs as warranted.

Standards Compliance

Avaya Inc. is not responsible for any radio or television interference caused by unauthorized modifications of this equipment or the substitution or attachment of connecting cables and equipment other than those specified by Avaya Inc. The correction of interference caused by such unauthorized modifications, substitution or attachment will be the responsibility of the user. Pursuant to Part 15 of the Federal Communications Commission (FCC) Rules, the user is cautioned that changes or modifications not expressly approved by Avaya Inc. could void the user's authority to operate this equipment.

Product Safety Standards

This product complies with and conforms to the following international Product Safety standards as applicable:

Safety of Information Technology Equipment, IEC 60950, 3rd Edition including all relevant national deviations as listed in Compliance with IEC for Electrical Equipment (IECEE) CB-96A.

Safety of Information Technology Equipment, CAN/CSA-C22.2 No. 60950-00 / UL 60950, 3rd Edition

Safety Requirements for Customer Equipment, ACA Technical Standard (TS) 001 - 1997

One or more of the following Mexican national standards, as applicable: NOM 001 SCFI 1993, NOM SCFI 016 1993, NOM 019 SCFI 1998

The equipment described in this document may contain Class 1 LASER Device(s). These devices comply with the following standards:

- EN 60825-1, Edition 1.1, 1998-01
- 21 CFR 1040.10 and CFR 1040.11.

The LASER devices operate within the following parameters:

- Maximum power output: -5 dBm to -8 dBm
- Center Wavelength: 1310 nm to 1360 nm

Luokan 1 Laserlaite

Klass 1 Laser Apparat

Use of controls or adjustments or performance of procedures other than those specified herein may result in hazardous radiation exposures. Contact your Avaya representative for more laser product information.

Electromagnetic Compatibility (EMC) Standards

This product complies with and conforms to the following international EMC standards and all relevant national deviations:

Limits and Methods of Measurement of Radio Interference of Information Technology Equipment, CISPR 22:1997 and EN55022:1998.

Information Technology Equipment – Immunity Characteristics – Limits and Methods of Measurement, CISPR 24:1997 and EN55024:1998, including:

- Electrostatic Discharge (ESD) IEC 61000-4-2
- Radiated Immunity IEC 61000-4-3
- Electrical Fast Transient IEC 61000-4-4
- Lightning Effects IEC 61000-4-5
- Conducted Immunity IEC 61000-4-6
- Mains Frequency Magnetic Field IEC 61000-4-8
- Voltage Dips and Variations IEC 61000-4-11
- Powerline Harmonics IEC 61000-3-2
- Voltage Fluctuations and Flicker IEC 61000-3-3

Federal Communications Commission Statement

Part 15:

Note: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Part 68: Answer-Supervision Signaling

Allowing this equipment to be operated in a manner that does not provide proper answer-supervision signaling is in violation of Part 68 rules. This equipment returns answer-supervision signals to the public switched network when:

- answered by the called station,
- answered by the attendant, or
- routed to a recorded announcement that can be administered by the customer premises equipment (CPE) user.

This equipment returns answer-supervision signals on all direct inward dialed (DID) calls forwarded back to the public switched telephone network. Permissible exceptions are:

- A call is unanswered.
- A busy tone is received.
- A reorder tone is received.

Avaya attests that this registered equipment is capable of providing users access to interstate providers of operator services through the use of access codes. Modification of this equipment by call aggregators to block access dialing codes is a violation of the Telephone Operator Consumers Act of 1990.

REN Number

For MCC1, SCC1, CMC1, G600, and G650 Media Gateways:

This equipment complies with Part 68 of the FCC rules. On either the rear or inside the front cover of this equipment is a label that contains, among other information, the FCC registration number, and ringer equivalence number (REN) for this equipment. If requested, this information must be provided to the telephone company.

For G350 and G700 Media Gateways:

This equipment complies with Part 68 of the FCC rules and the requirements adopted by the ACTA. On the rear of this equipment is a label that contains, among other information, a product identifier in the format US:AAAEQ##TXXXX. The digits represented by ## are the ringer equivalence number (REN) without a decimal point (for example, 03 is a REN of 0.3). If requested, this number must be provided to the telephone company.

For all media gateways:

The REN is used to determine the quantity of devices that may be connected to the telephone line. Excessive RENs on the telephone line may result in devices not ringing in response to an incoming call. In most, but not all areas, the sum of RENs should not exceed 5.0. To be certain of the number of devices that may be connected to a line, as determined by the total RENs, contact the local telephone company.

REN is not required for some types of analog or digital facilities.

Means of Connection

Connection of this equipment to the telephone network is shown in the following tables.

For MCC1, SCC1, CMC1, G600, and G650 Media Gateways:

Manufacturer's Port Identifier	FIC Code	SOC/REN/ A.S. Code	Network Jacks
Off premises station	OL13C	9.0F	RJ2GX, RJ21X, RJ11C
DID trunk	02RV2-T	0.0B	RJ2GX, RJ21X
CO trunk	02GS2	0.3A	RJ21X
	02LS2	0.3A	RJ21X
Tie trunk	TL31M	9.0F	RJ2GX
Basic Rate Interface	02IS5	6.0F, 6.0Y	RJ49C
1.544 digital interface	04DU9-BN	6.0F	RJ48C, RJ48M
	04DU9-IKN	6.0F	RJ48C, RJ48M
	04DU9-ISN	6.0F	RJ48C, RJ48M
120A4 channel service unit	04DU9-DN	6.0Y	RJ48C

For G350 and G700 Media Gateways:

Manufacturer's Port Identifier	FIC Code	SOC/REN/ A.S. Code	Network Jacks
Ground Start CO trunk	02GS2	1.0A	RJ11C
DID trunk	02RV2-T	AS.0	RJ11C
Loop Start CO trunk	02LS2	0.5A	RJ11C
1.544 digital interface	04DU9-BN	6.0Y	RJ48C
	04DU9-DN	6.0Y	RJ48C
	04DU9-IKN	6.0Y	RJ48C
	04DU9-ISN	6.0Y	RJ48C
Basic Rate Interface	02IS5	6.0F	RJ49C

For all media gateways:

If the terminal equipment (for example, the media server or media gateway) causes harm to the telephone network, the telephone company will notify you in advance that temporary discontinuance of service may be required. But if advance notice is not practical, the telephone company will notify the customer as soon as possible. Also, you will be advised of your right to file a complaint with the FCC if you believe it is necessary.

The telephone company may make changes in its facilities, equipment, operations or procedures that could affect the operation of the equipment. If this happens, the telephone company will provide advance notice in order for you to make necessary modifications to maintain uninterrupted service.

If trouble is experienced with this equipment, for repair or warranty information, please contact the Technical Service Center at 1-800-242- 2121 or contact your local Avaya representative. If the equipment is causing harm to the telephone network, the telephone company may request that you disconnect the equipment until the problem is resolved.

A plug and jack used to connect this equipment to the premises wiring and telephone network must comply with the applicable FCC Part 68 rules and requirements adopted by the ACTA. A compliant telephone cord and modular plug is provided with this product. It is designed to be connected to a compatible modular jack that is also compliant. It is recommended that repairs be performed by Avaya certified technicians.

The equipment cannot be used on public coin phone service provided by the telephone company. Connection to party line service is subject to state tariffs. Contact the state public utility commission, public service commission or corporation commission for information.

This equipment, if it uses a telephone receiver, is hearing aid compatible.

Canadian Department of Communications (DOC) Interference Information

This Class A digital apparatus complies with Canadian ICES-003.

Cet appareil numérique de la classe A est conforme à la norme NMB-003 du Canada.

This equipment meets the applicable Industry Canada Terminal Equipment Technical Specifications. This is confirmed by the registration number. The abbreviation, IC, before the registration number signifies that registration was performed based on a Declaration of Conformity indicating that Industry Canada technical specifications were met. It does not imply that Industry Canada approved the equipment.

Declarations of Conformity

United States FCC Part 68 Supplier's Declaration of Conformity (SDoC)

Avaya Inc. in the United States of America hereby certifies that the equipment described in this document and bearing a TIA TSB-168 label identification number complies with the FCC's Rules and Regulations 47 CFR Part 68, and the Administrative Council on Terminal Attachments (ACTA) adopted technical criteria.

Avaya further asserts that Avaya handset-equipped terminal equipment described in this document complies with Paragraph 68.316 of the FCC Rules and Regulations defining Hearing Aid Compatibility and is deemed compatible with hearing aids.

Copies of SDoCs signed by the Responsible Party in the U. S. can be obtained by contacting your local sales representative and are available on the following Web site: http://www.avaya.com/support.

All Avaya media servers and media gateways are compliant with FCC Part 68, but many have been registered with the FCC before the SDoC process was available. A list of all Avaya registered products may be found at: http://www.part68.org by conducting a search using "Avaya" as manufacturer.

European Union Declarations of Conformity

CE

Avaya Inc. declares that the equipment specified in this document bearing the "CE" (*Conformité Europeénne*) mark conforms to the European Union Radio and Telecommunications Terminal Equipment Directive (1999/5/EC), including the Electromagnetic Compatibility Directive (89/336/EEC) and Low Voltage Directive (73/23/EEC). This equipment has been certified to meet CTR3 Basic Rate Interface (BRI) and CTR4 Primary Rate Interface (PRI) and subsets thereof in CTR12 and CTR13, as applicable.

Copies of these Declarations of Conformity (DoCs) can be obtained by contacting your local sales representative and are available on the following Web site: http://www.avaya.com/support.

Japan

This is a Class A product based on the standard of the Voluntary Control Council for Interference by Information Technology Equipment (VCCI). If this equipment is used in a domestic environment, radio disturbance may occur, in which case, the user may be required to take corrective actions.

この装置は、情報処理装置等電波障害自主規制協議会（ＶＣＣＩ）の基準に基づくクラスＡ情報技術装置です。この装置を家庭環境で使用すると電波妨害を引き起こすことがあります。この場合には使用者が適切な対策を講ずるよう要求されることがあります。

Third-party license agreements:

Portions of this product include technology used under license as listed below, and are copyright of the respective companies and/or their licensors.

A) This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

The Apache Software License, Version 1.1

Copyright (c) 2000 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

“This product includes software developed by the Apache Software Foundation (http://www.apache.org/).”

Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names “Apache” and “Apache Software Foundation” must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ‘‘AS IS’’ AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Portions of this software are based upon public domain software originally written at the National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign.

B) Castor software

This product contains software developed by the Exolab Group (http://www.exolab.org).

Castor Copyright (C) 1999-2001 Intalio, Inc. All Rights Reserved.

Redistribution and use of this software and associated documentation (“Software”), with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain copyright statements and notices. Redistributions must also contain a copy of this document.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name “ExoLab” must not be used to endorse or promote products derived from this Software without prior written permission of ExoLab Group. For written permission, please contact info@exolab.org.

4. Products derived from this Software may not be called “ExoLab” nor may “ExoLab” appear in their names without prior written permission of ExoLab Group. Exolab is a registered trademark of ExoLab Group.

5. Due credit should be given to the ExoLab Group (http://www.exolab.org).

THIS SOFTWARE IS PROVIDED BY INTALIO, INC. AND CONTRIBUTORS ‘‘AS IS’’ AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.

IN NO EVENT SHALL INTALIO, INC. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

C) Java Service Wrapper Copyright (c) 1999, 2003 TanukiSoftware.org

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sub-license, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

To order copies of this and other documents:

Call: Avaya Publications Center
Voice 1.800.457.1235 or 1.207.866.6701
FAX 1.800.457.1764 or 1.207.626.7269

Write: Globalware Solutions
200 Ward Hill Avenue
Haverhill, MA 01835 USA
Attention: Avaya Account Management

E-mail: totalware@gwsmail.com

For the most current versions of documentation, go to the Avaya support Web site: http://www.avaya.com/support.

Contents

About this document

This chapter describes the:

Scope of this document

This document shows you on how to use the Communication Manager API to develop and debug applications that require device and media control.

- Chapter 1, “API Services” provides background information about Communication Manager API and CSTA.
- Chapter 2, “Getting started” gets you ready to program to this API.
- Chapter 3, “Writing a client application” and Chapter 4, “Debugging” guide you in developing and debugging applications.
- Appendix A, “Communication Manager Features” lists the switch features that your application can take advantage of.
- Appendix B, “Constant Values” lists the values for the XML message parameters which take a constant value and that are switch specific.
- The Glossary defines the terminology and acronyms used in this book.

Intended Audience

This document is written for applications developers. A developer must:

- know basic XML concepts
- be familiar with XML programming
- be familiar with XML Schema Definition (XSD)
- understand telephony concepts

You do not need to understand CSTA concepts or Avaya Communication Manager features and concepts, but they both might be helpful.

If you are new to CSTA, you may wish to start by reading *ECMA-269*, section 6.1, “CSTA Operational Model: Switching Sub-Domain Model”. Also become familiar with the table of contents so that you know the kinds of information available there. All of the descriptions of the CSTA services implemented by this API are also found in *Avaya Communication Manager Java Programmer’s Reference (called here XMLdoc)*, found online on the Avaya Developer Connection website (http://www.devconnectprogram.com) and on the Avaya Support Centre website (http://www.avaya.com/support) under “Communication Systems”.

For those new to Avaya Communication Manager, you may wish to take a course from Avaya University (http://www.avaya.com/learning) to learn more about Communication Manager and its features. It is recommended that you start with the *Avaya Communication Manager Overview* course (course ID AVA00383WEN). You may also wish to peruse Appendix A, "Communication Manager Features" in this guide to get some ideas of how applications can take advantage of Communication Manager's abilities.

Conventions used in this document

The following fonts are used in this document:

To represent...	This font is used...
Code and Linux commands	`<?xml version="1.0" encoding="UTF-8"?>`
XML requests, responses, events and field names	the `GetDeviceId` request
Window names	The buttons are assigned on the ***Station*** form.
Browser selections	Select **Member Login**
Hypertext links	Go to the http://www.avaya.com/support website. The term connector can be found in the glossary.

Related documents

While planning, developing, deploying, or troubleshooting your application, you may need to reference other Avaya Communication Manager API documents, Avaya Communication Manager documents, or CSTA documents listed below.

Communication Manager API documents

For developers, the other important source of API information is the XMLdoc:

- *Avaya Communication Manager API XML Programmer's Reference*

Here you can find details about each request, response, event and field in the API. You can also find out what parts of the CSTA protocol have been implemented.

Other Communication Manager API documents include:

- *Avaya Communication Manager API Overview* (03-300084)
- *Avaya Communication Manager API Release Notes* (03-300088)
- *Avaya Communication Manager API Quick Start* (03-300089)
- *Avaya Communication Manager API Installation and Administration* (03-300085)
- *Avaya Communication Manager API Management* (03-300086)

You can find all of these documents online on the Avaya Developer Connection website (http://www.devconnectprogram.com) and on the Avaya Support Centre website (http://www.avaya.com/support) under “Communications Systems”.

Communication Manager documents

Since this API gives you programmable access to Avaya Communication Manager features, you may wish to reference documents about that system. Those documents are found on the Avaya Support Centre website (http://www.avaya.com/support) under “Communications Systems”.

CSTA documents

The *XML Programmer’s Reference* (XMLdoc) contains much of what you need to know about CSTA services. For CSTA details not found in the XMLdoc or this document, please refer to the following CSTA documents. They are found in the Publications section of the ECMA web site (http://www.ecma-international.org/):

- *ECMA-269: Services for Computer Supported Telecommunications Applications (CSTA) Phase III*
- *ECMA-323: XML Protocol for Computer Supported Telecommunications Applications (CSTA) Phase III*
- *ECMA Technical Report TR/72: Glossary of Definitions and Terminology for Computer Supported Telecommunications Applications (CSTA) Phase III*

Other Avaya products and services

If your application also requires third-party call control, you may wish to consider also using the Avaya CVLAN SDK and server. For information on this companion product, see the DevConnect site. (http://www.devconnectprogram.com).

Providing feedback

Please provide input on this new API and its documents.

- How can the API be easier to use?
- What additional features would you like?
- What additional information would you like to see in this document?
- Are there any inaccuracies in the document?
- What do you like about the API or this document?

Please email feedback to devconcmapi@avaya.com.

Thank you.

1 API Services

This chapter provides an overview of what CSTA services the API supports and what extensions Avaya has implemented. This API supports these telephony services:

- device registration and control
- media control
- call recording, message playing and dubbing
- DTMF digit detection

These services are provided through an XML protocol. Some of the interfaces conform to the CSTA III standard (ECMA-269) and some are Avaya extensions to the CSTA standard.

CSTA specifies that for any given service some parameters are mandatory and some parameters are optional. To determine which of the optional parameters Avaya supports or which of the field values Avaya supports, refer to the requests and responses detailed in the programmer's reference (XMLdoc).

> **NOTE:**
> The ECMA standards body requests that CSTA-compliant implementations reflect conformance to the standard through a *Protocol Implementation Conformance Statement* (PICS). The Communication Manager API PICS is reflected in the programmer's reference.

This chapter lists:

Supported CSTA services

In CSTA, each service is defined to be a request that either comes from the application to the switch or from the switch to the application. This API, however, is based on a client/server model where the application is the client and the connector server software and the Communication Manager switch together act as the server. Thus, this API allows an application:

- to request services of the switch
- to request notification of asynchronous events on the switch

The following sets of CSTA services are supported in the Communication Manager API and described in the sections that follow:

Table 1: Supported CSTA services

Sets of supported CSTA services	CSTA specifications	CSTA XML protocol
Physical Device Services and Events	ECMA-269, section 21	ECMA-323, section 19
Voice Unit Services and Events	ECMA-269, section 26	ECMA-323, section 24
Monitoring Services	ECMA-269, section 15	ECMA-323, section 13

Physical Device Services and Events

CSTA's Physical Device Services provide physical device control. The control allows an application to manipulate and monitor the physical aspects of a device, which includes buttons, lamps, the display, and the ringer. The services simulate manual action on a device as well as provide the ability to request status of physical elements. The events provide notification of changes to the physical elements of the device. To learn how to use Physical Device Services and Events, see Monitoring and controlling physical elements on page 43.

This API supports these Physical Device Services:

Table 2: Physical Device Services

Service	Description	XSD
Button Press	Simulates the depression of a specified button on a device	button-press.xsd
Get Button Information	Gets the button information for either a specified button or all buttons on a device, including the button identifier, button function, associated extension (if applicable), and associated lamp identifier (if applicable)	get-button-information.xsd
Get Display	Gets a snapshot of the contents of the physical device's display	get-display.xsd
Get Hookswitch Status	Gets the hookswitch status of a specified device, either on hook or off hook	get-hookswitch-status.xsd
Get Lamp Mode	Gets the lamp mode status for either a specified button or all buttons on a device, including how the lamp is lit (flutter, off, steady, etc.), color and associated button	get-lamp-mode.xsd

1 of 2

Table 2: Physical Device Services

Service	Description	XSD
Get Message Waiting Indicator	Gets the message waiting status of a specified device, either on or off	get-message-waiting-indicator.xsd
Get Ringer Status	Gets the ringer status of the ringer associated with a device, including ring mode (ringing/not ringing) and the ring pattern (normal ring, priority ring, etc.)	get-ringer-status.xsd
Set Hookswitch Status	Sets the hookswitch status of a specified device to either onhook or offhook	set-hookswitch-status.xsd
	2 of 2	

This API supports these CSTA physical device events:

Table 3: Physical Device Events

Event	Description	XSD
Display Updated	Occurs if the contents of a device's display has changed	display-updated-event.xsd
Hookswitch Status Changed	Occurs if the switch has changed the device's hookswitch status	hookswitch-event.xsd
Lamp Mode Changed	Occurs if the lamp mode status of a particular lamp has changed	lamp-mode-event.xsd
Ringer Status Changed	Occurs if the ringer attribute associated with a device has changed status	ringer-status-event.xsd

Voice Unit Services and Events

CSTA's Voice Unit Services allow an application to record voice stream data coming into a device and to play messages to the device's outgoing voice stream.

Voice Unit Services has been extended to also provide a dubbing service and more specific stop, suspend, and resume services. See Extended Voice Unit Services on page 22.

To learn how to use Voice Unit Services, see Recording and playing voice media on page 47.

This API supports these Voice Unit Services:

Services	Description	XSD
Play Message	Plays a pre-recorded voice message on the outgoing RTP media stream of a particular device based on a specified criteria	play-message.xsd
Record Message	Starts recording the media stream for a specified device with the specified codec and criteria	record-message.xsd
Resume	Restarts the playing and recording of previously suspended messages at their current positions	resume.xsd
Stop	Stops the playing and recording of messages	stop.xsd
Suspend	Temporarily stops the playing and recording of messages and leaves their position pointers at their current locations	suspend.xsd
Delete Message	Deletes a specified message (Wave file) from the connector server	delete-message.xsd

The CSTA Voice Unit events that are supported include:

Table 5: Voice Unit Events

Events	Description	XSD
Play	Indicates that a message is being played	play-event.xsd
Record	Indicates that a message is being recorded	record-event.xsd
Stop	Indicates that a play or record operation for a message on a device has been stopped or has completed	stop-event.xsd
Suspend Play	Indicates that a message is suspended in play	suspend-play-event.xsd
Suspend Record	Indicates that a message is suspended during recording	suspend-record-event.xsd

Monitoring Services

CSTA's Monitoring Services allows clients to receive notification of events. By starting a monitor, the application indicates that it wants to be notified of events that occur on a device.

Once a monitor is established, the connector server notifies the application of relevant activity by sending messages called event reports, or simply events

To learn how to use Monitoring Services, see Requesting notification of events on page 37.

This API supports these Monitoring Services:

Table 4: Monitoring Services

Services	Description	XSD
Monitor Start	Initiates event reports (otherwise known as events) for a device	monitor-start.xsd
Monitor Stop	Cancels a previously initiated Monitor Start request	monitor-stop.xsd

Avaya extensions

The API provides extensions to CSTA that are meant to enhance the capabilities of CSTA and provide higher-level services and useful events that make development of telephony applications easier. The extensions are summarized in this section. More complete descriptions of each extension can be found in the *Programmer's Reference* (XMLdoc).

The Avaya extensions have been implemented per the CSTA guidelines described in *ECMA-269*, section 28, "Vendor Specific Extensions Services and Events".

The Avaya extensions are listed below and described in the following sections.

Table 5: Avaya extensions to CSTA services

Avaya extension	Extends which CSTA service set	Purpose
Device Services	None	Provides an identifier for a given extension on a given switch
Terminal Services and Events	None	Provides ability to gain exclusive or shared control over switch endpoints - also referred to as *device registration*
Media Control Events	None	Provides the ability to be notified when the far-end RTP and RTCP parameters for a media stream change
Extended Voice Unit Services	Voice Unit Services	Provides dubbing of recorded messages and other extensions
Tone Detection Events	Replaces Data Collection Services	Detects DTMF tones and reports each tone as it is detected
Tone Collection Services and Events	None	Detects DTMF tones and buffers them as requested before reporting them to the application

Device Services

All services that operate on a particular device use a device identifier to specify the device. Avaya's Device Services provide a device identifier for a given extension and switch. There are no events generated by these services.

To learn how to use the Device Services see Getting device identifiers on page 36.

Table 7: Device Services

Services	Descriptions	XSD
Get Device ID	Gets the device identifier that represents the device described by its extension number and the switch upon which it resides	get-device.xsd
Release Device ID	Releases the device identifier and the respective memory resources associated with a device identifier	release-device.xsd

Terminal Services and Events

Avaya's Terminal Services provide a way to gain exclusive or shared control over a device and to specify the desired media mode for that device.

Exclusive control and shared control are described in the "Device and media control" section of the "Capabilities of the API" chapter of the *Overview*. For a list of device types that can be controlled with this API and for further distinction between shared and exclusive control, see the "Controllable telephone types" section of the "Capabilities of the API" chapter of the *Overview*.

The desired media parameters are also specified at registration time. Media options are described in the "Media control modes" section of the "Capabilities of the API" chapter of the *Overview*.

To learn how to use Terminal Services, see Registering devices on page 39.

The Terminal Services are:

Table 8: Terminal Services

Services	Descriptions	XSD
Register Device	Registers a specific device with Communication Manager in order to gain exclusive or shared control over the phone or extension and specifies the desired media mode	register-device.xsd
Unregister Device	Unregisters the specified device from Communication Manager in order to give up control of the device	unregister-device.xsd

Table 9: Terminal Events

Events	Descriptions	XSD
Registered	Occurs when device successfully completes registration	terminal-events.xsd
Register Failed	Occurs when device fails to complete registration	terminal-events.xsd
Unregister	Occurs when device is unregistered either as a result of an Unregister Device request by the application, or an automatic unregister if the Communication Manager switch or the network between the connector server and Communication Manager goes down.	terminal-events.xsd

If a device is registered in client media mode, then the Media Control events described in the following Media Control Events section may also occur.

Media Control Events

Avaya's Media Control events provide a way for an application that is using client media under exclusive control mode to respond to changes in the far-end RTP/RTCP parameters of a media stream. Media Control events include:

Table 10: Media Control Events

Events	Descriptions	XSD
Media Start	Indicates when the far-end RTP parameters have changed and an RTP session has been established	media-events.xsd
Media Stop	Indicates when the far-end RTP parameters have changed to null and the RTP session has been disconnected	media-events.xsd

To learn how to use the media control events, see Determining when far-end RTP media parameters change on page 56.

Extended Voice Unit Services

Avaya's Extended Voice Unit Services are used in conjunction with CSTA's Voice Unit Services.

To learn how to use the Extended Voice Unit Services, see Recording and playing voice media on page 47.

These Extended Voice Unit services are provided:

Table 11: Extended Voice Unit Services

Services	Descriptions	XSD
Start Dubbing	Starts replacing an existing recording session with the specified file	start-dubbing.xsd
Stop Dubbing	Stops replacement of an existing recording session	stop-dubbing.xsd
Stop Playing	Stops only the player, not the recorder	stop-playing.xsd
Stop Recording	Stops only the recorder, not the player	stop-recording.xsd
Suspend Playing	Suspends only the player, not the recorder	suspend-playing.xsd
Suspend Recording	Suspends only the recorder, not the player	suspend-recording.xsd
Resume Playing	Resume playing, but not recording	resume-playing.xsd
Resume Recording	Resumes recording, but not playing	resume-recording.xsd

Tone Detection Events

Avaya's Tone Detection Events notifies an application whenever a DTMF digit has been detected coming into a device. Both in-band and out-of-band tone detection is supported. Out-of-band tone detection is recommended.

To learn how to use Tone Detection Events, see Detecting and collecting DTMF tones on page 53.

When the application requests monitoring for DTMF tones, the following event will be generated when a DTMF has been sent to the device:

Table 12: Tone Detection Events

Events	Description	XSD
Tone Detected	Occurs when a DTMF digit has been sent to the device	tone-detection-events.xsd

Tone Collection Services and Events

Avaya's Tone Collection Services collects DTMF tones coming into a device, stores them in a buffer, and reports the tones based on application-specified retrieval criteria. The retrieval criteria can be one or more of the following:

- The specified number of tones has been detected
- The specified tone has been detected
- The specified amount of time has passed

If multiple criteria are specified, then the first condition that occurs terminates the retrieval and reports the string of DTMF tones collected. Both in-band and out-of-band tone collection are supported. Out-of-band tone collection is recommended.

When tones are retrieved and reported to the application, they are removed from the buffer. If the buffer fills up, the oldest tones are overwritten with the new detected tones.

To learn how to use Tone Collection Services, see Detecting and collecting DTMF tones on page 53.

This API supports these Tone Collection Services:

Table 13: Tone Collection Services

Services	Description	XSD
Start Tone Collection	Starts collecting DTMF tones sent to a device and specifies the termination criteria	tone-collection-start.xsd
Tone Collection Criteria	Specifies the retrieval criteria	tone-collection-criteria.xsd
Stop Tone Collection	Stops collecting DTMF tones sent to a device and reports the tones that have been buffered. This flushes the buffer	tone-collection-stop.xsd
Flush Buffer	Reports the tones received since the last time the buffer was flushed and flushes the buffer	tone-collection-flushbuffer.xsd

Tone Collection Services generates these events:

Table 14: Tone Collection Events

Events	Description	XSD
Tones Retrieved	Occurs when tones are retrieved from the buffer. This event reports the retrieved tones to the application.	tone-collection-events.xsd

Differences between Avaya API and ECMA-269

The Avaya API differs from the ECMA specification in the following ways:

- Voice Unit Services perspective
- Negative acknowledgements

Voice Unit Services perspective

The mechanism for call control in this API is to register an extension with Communication Manager using Terminal Services and then to use Physical Device Services to manipulate that extension. Therefore this API follows a device-based call control model. There are a few subtle side effects of using the device-based control model that are worth noting.

- CSTA specifies that the Voice Unit Play Message service “plays a voice message on a particular connection”. While this is an ambiguous description, the apparent intent was to play a message *to* a particular device, which is a third party perspective. This API’s implementation of the Play Message service is just the opposite of this. This API’s Play Message service plays a message *from* the device, a first party perspective. It plays the message as if coming from the device and going to everyone else on the call.
- Similarly, CSTA specifies that the Voice Unit Record Message service “starts recording a new message from a specified connection.” The apparent intent was to record the data coming *from* the device. This API implementation records the data coming *to* the device. It records what the device hears instead of what someone says at the device.
- Since Avaya’s implementation of Voice Unit Services are relative to a device instead of a connection, only the device identifier portion of a connection identifier is used.

Negative acknowledgements

The CSTA standard uses negative acknowledgements to indicate that an error has occurred. That is, a request can have different responses, either a valid response, or a CSTA Exception. Each service request returns a valid response or a negative acknowledgement to indicate an error occurred. Each negative acknowledgement will contain an exception stack trace for additional information.

Each exception may also contain a number of chained exceptions. The full stack trace of the chained exceptions may be helpful to application developers in understanding the error and in debugging their own code. This stack trace may also be helpful in debugging problems in the server.

2 Getting started

This section describes what you need to do and what you need to know before you begin programming to this API, including:

Setting up the development environment

XML developers must have the necessary tools to traverse an XML message, such as an XML parser.

Downloading the Communication Manager API SDK

The Communication Manager API SDK contains the XSD files that you will need to reference as you write your application, as well as several sample applications.

To download the Communication Manager API SDK from the Avaya Developer's Connection website:

1. Go to www.devconnectprogram.com.
2. Select **Member Login**.
3. Log in with your email address and password.

   The ***Avaya Communication Manager Developers*** page appears.
4. Download the SDK (cmapi-sdk-2_1_*x*.zip).

   The download location defaults to the desktop, but it does not matter where you download the files in your directory system. The SDK file is:

   cmapi-sdk-2_1_*x*.zip

   where *x* is the load number.
5. Expand the SDK ZIP file using any application or tool that recognizes the ZIP file format. All of the SDK files are placed into a directory named cmapisdk.

The directories where the XSD files are:

- **`cmapisdk/xsd/castor`**
- **`cmapisdk/xsd/castor_avaya`**

The location of the primary XSD used to validate the data you will send to the server is:

- **`cmapisdk/xsd/castor_avaya/avaya-csta.xsd`**

The location of the sample applications provided with the SDK is:

- **`cmapisdk/examples/src/samplefiles`**

Setting up your test environment

Before running an application you will need to have a connector server machine setup. See the *Installation and Administration* guide for instructions.

Understanding basic CSTA concepts

CSTA stands for Computer-Supported Telecommunications Applications. It is a standard produced by ECMA, an international standards body (http://www.ecma-international.org). CSTA provides a standard for Computer-Telephony Integration (CTI). When fully implemented, CSTA allows an application to:

- monitor calls on extension lines or trunks
- modify the behavior of calls
- make a call between two parties

The Avaya Communication Manager API implements a subset of CSTA. The API supports monitoring and making calls at the physical device level. Applications using this API have first party device control and media control.

The following sections describe what you need to know about the CSTA concepts of:

- Devices
- Physical elements and Logical elements
- Calls
- Service requests and Service responses
- Events
- Negative acknowledgements

Devices

In the context of this API, a device refers to a software instantiation of a phone or extension that is registered on Communication Manager. Such a device is also referred to as a softphone. A device has physical and logical elements.

Physical elements

The physical element of a device encompasses the set of attributes, features, and services that have any association with physical components of the device that make up the user interface of the device. Physical elements can be manipulated, such as pushing buttons or going offhook, or they can be observed, such as observing the ringer or whether a lamp is lit. The physical elements of a device include:

- Buttons
- Hookswitch
- Display
- Lamps
- Message waiting indicator
- Ringer

This API supports all of these physical elements.

Logical elements

A logical element is the part of a device that is used to manage and interact with calls at a device. This element represents the media stream channels and associated call handling facilities that are used by the device when involved in a call. The logical elements that this API supports are:

- DTMF digits coming into the device
- Media stream coming into and out of the device

Calls

Calls are made from and received by a device using the device's physical and logical elements. CSTA performs most telephony services against a connection that identifies a particular call. In this API telephony services are requested for a device instead.

Request and response framework

Your application will need to be able to:

- make service requests
- parse service responses
- parse exceptions
- monitor for and parse events

This section describes what requests, responses, exceptions and events are and how to use them.

Service requests

In this API an application requests services of the connector server. Examples of a request are “give me a device identifier”, “press a button”, “give me information about a lamp’s status”, or “notify me when a digit comes into the device”.

Each request is processed by the connector server. The connector server may complete a request synchronously in one logical step or it may take additional steps to pass the request to the switch and handle the switch’s response(s) before responding asynchronously to the application with the request’s results in an event or exception.

To make a request, the application must construct the appropriate XML message and send that to the connector server.

Service responses

Each service request is acknowledged with a service response (positive acknowledgement). Some service requests, particularly those that request information, are completed in a single logical step, thus the results of the request are reported in the service response. (Single-step requests follow CSTA’s atomic model.) Other service requests, particularly those that require the connector server to pass on a request to the switch, take multiple steps to complete. (These follow CSTA’s multi-step model.) Responses to multi-step requests merely indicate that the request was received and is being processed.

In some cases, response data values may indicate an error in the request or in the processing of a single-step request. However, most errors are indicated through exceptions (see Negative acknowledgements below) or negative events (see Events below).

Service responses are in the form of XML messages. You must have the necessary tools to traverse an XML message in order to get the information out of the XML message.

Events

Events are asynchronous occurrences on a device that an application can choose to monitor for and respond to. Examples of events are the `DisplayUpdatedEvent,` which indicates that the device’s display has changed, or `RegisteredEvent,` which indicates that the device has successfully been registered.

An application indicates its desire to be notified of events by starting a monitor. Once a monitor is established, the application notifies the connector server of relevant activity by sending messages called event reports, or simply events.

Negative acknowledgements

The connector server may respond to a service request with an exception. The XML message received will have the XML tag `<CSTAException>` present. It will be up to the application as to how it chooses to handle such messages. The application does not need to send back a message to the Communication Manager API server when such a message is received. Review the server logs to make sure that other adverse conditions have not transpired.

Accessing the client API reference documentation

You will need to frequently reference the *XML Programmer's Reference* (XMLdoc) provided with this API. It is available on the DevConnect website (www.devconnectprogram.com) and (www.avaya.com/support) as both a viewable HTML document and a downloadable zip file.

If you choose to download the zip file, then do the following to browse the HTML-based XMLdoc:

1. Expand the ZIP file using any application or tool that recognizes the ZIP file format. All of the XMLdoc files are placed into a directory named XMLdoc
2. Go to the XMLdoc directory.
3. Double click on index.html (or open this file with a browser).

Learning from sample code

Another key learning tool is the set of sample code files that are provided with this API. This sample code is located in the following directory:

cmapisdk/examples/src/samplefiles

- CmapiXML.cs - a C# example
- CmapiXMLCpp.cpp - a C++ .NET example
- ExampleCmapiXML.java - a Java example

Each of the examples simply show how to connect to the CMAPI server and how to send very basic XML messages

3 Writing a client application

This chapter describes how to write an application using the Communication Manager API. It will frequently refer to the details in the XMLdoc, so you may wish to have ready access to the XMLdoc while reading this chapter. Read Accessing the client API reference documentation on page 29 to find out how to get access to the XMLdoc.

Your application may have these different parts:

- Setup
  - The CSTA Header
  - Establish a connection to the connector server
  - Setting up the IO Streams
  - Getting device identifiers
  - Receiving exceptions
  - Getting device identifiers
  - Requesting notification of events
  - Registering devices
- Telephony Logic for performing actions after various events occur
  - Monitoring and controlling physical elements
  - Recording and playing voice media
  - Detecting and collecting DTMF tones
- Cleanup
  - Stop collecting tones
  - Stop recording or playing
  - Unregister the device
  - Stop monitoring for events
  - Release the device identifier

Each part is described in the sections below.

Setup

This section describes the different setup steps that must be taken by every application. Applications using the Communication Manager API XML protocol require some infrastructure pieces to be built. Such facilities are:

- connection to the Communication Manager API server
- socket management
- exception handling
- synchronous and asynchronous message handling and event notification

Creating such facilities are very technology-specific and beyond the scope of this document.

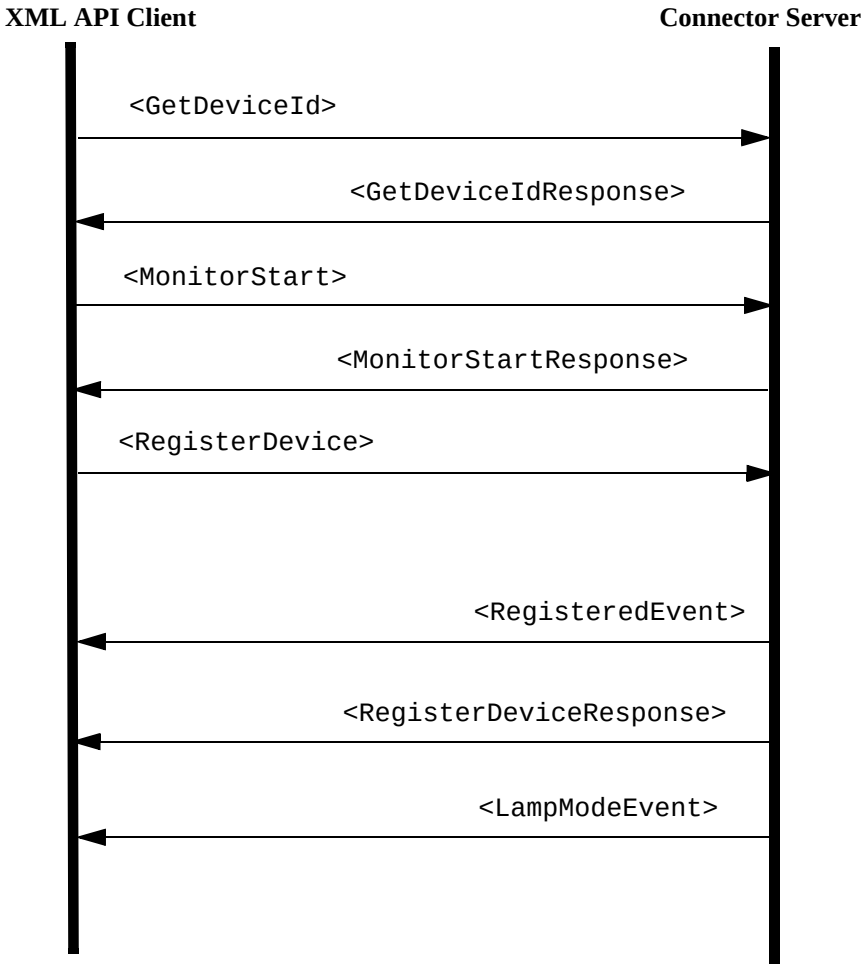
The following sections provide several code fragments to illustrate one implementation. These fragments are provided in C#, C++ and VB and come from various .NET applications. These code samples are for illustration purposes only, and as such, should not be taken as the optimal choice for your application. The following sections provide many sample XML messages that will be sent between the application and the connector server. The content of the messages are for illustration purposes only and will vary depending on your environment and application needs. However, the structure of the message is important to note.

The following figure shows the three specific XML messages that must be constructed by the application and sent to the connector server in order to get started.They are:

- `GetDeviceId`
- `MonitorStart`
- `RegisterDevice`

You must send these messages in the order shown. The figure also shows what XML messages your application will receive from the server in response. You must wait for a response for each of these requests before moving on to the next. After you send the `RegisterDevice` message you will receive the corresponding `RegisterDeviceResponse`, however, it is not guaranteed that the response message will come back in the order expected. At this point the connector can begin to send asynchronous messages. The use of each of these requests is described in further detail in later sections of this chapter.

Figure 1: Required XML API messages



The CSTA Header

Proper construction of the XML messages is very important. All XML messages must have prepended the appropriate header as specified by CSTA. The CSTA header consists of a two byte version field, a two byte length field and a four byte invoke ID as illustrated in the following figure. All data is sent using network byte ordering (most significant byte first).

0 | 1	2 | 3	4 | 5 | 6 | 7	8.....
Version	Length	Invoke ID	XML Message Body

Version

The version indicates the format of the XML instance message. The following header format is defined:

- 00 - indicates that the message body consists of an Invoke ID component followed by an XML instance message.

Length

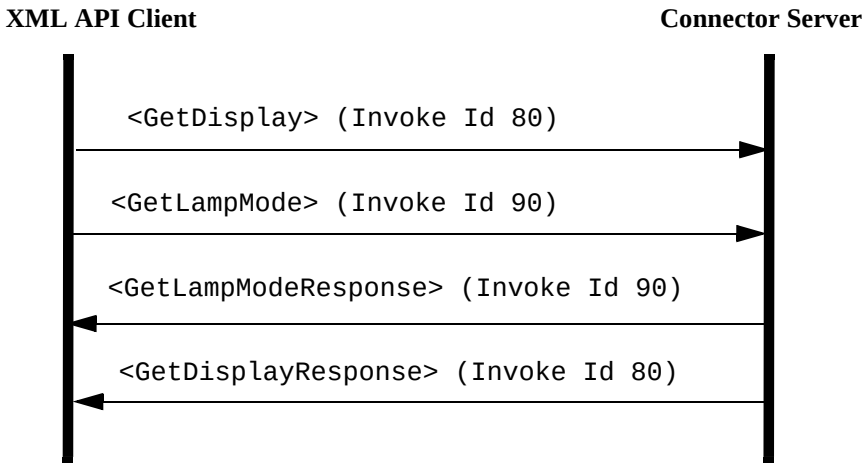
The length is defined as the length of the full message which is the length of the message body plus the length of the message header.

The invoke ID

A four byte Invoke ID is provided in order to correlate a CSTA service request with the corresponding response message. This Invoke ID is provided in the message body that precedes the XML instance message. It is encoded as four ASCII numerical characters. You will need to populate the Invoke Id field in the CSTA header with a unique value for every request you send. That same value will be returned in the Invoke ID field of the corresponding service response message (positive or negative). Responses to requests might not be sent to your application in the same order that the application sent the requests. Use the Invoke ID to correlate each response with the request.

Request and response XML messages must use Invoke IDs between 0 and 9998. A value of "9999" is used by the connector server when sending events to your application.

Figure 2: Example XML message passing



Establish a connection to the connector server

Connect to the connector server using the connector server IP address (that you have assigned) and the port. Communication Manager API uses port 4721 for remote client connections. The port is usually set during configuration of the connector server.

Following is a C# code fragment:

```
string server = "myServer.company.com";
int port = 4721;
...
TcpClient client = new TcpClient(server, port);
NetworkStream stream = client.GetStream();
```

Setting up the IO Streams

Set up the IO streams. Depending on what programming language you are using to develop your application, this step may already be taken care of by the language itself.

Following is a C# code fragment:

```
BinaryWrite writer = new BinaryWriter(stream);
BinaryReader reader = new BinaryReader(stream);
```

Receiving exceptions

Each service request may generate an exception. When the connector server responds with an exception, the XML message sent will have the XML tag <CSTAException> present. It will be up to the application as to how to handle such messages. The application does not need to send back a message to the Communication Manager API server when such a message is received.

An example of the structure of the XML message your application may receive is:

```
<?xml version="1.0" encoding="UTF-8"?>
<CSTAException xmlns="http://www.avaya.com/csta">
<exceptionClass>java.lang.Exception</exceptionClass>
<message>CMAPI Version:2.1.18 Unable to unmarshal &lt;?xml
   version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?
   &gt;&lt;GetDeviceId
   xmlns=&quot;http://www.avaya.com/csta&quot;&gt;&lt;switchName&gt;11
   1.2.33.444&lt;/switchName&gt;&lt;extension&gt;7510&lt;/extension&gt
   ;&lt;/GetDeviceId&gt;
   Possible causes include:
   1. The SDK and Server verions differ
   2. A required field was not populated in the CSTA object
   3. A field was populated with an illegal value in the CSTA object
</message>
<stackTrace>com.avaya.mvcs.proxy.CstaUnmarshallerNode$CstaUnmarshallerP
   rocessorThread.run(CstaUnmarshallerNode.java:180)com.avaya.common.c
   oncurrent.QueuedExecutor$RunLoop.run(QueuedExecutor.java:88)java.la
   ng.Thread.run(Thread.java:534)
</stackTrace>
<exception>rO0ABXNyABNqYXZhLmxhbmcuRXhjZXB0aW9u0P0fPho7HMQCAAB4cgATamF
   2YS5sYW5nLlRocm93YWJsZdXGNSc5d7jLAwADTAAFY2F1c2V0ABVMamF2YS9s
   YW5nL1Rocm93YWJsZTtMAA1kZXRhaWxNZXNzYWdldAASTGphdmEvbGFuZy9Td
```

```
   HJpbmc7WwAKc3RhY2tUcmFjZXQAHltMamF2YS9sYW5nL1N0YWNrVHJhY2VFbG
   VtZW50O3hwc3IAJm9yZy5leG9sYWIuY2FzdG9yLnhtbC5NYXJzaGFsRXh
</exception>
</CSTAException>
```

NOTE:
The exception field is the binary representation of the Java exception class.

We recommend that the application log all possible exceptions since this will be an important source of information for debugging the application. Look at the server side logs for more infomation about what happened. The logs will contain helpful information that you may need to provide to DevConnect.

Getting device identifiers

Set up a device identifier for each Communication Manager phone or extension that your application needs to instantiate as a softphone. The device identifier is made up of:

- IP address of a C-LAN board (CLAN) or the processor C-LAN (PROCR) on the Communication Manager system on which the phone or extension is administered
- extension number of the softphone-enabled station

Construct the `GetDeviceId` XML message and send it to the connector server. The following is the structure of the XML message you send:

```
<?xml version="1.0" encoding="UTF-8"/>
<GetDeviceId xmlns="http://www.avaya.com/csta">
   <switchName>111.2.33.444</switchName>
   <extension>4750</extension>
</GetDeviceId>
```

The Communication Manager API server responds with the `GetDeviceIdResponse` XML message. The application must extract the device identifier from within this message.

The following example shows the structure of the response from which you will extract information:

```
<?xml version="1.0" encoding="UTF-8"/>
<GetDeviceIdResponse xmlns="http://www.avaya.com/csta">
   <device typeOfNumber="other" mediaClass=""
      bitRate="constant">111.2.33.444:4750:0</device>
</GetDeviceIdResponse>
```

NOTE:
The XML tags for the device identifier are not consistent throughout the XML messages. This inconsistency is CSTA-related. In the sample above of the `GetDeviceIdResponse` XML message, the device identifier is bracketed by the <`device`> tags. In the `MonitorStart` XML message, the device identifier is bracketed by the <`deviceObject`> tags.

NOTE:
The `DeviceID` and `ConnectionID` formats are subject to change in future releases. Your applications should not be parsing these IDs, but rather should be using them as keys in their requests.

Requesting notification of events

To receive notification of all changes in the switching function, the application must use a feature called monitoring.

To monitor for certain types of events, an application must establish a monitor using the `MonitorStart` request. By starting a monitor, the application indicates the device that it is interested in observing.

By default, per CSTA standards, an application that sends a `MonitorStart` request will begin receiving all events. You must specifically state what events you do not want to receive.

Avaya has implemented a feature that allows an application to request to receive only certain events. This is done using the `invertFilter` parameter.

Once a monitor is established, the connector server notifies the application of relevant activity by sending messages called event reports, or simply events. For details about this request, see the XMLdoc.

Events for which your application can choose to be notified of include:

- Telephony events. Examples of telephony events are when the lamp state has changed or a DTMF digit has been detected.
- Asynchronous responses to service requests. For example, after requesting to register a device, an event is received indicating whether the request succeeded or failed.

To monitor for certain types of events, an application must construct the `MonitorStart` XML message and send it to the connector server.

Following is a sample `MonitorStart` message. Consult the tables of events in the API Services chapter and the XMLdoc to determine what events an application can request notification for:

```
"<?xml version="1.0" encoding="UTF-8"??>
<MonitorStart xmlns="http://www.ecma.ch/standards/ecma-323/csta/ed2">
  <monitorObject>
     <deviceObject typeOfNumber="other" mediaClass=""
        bitRate="constant">111.2.33.444:4750:0</deviceObject>
  </monitorObject>
  <requestedMonitorFilter/>
  <extensions>
     <privateData>
        <private>
           <ns1:AvayaEvents xsi:type="ns1:AvayaEvents"
              xmlns:ns1="http://www.avaya.com/csta"
                xmlns:xsi="http://www.w3.org/2001/SMLSchema-instance">
              <ns1:invertFilter>true</ns1:invertFilter>
              <ns1:terminal>
                 <ns1:registered>true</ns1:registered>
                 <ns1:unregistered>true</ns1:unregistered>
                 <ns1:registerFailed>true</ns1:registerFailed>
              </ns1:terminal>
           </ns1:AvayaEvents>
        </private>
     </privateData>
  </extensions>
</MonitorStart>"
```

This example `MonitorStart` message requests that the application be notified of the following events:

- `RegisteredEvent`
- `UnregisterEvent`
- `RegisterFailedEvent`

In response to the `MonitorStart` request, the connector server:

- starts a monitor
- allocates a Monitor Cross Reference Identifier that uniquely identifies the monitor
- provides a positive acknowledgement that includes this Monitor Cross Reference Identifier

The structure of the response is:

```
<?xml version="1.0" encoding="UTF-8"/>
<MonitorStartResponse xmlns="http://www.ecma.ch/standards/ecma-
   323/csta/ed2">
   <monitorCrossRefID>83</monitorCrossRefID>
</MonitorStartResponse>
```

After the device is registered, the connector will begin sending any events that the application has indicated that it was interested in receiving. A response from the server with an Invoke ID of 9999 indicates that this is an event. Each event contains the Monitor Cross Reference Identifier that correlates the event back to the Monitor Start service that established the monitor. Look in the body of the message for this Monitor Cross Reference Identifier to find out which `MonitorStart` this event is associated with. It is left to the application to determine how to handle the event messages that are being sent.

These event reports cease after the connector server terminates the monitor. Service termination can result from a client request (using the `MonitorStop` request) or it can be initiated by the server. When an application sends a `MonitorStop` request, the connector server will:

- stop the monitor
- release the Monitor Cross Reference Identifier
- stop providing events to the application

CAUTION:
There may be conditions that cause the server to terminate the monitor. In the future, the `MonitorStop` request may also be sent from the connector server to the application when the connector server stops an existing monitor.

Once the monitor is terminated, the monitor cross reference ID is no longer valid. See *ECMA-269*, section 6.7.2, “Monitoring” for an overview of monitoring and related concepts such as monitor objects, monitor types, monitor call types, and monitor filters.

Registering devices

To monitor or control a device, an application must register the device with Communication Manager. This tells Communication Manager whether you want exclusive or shared control of the device and what kind of media access you want. Only devices that are softphone-enabled on Communication Manager's ***Station*** form can be registered.

The list of phone types that can be registered is shown in the "Controllable telephone types" section of the "Capabilities of the API" chapter of the *Overview.* For details on how to softphone-enable a device, see *Communication Manager API Installation and Administration* or *Communication Manager Administration*.

Registration is performed through the `RegisterDevice` request of Terminal Services. In the `RegisterDevice` request you must at a minimum specify which device to register and the password that is administered for the device on the ***Station*** form. There are also a number of options to choose from.

An application must construct the `RegisterDevice` XML message and send it to the connector server.

```
<?xml version="1.0" encoding="UTF-8"?>
<RegisterDevice xmlns="htt[://www.avaya.com/csta">
   <device typeOfNumber="other" mediaClass=""
      bitRate="constant">111.2.33.444:4750:0</device>
   <loginInfo>
      <forceLogin>true</forceLogin>
      <sharedControl>false</sharedControl>
      <password>1234</password>
   </loginInfo>
   <localMediaInfo>
      <codecs>g729</codecs>
   </localMediaInfo>
</RegisterDevice>
```

Registration status is reflected through events. If an application has requested notification of these events and the registration request succeeds, the application receives a `RegisteredEvent`:

```
<?xml version="1.0" encoding="UTF-8"?>
   <RegisteredEvent xmlns="http://www.ecma.ch/standards/ecma-
      323/csta/ed2">
      <ns1:monitorCrossRefID
         xmlns:ns1="http://www.ecma.ch/standards/ecma-
         323/csta/ed2">50<ns1:monitorCrossRefID>
      <device>
         <ns2:deviceIdentifier typeOfNumber="other" mediaClass=""
            bitRate="constant"
            xmlns:ns2="http://www.ecma.ch/standards/ecma-
              323/csta/ed2">111.2.33.444:4750:0</ns2:deviceIdentifier>
      </device>
   </RegisteredEvent>
```

If registration fails, the application receives a `RegisterFailedEvent`. The reason is reflected in the event's cause code. Its value is one of the Registration Constants listed in Appendix B, "Constant Values" . Even if registration succeeds, it is possible that the connector server may automatically unregister the device and send an `UnregisterEvent`. The typical cause is when the network or Communication Manager is unresponsive. If a device becomes automatically unregistered, it is up to the application to reregister.

As part of the registration initialization process, multiple physical device events are sent by the Communication Manager indicating the initial states of the lamps, ringer, hookswitch and display. These events may occur before and/or after the `RegisteredEvent`.

> **NOTE:**
> If your application needs to register many devices, we recommend you spread out the registration of the devices. Register no more than 50 stations at one time and wait until your application has received all of the registered events before attempting to register more stations.

Some important decisions you will need to make when registering a device include:

- What device control mode to choose
- What media mode to choose
- What codecs to choose

Choosing a device control mode

At registration time, the application specifies the desired device control mode. This indicates who controls the device's physical elements and media. The control choices are:

- **Exclusive control mode**

  gives all control of the device to the application including control of the media stream. This must be used if the application will use Voice Unit Services, Tone Detection Services, or Tone Collection Services

  The necessary XML message fragment needs to be in the `RegisterDevice` XML message.

  ```
  <sharedControl>false</sharedControl>
  ```

  See the `RegisterDevice` example for context.

- **Shared control mode**

  gives control to both the telephone and the application. This must be used by applications that need to monitor and control a physical telephone.

  The necessary XML message fragment needs to be in the `RegisterDevice` XML message.

  ```
  <sharedControl>true</sharedControl>
  ```

  See the `RegisterDevice` example for context.

The differences between exclusive and shared device control are more thoroughly described in the "Device and media control" section of the "Capabilities of the API" chapter of the *Overview.*

Choosing a media mode

If the application registers the device with shared control, then the media is handled only by the telephone. However, if the application requests exclusive control, then the application can choose where the media is terminated and who handles the media. If they request telecommuter mode the application can choose what physical phone will receive and handle the media.

The following table shows the media modes available, when to use each, and how to request each:

Table 16: Choosing a media mode when using exclusive device control

Exclusive-control media modes	When to use	How to set
Server media mode	When an application wants the connector server to handle media processing or if no media processing is needed at all. Connector server handles media with Voice Unit Services and with Tone Detection Services or Tone Collection Services when detecting in-band DTMF digits. In this mode, media stream terminates on connector server. Requesting to be notified of the `MediaStopEvent` and the `MediaStartEvent` may also be used in this mode to detect changes to the far-end RTP/RTCP parameters.	If you want different codecs than the default, then set just the codecs in `LocalMediaInfo`, but do *not* set the RTP/RTCP address in `LocalMediaInfo`. Otherwise, do not set anything in `LocalMediaInfo`.
Client media mode	When an application wants to process the media itself. RTP stream can be terminated on any desired machine. Requesting to be notified of the `MediaStopEvent` and the `MediaStartEvent` may also be used in this mode to detect changes to the far-end RTP/RTCP parameters. In this mode Tone Detection Services and Tone Collection Services can also be used to detect out-of-band DTMF digits.	Set `LocalMediaInfo`'s RTP/RTCP address to IP address and port where the media stream is to be terminated. May also set preferred codecs.
Telecommuter media	When an application wants the media to go to a real telephone. The real telephone can be inside or outside of the Communication Manager domain	Leave out `LocalMediaInfo` and just set `LoginInfo`'s `telecommuterExtension` to the telephone number of the real telephone that you are directing the media to.

Following are two examples of how to choose exclusive device control and server media mode:

```
<?xml version="1.0" encoding="UTF-8"?>
<RegisterDevice xmlns="htt[://www.avaya.com/csta">
   <device typeOfNumber="other" mediaClass=""
      bitRate="constant">111.2.33.444:4750:0</device>
   <loginInfo>
      <forceLogin>true</forceLogin>
      <sharedControl>false</sharedControl>
      <password>1234</password>
   </loginInfo>
   <localMediaInfo>
      <codecs>g729</codecs>
   </localMediaInfo>
</RegisterDevice>
```

or

```
<?xml version="1.0" encoding="UTF-8"?>
<RegisterDevice xmlns="htt[://www.avaya.com/csta">
```

```
    <device typeOfNumber="other" mediaClass=""
       bitRate="constant">111.2.33.444:4750:0</device>
    <loginInfo>
       <forceLogin>true</forceLogin>
       <sharedControl>false</sharedControl>
       <password>1234</password>
    </loginInfo>
    </RegisterDevice>
```

Following is an example of how to choose exclusive device control and client media mode:

```
<?xml version="1.0" encoding="UTF-8"?>
<RegisterDevice xmlns="htt[://www.avaya.com/csta">
    <device typeOfNumber="other" mediaClass=""
       bitRate="constant">111.2.33.444:4750:0</device>
    <loginInfo>
       <forceLogin>true</forceLogin>
       <sharedControl>false</sharedControl>
       <password>1234</password>
    </loginInfo>
    <localMediaInfo>
       <rtpAddress>111.2.33.444</rtpAddress>
       <rtcpAddress>4750</rtcpAddress>
    </localMediaInfo>
</RegisterDevice>
```

Following is an example of how to choose exclusive device control and telecommuter media:

```
<?xml version="1.0" encoding="UTF-8"?>
<RegisterDevice xmlns="htt[://www.avaya.com/csta">
    <device typeOfNumber="other" mediaClass=""
       bitRate="constant">111.2.33.444:4750:0</device>
    <loginInfo>
       <forceLogin>true</forceLogin>
       <sharedControl>false</sharedControl>
       <password>1234</password>
       <telecommuterExtension>3035551234</telecommuterExtension>
    </loginInfo>
</RegisterDevice>
```

Choosing a codec

A codec is the algorithm used to encode and decode audio media. For devices being registered in exclusive control mode, the application can optionally specify at registration time what codecs are preferred for the device. The codec options are:

- G.711 A-law (g711A)
- G.711 Mu-law (g711U)
- G.729 (g729)
- G.729 Annex A (g729A)

Specify the set of codecs your application supports using the `RegisterDevice` request as shown below. Note that you cannot specify a mixture of G.711 and G.729 codecs for a single device. If you do not specify a set of codecs in the `RegisterDevice` request, the connector server will default to G.711 A-law as the first choice and G.711 Mu-law as the second choice. If Communication Manager cannot satisfy your request for specific codecs, then calls will still go through, but there will be no media.

The necessary XML message fragment needs to be in the `RegisterDevice` XML message.

```
<localMediaInfo>
<codecs>g729</codecs>
</localMediaInfo>
```

See the `RegisterDevice` example for context.

For more information about selecting and administering network regions and their codecs, see the "Administering Communication Manager" chapter of *Installation and Administration.*

Telephony Logic

The connector server notifies your application when requested telephony events occur:

- Terminal events indicate registration status.
- Physical Device events indicate changes to the status of the ringer, display, and lamps.
- Media Control events indicate when the media stream parameters have changed.
- Voice Unit events indicate the status of recording and playing messages.
- Tone Detection events indicate when a DTMF digit is received and when collection begins and ends.
- Tone Collection events indicate when specified tone retrieval criteria are met.

Your application's telephony logic usually begins upon receiving the `RegisteredEvent`.

Monitoring and controlling physical elements

Physical elements of a device are monitored and controlled with Physical Device Services. The set of supported services and events are reflected in Physical Device Services and Events on page 16.

To monitor for physical device events, you must request notification of these events through the `MonitorStart` XML message as described in Requesting notification of events on page 37.

Call control can be accomplished with a combination of:

- determining the current status of physical elements on a device, such as requesting the list of buttons administered for the device
- monitoring for particular physical device events, such as when the phone starts ringing
- activating physical elements of the device, such as going offhook

Knowing what buttons are administered

If your application needs to press any buttons or determine which lamps have changed state, you will need to know what buttons are administered on the device. Buttons are assigned to devices during station administration via the Communication Manager system access terminal (SAT) interface. Your application must send the `GetButtonInformation` request in order to the get the list of buttons administered for a device. Each button item in the list includes the following information:

- *button identifier* - indicates the address or location of the button. Its value is one of the Button ID Constants listed in Appendix B, “Constant Values” . Constants are available for both administered buttons and fixed buttons (those buttons that are preset and pre-labeled on a telephone set).
- *button function* - indicates what the button does when pressed. Its value is one of the Button Function Constants listed in Appendix B, “Constant Values” .
- *associated extension* - indicates whether there is an extension number associated with this button and what the extension number is.
- *associated lamp* - indicates whether there is a lamp associated with the button. If there is, its lamp identifier is the same as the button identifier.

Detecting an incoming call

When a call comes into a device, these three changes occur to the physical device:

- The phone rings.
- A green call appearance lamp flashes.
- The display changes to show caller information.

The following events are sent to an application that requested notification of these events:

- `RingerStatusEvent` - The ring pattern is supplied in the event structure. For a list of possible ring patterns see Appendix B, “Constant Values” for Ringer Pattern Constants. Following is an example of the structure of the `RingerStatusEvent` that a client application may receive. Since this message is an asynchronous message sent by the server there is a MonitorCrossRefID which is very important for correlating the message to the `MonitorStart`.

```
<?xml version="1.0" encoding="UTF-8"?>
<RingerStatusEvent xmlns="http://www.ecma.ch/standards/ecma-
   323/csta/ed2">
   <monitorCrossRefID>77</monitorCrossRefID>
   <device>
      <deviceIdentifier typeOfNumber="other" mediaClass=""
         bitRate="constant">111.2.33.444:4750:0</deviceIdentifier>
   </device>
   <ringer>0000</ringer>
   <ringMode>ringing</ringMode>
   <ringPattern>11</ringPattern>
</RingerStatusEvent>
```

> **NOTE:**
> Per CSTA specification the device identifier for the `RingerStatusEvent` XML message uses <`deviceIdentifier`> tags.

- `LampModeEvent` - The identifier of the lamp that has changed and the lamp’s mode is supplied in the event structure. Once you know where the call appearance lamps are (see Knowing what buttons are administered on page 43), you can determine if it is a call appearance lamp and if it is flashing by comparing the lamp mode against the `FLASH` constant. For a complete list of the possible lamp modes, see Appendix B, “Constant Values” for Lamp Mode Constants. Following is an example of the structure of the `LampModeEvent` that a client application may receive.

```
<?xml version="1.0" encoding="UTF-8"?>
<LampModeEvent
   xmlns="http://www.ecma.ch/standards/ecma-       323/csta/ed2">
  <monitorCrossRefID>2</monitorCrossRefID>
```

```
  <device>
       <deviceIdentifier typeOfNumber="other" mediaClass=""
         bitRate="constant">111.2.33.444:7512:0</deviceIdentifier>
  </device>
  <lamp>263</lamp>
  <lampMode>3</lampMode>
  <lampBrightness>unspecified</lampBrightness>
  <lampColor>1</lampColor>
</LampModeEvent>
```

NOTE:
Per CSTA specification the device identifier for the `LampModeEvent` XML message uses <`deviceIdentifier`> tags.

- `DisplayUpdatedEvent` - The display contents are supplied in the event structure. In general the number of complete `DisplayUpdatedEvent` messages you will receive before the diplay update is complete can vary. For the 4624 IP telephone there will be three such messages corresponding to the same `monitorCrossRefID`. This is specific to Communication Manager. The application will have to discard all but the last `DisplayUpdatedEvent` messages. Following is an example of the structure of the `DisplayUpdatedEvent` that a client application may receive.

```
<?xml version="1.0" encoding="UTF-8"?>
<DisplayUpdatedEvent xmlns="http://www.ecma.ch/standards/ecma-
   323/csta/ed2">
   <monitorCrossRefID>77</monitorCrossRefID>
   <device>
      <deviceIdentifier typeOfNumber="other" mediaClass=""
         bitRate="constant">111.2.33.444:4750:0</deviceIdentifier>
   </device>
   <logicalRows>1</logicalRows>
   <logicalColumns>48</logicalColumns>
   <contentsOfDisplay>a=EXT 4750 to EXT 4700 so </contentsOfDisplay>
</DisplayUpdatedEvent>
```

NOTE:
Per CSTA specification the device identifier for the `DisplayUpdatedEvent` XML message uses <`deviceIdentifier`> tags.

Your application may want to key off of just the `RingerStatusEvent`, or it may want to wait for the other events before responding to an incoming call. The events might come in any order.

Determining that the far end has ended the call

If all far-end parties drop on a call, these changes occur on the local device:

- The call appearance green lamp turns off.
- The display returns to its original state before the call arrived.

The following events are sent to an application that requested notification of these events:

- `LampModeEvent` - The identifier of the lamp that has changed and the lamp's mode is supplied in the event structure. Once you know where the call appearance lamps are (see Knowing what buttons are administered on page 43), you can determine if it is a call appearance lamp and if the lamp is now off by comparing the lamp mode against the *OFF* constant. A complete list of possible lamp modes can be found in Appendix B, "Constant Values" under Lamp Mode Constants.

  NOTE:
  Communication Manager sends lamp updates not only for lamp transitions, but also to refresh lamps. Therefore, some `LampModeEvents` indicate that the lamp is in the same state it was in before the event.

- `DisplayUpdatedEvent` - The display contents are supplied in the event structure.

Making a call

To make a call from a telephone, a person would typically:

1. Go offhook
2. Press a sequence of dial pad buttons (0-9, *, #) to initiate a call, such as pressing 5551234
3. Listen for an answer
4. Begin a two-way conversation or listen to a recording

Here is how you might program each of those steps:

1. To go offhook, you could simply send the `SetHookswitchStatus` request. However, this approach could cause a conflict with a potential incoming call. That is, if a call came in just before you went offhook, then your dialing attempt would fail and instead you would be connected with the incoming caller.

   To avoid conflicting with an incoming call, keep track of the lamp transitions of the call appearances. If a lamp goes from off to steady, then you can make an outbound call. But if the lamp goes from off to flashing and then to steady, then you have just picked up an incoming call.

   One method to reduce the chance of conflicting with an incoming call is to begin a call by pressing the *last* call appearance button using the `ButtonPress` request. This avoids using the same call appearance as an incoming call which comes in to the first available call appearance. To further assure that even the last call appearance is not in use, make sure the lamp is off before you press the button.

2. To press the dial pad buttons to dial a number, send the `ButtonPress` request and the constants defined for dial pad buttons in Appendix B, "Constant Values" for Button ID Constants, such as *Dial Pad 7* or *Dial Pad #*.

   Tip:
   Dial Pad 0 through *Dial Pad 9* have string values of "0" through "9", therefore using the strings "0" - "9" will work. However, *Dial Pad* * and *Dial Pad #* are *not* set to "*" and "#"; instead they are "10" and "11", respectively (as specified by CSTA). The `ButtonPress` request will *not* work with "*" and "#". Therefore, it is safer to get in the habit of using the constants, not the literals.

An application must construct the `ButtonPress` XML message and send it to the connector server.

```
<?xml version="1.0" encoding="UTF-8"?>
<ButtonPress xmlns="http://www.ecma.ch/standards/ecma-
      323/csta/ed2">
   <device typeOfNumber="other' mediaClass=""
      bitRate="constant">111.2.33.444:4750:0</device>
   <button>7</button>
</ButtonPress>
```

Below is the corresponding response:

```
</xml version="1.0" encoding="UTF-8"?>
<ButtonPressResponse xmlns="http://www.ecma.ch/standards/ecma-
   323/csta/ed2"/>
```

3. If your application is handling its own media (as determined by the local RTP address set at registration time) then you can determine from the media stream that there is media other than ringback coming from the far end. Your application will need an RTP stack and a call progress tone detector. For the RTP stack, you may use a third-party vendor stack, your own stack or the media stack provided by Avaya. Avaya's media stack is described in the Media Stack API Javadoc. Avaya does not provide a call progress tone detector.

   If the connector server is handling the media, you should wait an appropriate period of time before playing a message to the far end or recording the far end's media stream. This is to allow time for the RTP connection to be made end-to-end.

4. To have a real-time conversation, the application must handle the media.

   To play a message to the far end or record the far end's media stream, use Voice Unit Services (see Recording and playing voice media on page 47 for more details).

Recording and playing voice media

If your application needs to record incoming media or play a message on a call, then at registration time:

- You must request exclusive control of the device.
- You can choose to handle the media yourself (client media) or have the connector server do it for you (server media). See Choosing a media mode on page 40.

This section describes how to use Voice Unit Services to have the connector server record and play media for you. The supported services are described in Voice Unit Services and Events on page 17. The details about using the services are described in the XMLdoc.

Some basic rules of Voice Unit Services are:

- **Wave files**

  All digital audio files that are created or played using Voice Unit Services are in the Wave Resource Interchange File Format (RIFF). The standard Wave file structure is used for all encoded media types. See http://www.sonicspot.com/guide/wavefiles.html for a description of the Wave structure.

  G.729 formatted files, however, use non-standard field values in some of the standard format chunk fields. They are:

  — The compression code value is 131 (0x0083).

  — The block align value is 10 (0x000A).

  — The bits per sample value is 1 (0x0001).

  An external G.729 converter is required to convert a G.729 Wave file into a standard RIFF Wave file that can be played.

- **Files on connector server**

  All Wave files are assumed to be on the connector server machine in the directory specified in:

  cmapi-user-configuration.properties

  by the properties called:

  `com.avaya.mvcs.media.VoiceUnitServices.dir.player` *`dir`*

  `com.avaya.mvcs.media.VoiceUnitServices.dir.recorder` *`dir`*

  NOTE:
  These two directories do not have to be the same. These directories are the root directory of the relative paths specified in the `PlayMessage` and `RecordMessage` requests.

- **Encoding algorithms**

  Files to be played can be encoded in these formats:

  — PCM 8 bit or 16 bit

  — G.711 A-law

  — G.711 Mu-law

  — G.729

  — G.729A

  Recordings can be made of calls in these formats

  — PCM 8 bit or 16 bit

  — G.711 A-law

  — G.711 Mu-law

  — G.729

  — G.729A.

- **Conversions between encoding algorithms**

  Files to be played can be converted from any PCM type to any G.711. No other conversions are supported for playing.

  Messages to be recorded can be converted from G.711 to PCM. No other conversions are supported for recording.

- **Using with Tone Detection or Tone Collection Services**

  The Voice Unit Services player and recorder may be setup to detect DTMF tones at the same time Tone Detection or Tone Collection Services is being used. However, there is no guarantee which service will detect a tone first. See Possible race conditions on page 61 for more specifics.

Recording

To record the RTP media stream of a device, send the Voice Unit Services `RecordMessage` request. This request records only the media coming from other parties on the call to the device; not the media that is being played from this device using the Voice Unit Services `PlayMessage` request. Only media packets that are received are recorded; lost packets are not replaced.

The application can specify an alphanumeric filename for the recording or let the filename default to a format of <*timestamp*><*extension*>.wav. The alphanumeric filename may contain a relative directory path. If it is defaulted, then the resulting filename is returned in the `RecordMessageResponse message.`

Since recording is associated with a device rather than a call, a recording could contain the incoming media from multiple calls. Recording continues until one of the following occurs:

- Application explicitly stops the recording by sending a Voice Unit Services `Stop` request (stops both playing and recording) or by sending an Extended Voice Unit Services `StopRecording` request.
- Application requests that the connector server automatically stop the recording when a specified termination criterion is met. Multiple termination criteria can be specified in which case the first criterion that is met stops the recording. Termination criteria options include:
  - When a DTMF digit is received by the device. To request this termination criterion, set the `termination` parameter's terminating conditions so that `DTMFDigitDetected` is set to true.
  - When recording reaches a specified duration. To request this termination criterion, set the `maxDuration` parameter to the maximum number of milliseconds allowed for the recording.

If you wish to record one entire call and only one call, then your application can monitor the lamp events to determine when the call has ended and explicitly stop the recording after the call has ended.

> **NOTE:**
> No exception is received if the application requests that recording stop when there is no active recording.

Once an active recording on a device has been stopped, a `StopEvent` sent to the application indicates that the recording has finished and the recorded file is ready for the application.

The recording can also be:

- suspended temporarily - with the Voice Unit Services `Suspend` request (suspends both recording and playing) or with the Extended Voice Unit Services `SuspendRecording` request.
- dubbed with another recording - with the Extended Voice Unit Service `StartDubbing` and `StopDubbing` requests. See next section for more information on dubbing.
- resumed - with the Voice Unit Services `Resume` request (resumes both recording and playing) or with the Extended Voice Unit Services `ResumeRecording` request.
- stopped - with the Voice Unit Service`s Stop` request (stops both recording and playing) or the Extended Voice Unit Services `StopRecording` request.
- deleted - with the Voice Unit Services `DeleteMessage` request.

To record a message an application must construct the `RecordMessage` XML message and send it to the connector server.

```
<?xml version="1.0" encoding="UTF-8"?>
<RecordMessage xmlns="http://www.ecma.ch/standards/ecma-
            323/csta/ed2">
   <callToBeRecorded>
      <deviceID typeOfNumber="other" mediaClass=""
         bitRate="constant">111.2.33.444:4750:0</deviceID>
   </callToBeRecorded>
</RecordMessage>
```

Below is the corresponding response:

```
<?xml version="1.0" encoding="UTF-8"?>
<RecordMessageResponse xmlns="http://www.ecma.ch/standards/ecma-
   323/csta/ed2"/>
  <resultingMessage>200403281732075475O</resultingMessage>
  <extensions>
    <privateData>
      <private>
      <ns1:RecordMessageResponsePrivateData
         xsi:type="ns1:RecordMessageResponsePrivateData"
        xmlns:ns1="http://www.avaya.com/csta"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <ns1:timestamp>1080520327543</ns1:timestamp>
           <ns1:filename>200403281732074750</ns1:filename>
        <ns1:RecordMessageResponsePrivateData>
      </private>
    </privateData>
  </extensions>
</RecordMessageResponse>
```

NOTE:
Per CSTA specification the device identifier for the `RecordMessage` XML message uses the tag <`deviceID`>

Dubbing

A recording can be dubbed with another Wave file by sending the Extended Voice Unit Services `StartDubbing` and `StopDubbing` requests. Dubbing records the specified Wave file over the recording repeatedly from the time the `StartDubbing` request is sent until the `StopDubbing` request is sent. This may be helpful to avoid recording sensitive information such as a spoken password or other private or security-based information.

Since the application must explicitly stop the dubbing, the application must have the logic to know when to stop. It may be based on time, or an incoming DTMF digit such as "#", or a manual action by an agent who is monitoring events.

Playing

To play one or more messages to the RTP stream of a call as if the message(s) are coming from the device, send the Voice Unit Services `PlayMessage` request. The message(s) can be played once, multiple times, for a particular duration or until a DTMF digit is received by the device.

The filename of the file to be played can be alphanumeric. The alphanumeric filename may contain a relative directory path. One or more files can be specified as long as they are of the same encoding type.

To play a message an application must construct the `PlayMessage` XML message and send it to the connector server.

```
<?xml version="1.0" encoding="UTF-8"?>
<PlayMessage xmlns="http://www.ecma.ch/standards/ecma-
            323/csta/ed2">
   <messageToBePlayed>111</messageToBePlayed>
   <overConnection>
      <deviceID typeOfNumber="other" mediaClass=""
         bitRate="constant">111.2.33.444:4750:0</deviceID>
   </overConnection>
</PlayMessage>
```

Below is the corresponding response:

```
<?xml version="1.0" encoding="UTF-8"?>
<PlayMessageResponse xmlns="http://www.ecma.ch/standards/ecma-
   323/csta/ed2"/>
```

Since playing is associated with a device rather than a call, the playing of the message(s) may continue across multiple calls to which the device is a party. Playing continues until one of the following occurs:

- Application explicitly stops the playing with a Voice Unit Services `Stop` request (stops both playing and recording) or with an Extended Voice Unit Services `StopPlaying` request.
- A specified termination criterion is met. Multiple termination criteria can be specified in which case the first criterion that is met stops the playing. Termination criteria options include:
  - — When a DTMF digit is received by the device. To request this termination criterion, set the `termination` parameter's terminating conditions so that `DTMFDigitDetected` is set to true.
  - — When playing occurs for a specified duration. To request this termination criterion, set the `duration` parameter to the maximum number of milliseconds allowed for the playing.
  - — When the played message(s) have been repeated a specified number of times with a specified interval in between. To request this termination criterion, set `playCount` and `playInterval` in the `extensions` parameter.

If you wish to play message(s) to one entire call and only one call, then your application can watch the lamp events to determine when the call has ended and explicitly stop the playing after the call has ended.

Once active playing on a device has stopped, a `StopEvent` indicates that the playing has finished.

> **NOTE:**
> No exception is returned and no event is generated if the application requests that playing stop when there is no active playing.

The playing of the message can also be:

- suspended temporarily- with the Voice Unit Services `Suspend` request (suspends both recording and playing) or the Extended Voice Unit Services `SuspendPlaying` request
- resumed - with the Voice Unit Services `Resume` request (resumes both recording and playing) or the Extended Voice Unit Services `ResumePlaying` request
- stopped - with the VoiceUnitServices `Stop` request (stops both recording and playing) or the Extended Voice Unit Services `StopPlaying` request

Monitoring Voice Unit Events

Your application can receive and respond to Voice Unit events by requesting to be notified of those events through the `MonitorStart` XML message as described in Requesting notification of events on page 37. The events indicate when your Voice Unit Service requests have been accepted and processing has begun, and when processing has ended.

The following example shows the structure of the asynchronous message `RecordEvent` that a client application may receive. The `monitorCrossRefID` is to be used by the application to correlate messages.

```
<?xml version="1.0" encoding="UTF-8"?>
<RecordEvent xmlns="http://www.ecma.ch/standards/ecma-323/csta/ed2">
   <monitorCrossRefID>182</monitorCrossRefID>
   <connection>
      <deviceID typeOfNumber="other" mediaClass=""
         bitRate="constant">111.2.33.444:4750:0</deviceID>
   </connection>
   <message>200404091119464750</message>
   <length>0</length>
   <currentPosition>0</currentPosition>
   <cause>normal</cause>
   <extensions>
      <privateDate>
         <private>
            <ns1:RecordMessagePrivateData
               xsi:type="ns1:RecordMessagePrivateData"
               xmlns:ns1="http://www.avaya.com/csta"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
              <ns1:filename>200404091119464750.wav</ns1:filename>
            <ns1:timestamp>1081531186301</ns1:timestamp>
            </ns1:RecordMessagePrivateData>
         </private>
      </privateData>
   </extensions>
</RecordEvent>
```

NOTE:
Per CSTA specifications the device identifier tag in the above example of the `RecordEvent` XML message is <`deviceID`>.

Detecting and collecting DTMF tones

If your application needs to detect DTMF tones coming to a device from another party on the call, then you can use Tone Detection Services or Tone Collection Services. To use these services, you must do both of the following:

- Register the device in exclusive control mode.
- Register the device in server media mode to detect both out-of-band and in-band tones or in client media mode to detect only out-of-band tones.

DTMF tones are generated by parties on a call by pressing the dial pad digits 0 through 9 and * and # during the call. If the device that is being monitored is on a call and another party on the call presses a dial pad digit, then Tone Detection Services can be used to report each DTMF tone to the application.

In contrast, Tone Collection Services can be used to buffer the received tones and report them to the application when the specified retrieval criteria are met. The retrieval criteria may be one or more of the following:

- a specified number of tones has been detected
- a specified tone has been detected (called a “flush character”)
- a specified amount of time has elapsed (called a “timeout interval”)

When at least one of the retrieval criteria is met, the following retrieval steps are performed by the connector server:

1. The buffered tones, up to and including the tone which met the retrieval criteria, are removed from the buffer.
2. The retrieval criteria are cleared (optional).
3. The application is notified of the retrieved tones with the `TonesRetrievedEvent`.

If more than one retrieval criterion is specified, the first one to occur causes the retrieval criteria to be met.

The supported services are summarized in the Tone Detection Events and the Tone Collection Services and Events on page 23. The details about using the services are described in the XMLdoc.

Some basic rules of these services are:

- **Touch tone detection mode**

  Both sets of services can detect in-band and out-of-band DTMF tones. In-band tones are transmitted within the media stream. Out-of-band tones are transmitted in the signalling channel. The connector server always detects out-of-band tones. If the application wishes to also detect in-band tones (not recommended), then the tone detection mode must be explicitly set to in-band.

  Set the tone detection mode before the connector server is started up. The mode is set in the file called:

  cmapi-user-configuration.properties

  with the property called:

  `com.avaya.mvcs.media.ToneDetectionServices.ttd_mode`

  To detect only out-of-band, set the mode to `OUT_BAND`. To detect both in-band and out-of-band, set to `IN_BAND`. See the *Installation and Administration* guide for instructions of how to choose between in-band and out-of-band for the connector server and Communication Manager and how to setup the `ttd_mode` property.

- **Using with Voice Unit Services tone detection**

  The Voice Unit Services player and recorder may be setup to detect DTMF tones at the same time Tone Detection or Tone Collection Services is being used. However, there is no guarantee which service will detect a tone first. See Possible race conditions on page 61 for more specifics.

Detecting individual tones

To detect tones one at a time:

- Request to be notified of the `ToneDetectedEvent` through the `MonitorStart` XML message as described in Requesting notification of events on page 37.
- Now each time a tone is detected by the connector server, the application will receive a `ToneDetectedEvent`.
- When you no longer wish to be notified of detected tones, send a `MonitorStop` request.

Collecting multiple tones

To have the connector server collect multiple tones and report them to the application based on specified retrieval criteria, send the `ToneCollectionServices` requests in this order:

1. Request to be notified of the `TonesRetrievedEvent` through the `MonitorStart` XML message as described in Requesting notification of events on page 37.

2. Start tone collection by sending the Tone Collection Services `ToneCollectionStart` request. This causes each detected tone to be put in a buffer. Following is an example `ToneCollectionStart` XML message:

```
<?xml version="1.0" encoding="UTF-8"?>
<ToneCollectionStart xmlns="http://www.avaya.com/csta">
    <object>
      <ns1:call xmlns:ns1="http://ww.ecma.ch/standards/ecma-
        323/csta/ed2">
          <ns1:deviceID typeOfNumber="other" mediaClass=""
            bitRate="constant">111.2.33.444:4750:0</ns1:deviceID>
      </ns1:call>
    </object>
</ToneCollectionStart>
```

   Following is the corresponding response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ToneCollectionStartResponse xmlns="http://www.avaya.com/csta"/>
```

3. Set the retrieval criteria with the Tone Collection Services `ToneRetrievalCriteria` request. Following is an example `ToneRetrievalCriteria` XML message.

```
<?xml version="1.0" encoding="UTF-8"?>
<ToneRetrievalCriteria xmlns="http://www.avaya.com/csta">
    <object>
      <ns1:call xmlns:ns1="http://ww.ecma.ch/standards/ecma-
        323/csta/ed2">
          <ns1:deviceID typeOfNumber="other" mediaClass=""
            bitRate="constant">111.2.33.444:4750:0</ns1:deviceID>
      </ns1:call>
    </object>
    <numberOfTones>5</numberOfTones>
</ToneRetrievalCriteria>
```

   Following is the corresponding response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ToneRetrievalCriteriaResponse
xmlns="http://www.avaya.com/csta"/>
```

When one of the retrieval criteria is met, you will receive the `TonesRetrievedEvent`. This event will contain the cause of the event. The cause of the event may be any of the following:

- BUFFERFLUSHED - when the `ToneCollectionFlushBuffer` request is sent
- CHARCOUNTRECEIVED - when the number of tones specified in the retrieval criteria is received
- FLUSHCHARRECEIVED - when the tone specified in the retrieval criteria is received
- TIMEOUT - when the amount of time specified in the retrieval criteria has elapsed

If you wish to be notified of more tones, send the Tone Collection Services `ToneRetrievalCriteria` request again. There is no need to stop and restart the collection.

You can start and stop monitors at any time during collection.

If for any reason you wish to flush the buffer during collection, send the Tone Collection Services `ToneCollectionFlushBuffer` request. Your application will receive a `TonesRetrievedEvent` with a cause of BUFFERFLUSHED as a result. This will report the tones collected since the last time the buffer was flushed. You may want to flush the buffer at the end of a call. Following is an example `ToneCollectionFlushBuffer` XML message.

```
<?xml version="1.0" encoding="UTF-8"?>
<ToneCollectionFlushBuffer xmlns="http://www.avaya.com/csta">
   <object>
     <ns1:call xmlns:ns1="http://ww.ecma.ch/standards/ecma-
       323/csta/ed2">
         <ns1:deviceID typeOfNumber="other" mediaClass=""
           bitRate="constant">111.2.33.444:4750:0</ns1:deviceID>
     </ns1:call>
   </object>
   <sendEvent>false</sendEvent>
   <clearCriteria>true</clearCriteria>
</ToneCollectionFlushBuffer>
```

When you no longer wish to be notified of collected tones, stop tone collection with the Tone Collection Services `StopToneCollection` request and send a `MonitorStop` request. This will cause the server to stop collecting DTMF tones sent to a device and report the tones that have been buffered. This flushes the buffer.

You may want to set up interdigit timers limiting the maximum amount of time you will wait between tones or a duration timer limiting the maximum amount of time before stopping the tone detection

Determining when far-end RTP media parameters change

To determine when the far-end RTP media parameters change, applications that control their own media (client media mode) will need to request to be notified of the following events:

- `MediaStartEvent`
- `MediaStopEvent`

Here is the sequence of media control events an application should be prepared to receive:

1. When media is first established for a call, the application receives a `MediaStartEvent`. However, it does not guarantee that the call has been established end-to-end yet.
2. If the switch changes the far-end RTP parameters for a call, the application receives a `MediaStopEvent`. At that point the current far-end RTP parameters should no longer be used. A `MediaStopEvent` could indicate that the call has ended, but do not depend on that event alone to determine the end of a call; the call appearance lamp will also change if it is the end of a call.
3. If there are new far-end RTP parameters, then the application will subsequently receive a `MediaStartEvent.`

One scenario in which a `MediaStopEvent` and then a `MediaStartEvent` may be received is when the switch *shuffles* a call. Shuffling occurs when the switch changes the path of the media. For example, if the media is going from calling party A to the switch and then to called party B, the switch may choose to change the media path such that it goes directly between endpoints A and B. In this case, the switch would tell both A and B to stop using the switch address as the far-end address and to start using the other endpoint as the far-end address. In other words, A would be notified that B is now the far end and B would be notified that A is the far end. Later the switch may choose to change the path again due to some change in the call, such as a conference or transfer. Each time the path changes, the endpoints are notified via media control events to stop using the current far-end address and to start using the new far-end address.

Cleanup

It is important to free resources when they are no longer needed. This is most likely to occur when your application:

- Detects the end of a call
- Is finished with a device
- Is about to exit

Cleanup should occur in this order:

1. **Stop collecting tones**

   At the end of a call, you can choose to stop collecting tones for the device. Alternatively, you can let the collection and the retrieval criteria continue across calls. In that case you might just flush the buffer at the end of each call.

   When finished with a device, stop the tone collection for that device.

   When the application is about to exit, stop tone collection on all devices.

2. **Stop recording or playing**

   At the end of a call, you can choose to stop recording or playing, or let the recording or playing continue across calls on the device.

   When finished with a device, stop both recording and playing on that device.

   When the application is about to exit, stop both recording and playing on all registered devices.

   Both recording and playing can be stopped on a device by sending the Voice Unit Services `Stop` request or can be individually stopped by sending the Extended Voice Unit Services `StopRecording` request or `StopPlaying` request.

3. **Unregister the device**

   When finished with a device, unregister it by sending the Terminal Services `UnregisterDevice` request.

   When the application is about to exit, unregister each registered device.

4 **Stop monitoring for events**

When your application no longer needs to receive events for a device, send a `MonitorStop` request.

5 **Release the device identifier**

When finished with a device identifier, release it by sending the Device Services `ReleaseDeviceID` request.

When the application is about to exit, release each device identifier.

Security considerations

Your application development organization has the responsibility of providing the appropriate amount of security for your particular application and/or recommending appropriate security measures to your application customers for the deployment of your application. Therefore you should be aware of the security measures that Communication Manager API already takes and what risks are known.

Security measures taken by Communication Manager API include:

- The station password is required to register a device.
- Filenames specified for recorded files must be relative to the configured directory, their directories must already exist, and recordings cannot overwrite an existing file.
- Only files within the configured recorder directory can be deleted using the Voice Unit Services `DeleteMessage` request.

The security risks include:

- The signaling and bearer channels are unencrypted.
- XML messages transmitted between the client application and the connector server software are unencrypted.
- There is no authentication on the JMX™ web console.

Avaya recommends that the application machine, connector server machine, and Communication Manager machines reside on a private LAN.

4 Debugging

In debugging your application, you will rely on:

1. **Exceptions** that your application received and logs. See Receiving exceptions on page 35.
2. **Server-side logs** found at /opt/cmapi/logs. See the *Management* guide to learn more about the server's logs.

The remainder of this chapter describes:

Common exceptions

Common exceptions that you may encounter are listed below along with their potential solutions:

Table 17: Common exceptions

Exception	Potential cause and solution
`java.net.ConnectException: Connection refused`	This means that the connector server IP address your application used to establish a connection to the connector server is a valid IP address, but it cannot be connected to. This may occur: • if the connector server software is not actively running or if the wrong address was provided. Check that the connector server software is running and that the correct IP address is provided for the server. • if there are network issues between the application machine and the connector server. Check to see if you can ping the server. • The connector server software is not running on the connector server. Try restarting it. If you have checked all of these things and are still having a problem, you may also want to check the /etc/hosts file on the connector server and verify that you have a line in that file that explicitly lists the IP address of the connector server, in addition to the localhost line.
`java.nio.channels.UnresolvedAddressException`	This likely means that your application used an invalid IP address to establish a connection to the connector server. Correct the address. Do *not* use local host address 127.0.0.1; use the actual IP address.
`ch.ecma.csta.errors.CstaException <stack trace> Caused by: java.nio.channels.UnresolvedAddressException`	The IP address for Communication Manager that was specified in the Device Services `GetDevice` request is incorrect. This exception is not sent until you first try to use the `deviceId` in a Terminal Services `RegisterDevice` request. Ensure that you are using the correct IP address for Communication Manager.
`InvalidConnectionIDException: Player unavailable for this session: Note: Player will be unavailable if the device is using G729 or G729A codec <connection id>`	Device specified in the Voice Unit Services `PlayMessage` request is unregistered. It may have been automatically unregistered due to a loss of communication with Communication Manager. See Possible race conditions below. Check the health of Communication Manager and the network and then try again.

1 of 2

Table 17: Common exceptions *Continued*

Exception	Potential cause and solution
`InvalidConnectionIDException: invalid mediasession or Recorder <connection id>`	Device specified in the Voice Unit Services `RecordMessage` request is unregistered. It may have been automatically unregistered due to a loss of communication with Communication Manager. See Possible race conditions below. Check the health of Communication Manager and the network and then try again.
`NotAbleToPlayException: File already being played.`	Two different threads of the application may be trying to play messages to the same device at the same time. See Possible race conditions below. Modify the application so that this does not occur.
`ch.ecma.csta.errors.CstaException: Request with info=com.avaya.mvcs.proxy.XmlGatewayClient$RequestInfo@fa70a4 timed out`	Connector server did not respond in time so the client proxy timed out. Look at the connector server logs to determine the cause.

2 of 2

Possible race conditions

You should be aware of some scenarios in which two different threads may be acting in opposition to one another, against a single device. Some known race conditions are described below:

- When the connector server detects through its “keep-alive” mechanism that it can no longer communicate with a Communication Manager C-LAN (CLAN) or processor C-LAN (PROCR) that has devices registered to it (possibly due to a network failure or congestion), the connector server automatically unregisters the devices; any media sessions for those devices are cleaned up; and an `UnregisterEvent` is sent to all the applications that requested to be notified of these events. If, at the same time, an application is in the middle of sending a media request to play a file, start tone detection, or start recording on one of those automatically unregistered devices, an `InvalidConnectionIDException` is sent indicating that the media session is unavailable.
- If one thread is actively playing a message to a device and a second thread attempts to play a message to the same device, the second thread will receive a `NotAbleToPlayException`.
- If an application simultaneously uses both of the following:
  - Voice Unit Services to request that playing or recording terminate when a specified DTMF digit is detected
  - Tone Detection Services or Tone Collection Services to listen for DTMF tones

  there is no guarantee of which of the following occurs first when the termination digit is received:
  - the Voice Unit Services player/recorder terminates and a `StopEvent` is generated.
  - the Tone Detection Services/Tone Collection Services generates a `ToneDetectedEvent`/`TonesRetrievedEvent`

If the application assumes the player or recorder has stopped playing/recording when the `ToneDetectedEvent`/`TonesRetrievedEvent` is received and requests another play/record request, an exception may be received if the play/record is still in progress on that device. If the application wants to be certain that the player/recorder is finished, it should wait for the `StopEvent` before making another play/record request.

Improving performance

Many factors can affect the performance of your system. The system has three main parts that may be affected:

- the connector server
- Communication Manager
- the network

An excessive load on any of these has the potential to slow down the overall system. Here are other factors to check that also may affect your system performance.

On the connector server:

- Ensure that your connector server machine meets the minimum requirements specified in the *Quick Start* guide.
- Avoid running any other applications on the connector server machine.
- Check that the connector servers' Linux operating system resources are tuned correctly for your application needs. The connector server software makes no assumptions concerning your application needs and therefore does not tune the kernel for you. The *Communication Manager API Installation and Administration* guide provides some guidance in tuning Linux. Also refer to Linux documentation found at http://www.redhat.com/docs/manuals/linux/RHL-9-Manual.
- Update the Linux kernel with the latest updates available.

On Communication Manager:

- Ensure that Communication Manager is properly configured for your network and business needs. Misconfigured switch elements can lead to performance hits.

On the network:

- Ensure that your network traffic is properly balanced. One way to do this professionally is to ask Avaya to perform a network assessment. There is also a *VoIP Readiness Guide* available from the Avaya Support Centre (http://www.avaya.com/support). For more information about improving the performance of your network, see the "Network Quality and Management" section of *Administration for Network Connectivity for Avaya Communication Manager* (555-233-504).

Getting support

If you have trouble while developing an application, contact DevConnect (email devconcmapi@avaya.com) for support.

A Communication Manager Features

Here is a list of key features in Communication Manager that you may wish to take advantage of in developing your applications. This is not an exhaustive list - just a subset of features most likely to be used in applications. For descriptions of these features, see any of the Communication Manager administration guides.

- Abbreviated Dialing
- Abbreviated Programming
- Automatic Alternate Routing (AAR)
- Automatic Call Distribution (ACD) Features
  - Announcements
  - Automatic Answering With Zip Tone
  - Multiple Call Handling
  - Service Observing
    - Observe Digital Sets/IP Phones
    - Observe Logical Agent IDs
- Bridged Call Appearance (single-line, multi-appearance)
- Call Forwarding / Busy Don’t Answer @ Call Vectoring
  - VDN of Origin Announcements
- Drop (button operation)
- Hold (single-line, multi-appearance)
- Hold - Automatic
- IP Trunks
- Last Number Dialed
- Message Waiting Indication
- Music-on-Hold Access
  - Held Calls
- Tone on Hold
- Transfer (single-line, multi-appearance)
- Voice Terminal Display
  - Calling Number Display (SID/ANI/Extn ID)
  - Called Number Display (internal & DCS)
  - Stored Button Display
- Automatic Route Selection (ARS)

- Automatic Call Distribution (ACD) Features:
  - Basic Hunt Group Call
  - Agents in Multiple Splits
  - Agent Login/Out
  - Display - Agent Terminal
- Automatic Callback (on Busy)
- Call Coverage Features
  - Call Coverage Consult with Conference/Transfer
- Call Forwarding by Service Observer
- Call Forwarding by Service Observed
- Call Park
- Call Pickup
  - Directed Call Pickup
- Call Vectoring:
  - VDN of Origin Display
- Group Paging
- Hunting/Hunt Groups
- Leave Word Calling - Switch
- Malicious Call Trace
- Personalized Ringing
- Priority Calling
- AAR/ARS Partitioning
- Abbreviated and Delayed Ringing
- Abbreviated Dialing
- Administration Without Hardware
- Authorization Codes
- Automatic Alternate Routing (AAR)
- Automatic Callback (on Busy)
- Automatic Callback on Don’t Answer
- Automatic Route Selection
- Bridged Call Appearance (single-line, multi-appearance)
  - Hunt Group Redirect Coverage
  - Multiple Coverage Paths
  - Temporary Bridged Appearance
  - Remove Temporary Bridged Appearance

- Call Coverage Features
  - Consult
  - Coverage Paths
  - Send All Calls
  - Temporary Bridged Appearance
- Call Forwarding Features:
  - Call Forwarding All Calls
  - Call Forwarding - Busy and Don’t Answer
  - Call Forwarding - Don’t Answer
  - Call Forwarding - Off Net
- Call Park
- Call Pickup
  - Directed Call Pickup
  - Remove Temporary Bridged Appearance
  - Remove Auto-Intercom
- Consult
- Coverage of Calls Redirected Off Net
- Hold (single-line, multi-appearance)
- Hold - Automatic [from IR1V4 WCC]
- Hunting/Hunt Groups
- Malicious Call Trace
- Music-on-Hold Access
  - Held Calls
  - Conference-Terminal Calls
  - Transferred Trunk Calls
- Personal Station Access
- Personalized Ringing
- Priority Calling
- Recorded Announcement
- Terminal Translation Initialization (TTI)

B Constant Values

Here are tables describing the values for the XML message parameters which take a constant value and that are switch specific. These values are specific to Avaya's Communication Manager switch. These include:

- Physical Device Constants
  - Lamp Mode Constants
  - Lamp Color Constants
  - Button Function Constants
  - Button ID Constants
  - Ringer Pattern Constants
- Registration Constants
  - Reason code constants for the `RegisterFailedEvent` and the `UnregisterEvent`
  - Reason code constants for the `UnregisterEvent`
  - Reason code constants for the `RegisterFailedEvent`

Physical Device Constants

Table 18: Lamp Mode Constants

Lamp Mode	Value
Broken Flutter	0
Flutter	1
Off	2
Steady	3
Wink	4
Inverted Wink	6
Flash	7
Inverted Flash	8

Table 19: Lamp Color Constants

Lamp Color	Value
Red	1
Green	3

Table 20: Button Function Constants

Button Function	Value
Abbreviated Dial Program	"67"
AD Special Character	"68"
Analog Bridged Appearance	"114"
Abbreviated Dial	"65"
Button to force Abbreviated / Delayed Ring Transition	"226"
Administered Connection alarm button	"128"
ACA referral call activate button	"77"
CDR Account Code Button	"134"
Enter Terminal Self-Admin Mode	"150"
ACD - After Call Work	"91"
Move agent while staffed alert	"225"
Alternate FRL button	"162"
Incoming ANI Request (Russian trunks only)	"146"
ACD - supervisor assist button	"90"
Attended group - number of queued calls	"89"
Attended group - oldest time	"88"
Automatic message waiting indication	"70"
Auto-dial ICOM	"69"
ACD - Auto-In	"92"
Automatic Wakeup entry mode	"27"
Autodial button	"227"
Bridged appearance of primary extension	"73"
Button Ring Control entry mode	"258"
Enhanced View Button	"151"
Station busy indicator	"39"
General call appearance, no aux data	"6"
Call the displayed number	"16"
Call forwarding button	"74"
Park/Unpark button	"45"
UM call pickup	"34"
Call Timer	"243"

1 of 5

Table 20: Button Function Constants *Continued*

Button Function	Value
Caller Information	"131"
CAS (branch) back-up mode lamp	"76"
SMDR primary printer alarm	"106"
SMDR secondary printer alarm	"117"
Call Forward/Busy Don't answer	"84"
Check-in entry mode	"29"
Check-out entry mode	"28"
Clocked override-time of day routing	"112"
Consult/return	"42"
Coverage LWC call back	"17"
Coverage retrieve LWC message	"12"
Per call CPN blocking activate	"164"
Per call CPN unblocking activate	"165"
Crisis Alert to Digital Station or Attendant.	"247"
Data extension button	"43"
TOD/DATE display mode	"23"
Delete LWC message	"14"
Autodial, aux is uid of destination	"32"
DID remove entry mode	"276"
DID view entry mode	"256"
Directed call pickup	"230"
Directory listing for name search	"230"
Display Charge button	"232"
NORMAL/LOCAL mode button	"124"
User do not disturb button	"99"
Drop the last party on a conference call	"1"
Manual exclusion	"41"
Do not disturb - ext.	"95"
Conf Select Far End Mute	"328"
Flash button for station on CAS MAIN call OR a call using Trunk Flash	"110"
Goto coverage button	"36"
Do not disturb - grp.	"96"
Group Paging button	"135"

2 of 5

Table 20: Button Function Constants *Continued*

Button Function	Value
Headset in use	"241"
Hunt night service button	"101"
Inspect display mode	"21"
Internal Automatic Answer button	"108"
License error	"312"
Link alarm button	"103"
SVN login security violation	"144"
Cancel LWC	"19"
Lockout LWC	"18"
LWC store message	"10"
Major alarm button	"104"
Manual message waiting button	"38"
Manual override-time of day routing	"113"
ACD - Manual-In	"93"
MCT: Malicious Call Trace Activate	"160"
MCT: Malicious Call Trace Control	"161"
Major/Minor alarm button	"82"
Multimedia Voice/Data activate/deactivate	"169"
Multimedia Call Mode activate	"167"
Multimedia Call Forward	"244"
Multimedia Data Conference activate	"168"
Multimedia Multi Address activate	"170"
Multimedia PC-Audio activate	"166"
Principal retrieve LWC message	"11"
Message notification on mode	"97"
Message notification off mode	"98"
Step through LWC message	"13"
Night Activate	"53"
No Answer Alert button for Redirect On No Answer (RONA) timeout for split.	"192"
Normal display mode	"15"
Off-board DS1 board alarm button	"126"
Personal CO line, aux is grp id	"31"
PMS alarm button	"105"

3 of 5

Table 20: Button Function Constants *Continued*

Button Function	Value
Wakeup journal printer alarm	"116"
PMS journal printer alarm	"115"
BCMS printer link alarm button	"120"
Priority calling button	"81"
Hunt group - number of queued calls	"87"
Hunt group - oldest queued time	"86"
ACD - Release	"94"
Ringer cut button for stations	"80"
System reset alert	"109"
SVN remote access security violation	"145"
SCROLL mode button	"125"
Send all calls button	"35"
Terminating extension group SAC button	"72"
Service observing button	"85"
Manual signalling	"37"
SVN station security call activate button	"231"
Station Lock	"300"
Start Billing	"257"
STORED-NUMBER display mode	"22"
Single-Digit Stroke Counts	"129"
Secondary extensions	"40"
ELAPSED-TIME display mode	"24"
Conf/Transfer Toggle Swap	"327"
Facility acc test trunk access alert	"121"
Trunk ID button	"63"
Trunk name when DCS (also DCS CAS MN)	"111"
Trunk night service button	"102"
UUI-info button	"228"
Busy verification button	"75"
VDN of Origin Annc. Repeat Button	"208"
Vu-Stats Station Displays	"211"
Activate Whisper Page	"136"
	4 of 5

Table 20: Button Function Constants *Continued*

Button Function	Value
Answerback for Whisper Page	"137"
Whisper Page Off	"138"
IM_MDSTROKE	"130"
	5 of 5

Table 21: Button Id Constants

Button Id	Value
Dial Pad 0	"0"
Dial Pad 1	"1"
Dial Pad 2	"2"
Dial Pad 3	"3"
Dial Pad 4	"4"
Dial Pad 5	"5"
Dial Pad 6	"6"
Dial Pad 7	"7"
Dial Pad 8	"8"
Dial Pad 9	"9’
Dial Pad *	"10"
Dial Pad #	"11"
Principal Module	"256"
Redial button on IP phone sets	"257"
Drop button on DCP phone sets	"258"
Conference button on IP/DCP phone sets	"259"
Transfer button on IP/DCP phone sets.	"260"
Hold button on IP/DCP phone sets.	"261"
First call appearence button on IP/DCP phone sets.	"263"
Second call appearence button on IP/DCP phone sets.	"264"
Third call appearence button on IP/DCP phone sets.	"265"
Button 4 on IP/DCP phone sets.	"266"
Button 5 on IP/DCP phone sets.	"267"
Button 6 on IP/DCP phone sets.	"268"
Button 7 on IP/DCP phone sets.	"269"
	1 of 3

Table 21: Button Id Constants *Continued*

Button Id	Value
Button 8 on IP/DCP phone sets.	"270"
Button 9 on IP/DCP phone sets.	"271"
Button 10 on IP/DCP phone sets.	"272"
Button 11 on IP/DCP phone sets.	"273"
Button 12 on IP/DCP phone sets.	"274"
Button 13 on IP/DCP phone sets.	"275"
Button 14 on IP/DCP phone sets.	"276"
Button 15 on IP/DCP phone sets.	"277"
Button 16 on IP/DCP phone sets.	"278"
Button 17 on IP/DCP phone sets.	"279"
Button 18 on IP/DCP phone sets.	"280"
Button 19 on IP/DCP phone sets.	"281"
Button 20 on IP/DCP phone sets.	"282"
Button 21on IP/DCP phone sets.	"283"
Button 22 on IP/DCP phone sets.	"284"
Button 23 on IP/DCP phone sets.	"285"
Button 24 on IP/DCP phone sets.	"286"
Feature Module	"512"
Call Coverage Module	"768"
Display Module	"1024"
DTDM	"1280"
DXS/BLF Module	"1536"
Terminal Module (Type 2)	"1792"
Menu Button on IP/DCP phone sets.	"785"
Exit Button on IP/DCP phone sets.	"782"
First Button on the first row on IP/DCP phone sets.	"770"
Second Button on the first row on IP/DCP phone sets.	"771"
Third Button on the first row on IP/DCP phone sets.	"772"
Fourth Button on the first row on IP/DCP phone sets.	"773
First Button on the second row on IP/DCP phone sets	"774"
Second Button on the second row on IP/DCP phone sets	"775"
Third Button on the second row on IP/DCP phone sets	"776"
Fourth Button on the second row on IP/DCP phone sets	"777"
	2 of 3

Table 21: Button Id Constants *Continued*

Button Id	Value
First Button on the third row on IP/DCP phone sets	"778"
Second Button on the third row on IP/DCP phone sets	"779
Third Button on the third row on IP/DCP phone sets	"780"
Fourth Button on the third row on IP/DCP phone sets	"781"
Exit Button on DCP 8410D phone sets.	"273"
First Button on the first row on DCP8410 phone sets.	"274
Second Button on the first row on DCP8410 phone sets.	"275"
Third Button on the first row on DCP8410 phone sets.	"276"
Fourth Button on the first row on DCP8410 phone sets.	"277"
First Button on the second row on DCP8410 phone sets.	"278"
Second Button on the second row on DCP8410 phone sets.	"279"
Third Button on the second row on DCP8410 phone sets.	"280"
Fourth Button on the second row on DCP8410 phone sets.	"281"
First Button on the third row on DCP8410 phone sets.	"282
Second Button on the third row on DCP8410 phone sets.	"283"
Third Button on the third row on DCP8410 phone sets.	"284"
Fourth Button on the third row on DCP8410 phone sets.	"285"
Exit Button on DCP 8434D phone sets.	"529"
First Button on the first row on DCP8434 phone sets.	"530"
Second Button on the first row on DCP8434 phone sets.	"531"
Third Button on the first row on DCP8434 phone sets.	"532"
Fourth Button on the first row on DCP8434 phone sets.	"533"
First Button on the second row on DCP8434 phone sets.	"534"
Second Button on the second row on DCP8434 phone sets.	"535"
Third Button on the second row on DCP8434 phone sets.	"536"
Fourth Button on the second row on DCP8434 phone sets.	"537"
First Button on the third row on DCP8434 phone sets.	"538"
Second Button on the third row on DCP8434 phone sets.	"539"
Third Button on the third row on DCP8434 phone sets.	"540"
Fourth Button on the third row on DCP8434 phone sets.	"541"
	3 of 3

Table 22: Ringer Pattern Constants

Ring Pattern	Value
Ringer Off	0
Manual Signal	1
Attendant Incoming Call	4
Attendant Held Call	5
Attendant Call Waiting	6
Attendant Emergency	7
Intercom	9
Standard Ring	11
DID/Attendant Ring	12
Priority Ring	13
Ring Ping	14

Registration Constants

Table 23: Registration Constants

Reason code constants for the RegisterFailedEvent and the UnregisterEvent	Value
Unknown	-1
Non-specific	-2
Internal failure	-3
Network timeout	1000

Table 24: Registration Constants

Reason code constants for the UnregisterEvent only	Value
Client application requested unregistration.	2000

Table 25: Registration Constants

Reason code constants for the RegisterFailedEvent only	Value
A bad password/DeviceID combination was sent in the register request	3000
A non-existent extension on the switch was specified.	3001
The phone/extension is already registered with the switch and the forceLogin option was set to false	3002
The limit of allowed CMAPI softphones has been reached	3003
The limit of allowed IP phones has been reached	3004
The license file indicates that no IP_API_A licenses have been purchased	3005
The 911 registration parameters did not match the 911 settings of the station on the switch	3006
The station administration for the specified extension is not valid for softphone use	3007

Glossary

A

API
Application Programming Interface. A “shorthand” term in this documentation for the Java interface provided by the Communication Manager API. See also connector client API library

application machine
The hardware platform that the connector client API library and the client application are running on

B

BHCC
busy hour call capacity

C

client application
An application created using Communication Manager API

CMAPI softphone
Communication Manager API software objects that represent softphone-enabled, Communication Manager telephones or extensions

Communication Manager API
The product name. This includes the server-side runtime software (see connector server software) and the connector client API library. This term is never used to reference only the client API library.

connector
This describes the function of Communication Manager API

In this context, “connector” means software and communications protocol(s) that allow two disparate systems to communicate. Often used to provide open access to a proprietary system. In the case of Communication Manager API, the connector enables applications running on a computing platform to incorporate telephony functionality through interaction with Communication Manager.

connector client API library
The Communication Manager API Java API, also referred to as the API

connector server machine
The hardware platform that the connector server software is running on. In these books, the term “connector server” by itself never refers to the connector server machine. See connector server software.

connector server software
The Communication Manager API server-side runtime software, often referred to as the “connector server” in these documents

CSTA
Computer-Supported Telecommunications Applications

D

DMA
Direct memory access

E

ECMA
European Computer Manufacturers Association. A European association for standardizing information and communication systems in order to reflect the international activities of the organization.

H

hold time
The total length of time in minutes and seconds that a facility is used during a call

J

JDK
Java Developers Kit

J2SE
Java™ 2 Platform, Standard Edition

JSW
Java Service Wrapper

JVM
Java Virtual Machine

R

RPM
Red Hat Package Manager

S

SAT
System Access Terminal (for Communication Manager)

SDK
Software Development Kit. An SDK typically includes API library, software platform, documentation, and tools.

T

TCP
Transmission Control Protocol. A connection-oriented transport-layer protocol, IETF STD 7. RFC 793, that governs the exchange of sequential data. Whereas the Internet Protocol (IP) deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data, and also guarantees that packets are delivered in the same order in which the packets are sent.

TTI

Terminal Translation Initialization. This is a feature in Communication Manager that allows administrators, when initially administering new DCP stations, to not initially bind the extension number to a port. When the technician is installing the stations, they then use the TTI feature access code to bind the extension number to the station.

V

VoIP

Voice over IP. A set of facilities that use the Internet Protocol (IP) to manage the delivery of voice information. In general, VoIP means to send voice information in digital form in discrete packets instead of in the traditional circuit-committed protocols of the public switched telephone network (PSTN). Users of VoIP and Internet telephony avoid the tolls that are charged for ordinary telephone service.

XML

Extensible Markup Language

XSD

XML Schema Definition. Specifies how to formally describe the elements in an Extensible Markup Language (XML) document. This description can be used to verify that each item of content in a document adheres to the description of the element in which the content is to be placed. This protocol uses these instead of DTD’s.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X