Cisco Unified Communications Gateway Services API Guide

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Text Part Number: OL-25225-01

Cisco Unified Communications Gateway Services API Guide OL-25225-01

CONTENTS

Preface

This preface describes the purpose, audience, organization, and conventions of this guide and provides information on how to obtain related documentation.

The preface covers these topics:

Purpose

This document describes the Cisco Unified Communications Gateway Service Application Programming Interface . This document outlines the concepts of the Cisco Unified Communications Gateway Services API and provides information on configuring the Cisco Integrated Services Router.

Audience

This document is for developers who write applications to access Cisco Unified Communication IOS services on the Cisco ISR. The developer must have knowledge or experience in the following areas:

- Cisco IOS software
- Web Services Description Language (WSDL)
- Simple Object Access Protocol (SOAP)
- HTTP

Organization

This guide includes the following sections:

Chapter Title	Description
Cisco Unified Communications Gateway Services API	This chapter describes the concepts and information on providers in the InterfaceCisco Unified Communications Gateway Services API.
Configuring Cisco Unified Communication IOS Services	This chapter describes the CLI commands that are used to enable and troubleshoot the Cisco Unified Communication IOS services on the voice gateway.
Provider and Application Interactions	This appendix contain the interactions and sample messages that are passed between the application and the provider.

Conventions

This document uses the following conventions:

Convention	Indication
bold font	Commands and keywords and user-entered text appear in **bold** font.
italic font	Document titles, new or emphasized terms, and arguments for which you supply values are in *italic* font.
[ ]	Elements in square brackets are optional.
{x | y | z }	Required alternative keywords are grouped in braces and separated by vertical bars.
[x | y | z]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.
`courier` font	Terminal sessions and information the system displays appear in `courier` font.
< >	Nonprinting characters such as passwords are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

Note Means *reader take note*.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

Subscribe to the *What's New in Cisco Product Documentation* as an RSS feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service. Cisco currently supports RSS Version 2.0.

CHAPTER 1

Cisco Unified Communications Gateway Services API

This chapter describes the Cisco Unified Communications Gateway Services API. This API enables the development of advanced Cisco Unified Communication applications and services on the Cisco Integrated Services Router (ISR) by providing an interface to the Cisco Unified Communication IOS services.

Cisco Unified Communications Gateway Services API provides the developer with access to the following unified communication IOS services:

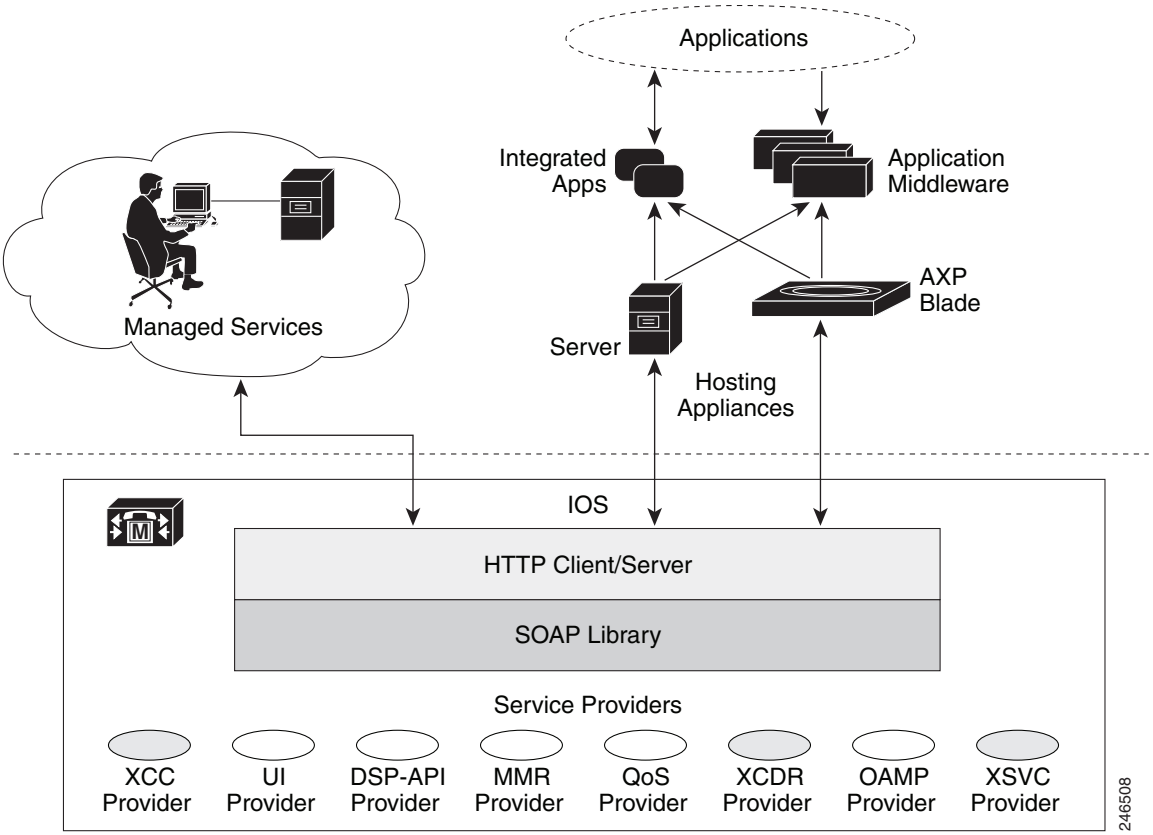
- Extended Call Control Service
- Extended Serviceability Service
- Extended Call Detail Record (CDR) Service

Overview

Cisco Unified Communications Gateway Services API allows you to develop an application that interacts with the Cisco Unified Communication IOS services on voice gateways. The application accesses the Cisco Unified Communication IOS services via SOAP messages.

Figure 1-1 illustrates the Cisco Unified Communication IOS service interface. Cisco currently supports the extended call control (XCC) provider, extended call detail record (XCDR) provider, and extended serviceability (XSVC) provider.

Figure 1-1 Cisco Unified Communications Gateway Services API



Cisco Unified Communication IOS Services

Web service is a standards-based framework that allow applications operating on different platforms to interact over the Internet. Cisco Unified Communication IOS services, like web services, are platform independent and language neutral. With Cisco Unified Communications Gateway Services API, you can develop your application in the language and operating system of your choice and communicate directly with the Cisco Unified Communication IOS services running on the voice gateway.

The Cisco Unified Communications Gateway Services API supports the following standards and protocol:

- XML 1.0
- Web Services Description Language (WSDL) 1.1
- SOAP, version 1.2
- HTTP, version 1.1

Providers

The providers on the voice gateway provide services on the voice gateway for remote applications. Cisco Unified Communications Gateway Services API enables applications to interact with the providers and is comprised of the following provider objects:

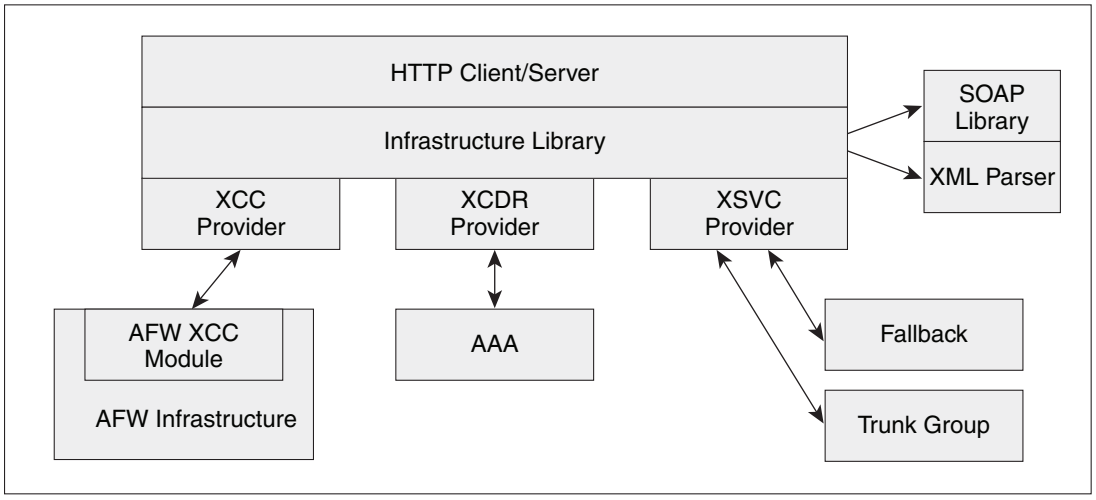
- XCC Provider— Extended Call Control (XCC) provider supports operations that allow an application to perform call control and real-time call monitoring.
- XCDR Provider—Extended Call Detail Record (XCDR) provider supplies CDR information to the application and notifies the application when calls have ended.
- XSVC Provider—Extended Serviceability (XSVC) provider monitors trunk status, and provides real-time link status and configuration change notification to application.

Each provider has a unique URL identifier and communicates with the application via SOAP messages. The providers can be in one of two states:

- In-service—Provider is active and available for use.
- Shutdown—Provider is disabled and no longer available. The API methods associated with this provider are invalid in this state.

Figure 1-2 illustrates the relationship between the IOS components.

Figure 1-2 Cisco Unified Communication IOS Services Components



When a provider is configured and enabled on the voice gateway, it performs the following functions:

- Manages the registration process between the application and the provider.
- Sends notification to the application when a provider changes its status.
- Passes incoming messages to the appropriate provider.
- Notifies the provider when there is a message exchange failure.
- Sends probing messages to maintain an active registration session.
- Sends negative probing messages to detect the status of an application. If the number of failed responses exceeds a configured number of negative probing messages, the voice gateway unregisters the application.

WSDL Files

Cisco Unified Communications Gateway Services API uses the WSDL specification to define the services that are available on the voice gateway. These services are represented as providers on the voice gateway.

Table 1-1 lists the namespace for the Cisco Unified Communications IOS services

Table 1-1 Cisco Unified Communication IOS Services Namespace

Service	Location
XCC	http://www.cisco.com/schema/cisco_xcc/v1_0
XCDR	http://www.cisco.com/schema/cisco_xcdr/v1_0
XSVC	http://www.cisco.com/schema/cisco_xsvc/v1_0

Inbound Ports

Table 1-2 lists the URL and inbound location that the application uses to communicate with the server.

Table 1-2 Location of the Inbound Port

Service	Namespace
XCC	http://<access_router>:8090/cisco_xcc [1]
XCDR	http://<access_router>:8090/cisco_xcdr [1]
XSVC	http://<access_router>:8090/cisco_xsvc [1]

1.The access router is the hostname or IP address of the router that with Cisco Unified Communications IOS services.

Registering an Application

Before an application can register with the voice gateway, you must first configure the application's service URL on the router. The URL is used to authenticate messages from the application. When the router first boots up, the provider sends status messages to the applications that are in its configuration. The router sends status messages when the provider changes its status.

The application initiates registration by sending a registration message to the appropriate provider. The provider generates a unique registration ID and sends it back to the application. The unique registration ID identifies the registered session and is used in all messages that are sent during the registered session.

States of a Registered Session

The state of the registered session and the status of the messages that are sent between the provider and application have one of the following value:

- Steady State—This state is the normal state of the registered session. Messages and acknowledgements are exchanged regularly in this state.

- Keepalive State—When the provider does not have messages to send, the voice gateway sends keepalive probing messages to the registered application This keeps the connection between the application and the provider active. The messages that are sent in this state contain information on the health and connectivity status of the provider.
- Negative Probe State—When the number of failed responses exceeds the maximum number of failed responses, the registered session enters the negative probe state. In the negative probe state, the voice gateway sends negative-probing messages in an attempt to reestablish the steady state or the keepalive state with the application. The only message sent in a negative probe state is a negative probe message. The registered session returns to a steady state or keepalive state upon receipt of a successful response to a negative probe message, and regular messages resume.
- Unregistered State—The session is unregistered and no messages are exchanged between the provider and the application. The session enters an unregistered state under the following conditions:
  - When the application unregisters with the provider
  - When an application fails to respond to probing messages
  - When the administrator shuts down the provider service on the voice gateway

XCC Provider

The XCC provider gives an application the capability to control all the legs of a standard call. With the XCC provider, the application can perform auxiliary call control and can control some network elements.

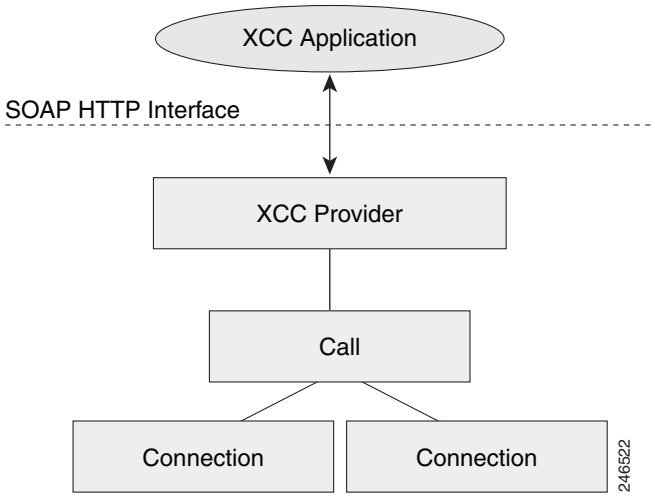
Characteristics of the XCC Provider

The XCC provider has the following characteristics:

- The XCC provider allows the application to maintain stateful control on a call over the entire life cycle of the call.
- The XCC provider allows the application to subscribe and receive mid-call event notification. The application can change event subscription over the life of the call.
- The XCC provider allows services to be invoked on a network triggered event. The provider reports on notifications from a direct application request.
- The XCC provider follows a generic call model in which the underlying communication protocol and architecture is hidden from the developer. XCC provider uses a high-level call control model for maintaining and managing the state of a call session.

Figure 1-3 illustrates the XCC call control abstraction.

Figure 1-3 XCC Call Control



XCC Provider API

When an application registers with the XCC provider, the application configures event filter parameters that the application is interested in monitoring, and the XCC provider installs a connection listener to monitor the calls. XCC notifies the application when a call or connection event matches the event filters that were configured. When the application updates event filter parameters, the updates only apply to new calls, not existing calls.

The XCC provider API is described in Table 1-3. For additional information, see the XCC Provider API.

Table 1-3 XCC Provider API

XCC Provider API	Direction	Description
RequestXccRegister	Application to XCC Provider	Registration request sent with event filter settings for the blocking timeout, connection event, or media filters in the message.
RequestXccUnRegister	Application to XCC Provider	Message sent from the application requesting to be unregistered.
RequestXccControlUpdate	Application to XCC Provider	Message sent with updated connection or media event filters and updated blocking timeout setup.
ResponseXccRegister	XCC provider to the application	Message sent in response to a registration request.
ResponseXccUnRegister	XCC provider to the application	Message sent in response to an unregistration request.
ResponseXccControlUpdate	XCC provider to the application	Message sent in response to a updated event filter request.
NotifyXccStatus	XCC provider to the application	Operation-triggered message sent reporting on the XCC provider status. The following statuses are valid: • IN_SERVICE • SHUTDOWN

Table 1-3 XCC Provider API

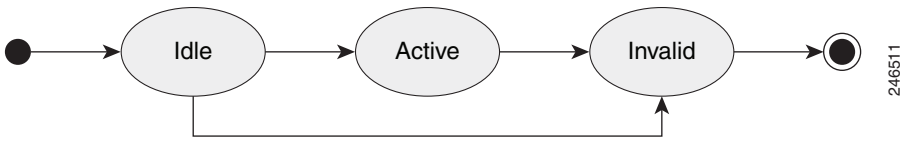
XCC Provider API	Direction	Description
SolicitXccProbing	XCC provider to the application	Blocking message sent to keep the registration session alive and to check on the health of the application.
SolicitXccProviderUnRegister	XCC provider to the application	Blocking message sent to inform the application that the XCC provider has entered the shutdown state and the registration session is now unregistered.
ResponseXccProbing	Application to XCC Provider	Message sent in response to the XCC probing message.
ResponseXccProviderUnRegister	Application to XCC Provider	Message sent in response to the XCC unregistered message.

The XCC call APIs describe the endpoints and trunks that are associated with a call. The APIs in XCC call API, and the associated XCC connection describes the control and media flow in a call. The provider notifies the application when there is a change to the state of a call and sends update information on the call, address, and connections.

A call abstraction is represented in Figure 1-4 on the voice gateway in one of the following three states:

- IDLE—Initial state of all calls. A call in an idle state has zero connections.
- ACTIVE—Call with ongoing activity. A call in an active state has one or more associated connections.
- INVALID—Final state of all calls. Calls that lose all their connections are moved into this state. A call in the invalid state has zero connections.

Figure 1-4 Call Abstraction Model



The XCC call API is described in Table 1-4.

Table 1-4 Xcc Call API

Operation	Direction	Description
RequestXccCallRelease	Application to XCC Provider	Message sent requesting that a call session be released
ResponseXccCallRelease	XCC provider to the application	Message sent in response to the application's call release request.
RequestXccCallMediaForking	Application to XCC Provider	Message sent to enable media forking for the call session.
RequestXccCallMediaSetAttributes	Application to XCC Provider	Message sent to notify the XCC provider that the media attributes for the call session has changed, for example if a call is changed from "voice" to "fax".

Table 1-4 Xcc Call API

Operation	Direction	Description
ResponseXccCallMediaForking	XCC provider to the application	Message sent in response to the application's media forking request.
ResponseXccCallMediaSetAttributes	XCC provider to the application	Message sent in response to the application's media set attribute request.
NotifyXccCallData	XCC provider to the application	Operation-triggered message sent to notify the application when one of the following conditions occurs in a call session: • The mode has changed. • DTMF[1] digit was detected. • Media inactive or active is detected.

1. DTMF = dual-tone multi-frequency

XCC Call Media Set Attributes

External applications can enable the voice gateway to detect changes to the call media set attributes on a call and have the voice gateway send a notify event message. Table 1-5 lists the call media set attributes that the gateway can detect.

Table 1-5 Call Media Set Attributes

Call Media Set Attributes	Description
Call Mode Change	Enables the voice gateway to detect when a call changes between the following call modes: • Voice Call • Fax Call • Video Call • Modem Call • Data Call **Note** ISDN calls with an unrestricted bearer capability value are reported as data calls.
DTMF	Enables the voice gateway to detect a DTMF digit in the media stream or a DTMF relay. **Note** The notify event message includes the timestamp if the DTMF event is detected in IOS. **Note** For notify event messages, the application should use the voice gateway as the NTP[1] server for synchronizing clocks.
Media Activity	Enables the voice gateway to detect when the media activity state changes from "Active" to "Inactive" or vice versa.

Table 1-5 Call Media Set Attributes

Call Media Set Attributes	Description
Tone	Enables the voice gateway to detect the following specified tones: • Busy Tone • Dial Tone • Ringback Tone • Out-of-Service Tone • Second Dial Tone **Note** Tone detection is not supported for a FXO voice port if the supervisory tone detection feature is enabled.
Media Forking	Enables media forking on a connected call to target a RTP address. For more information on media forking, see the "XCC Call Media Forking" section on page 1-9.

1. NTP = network time protocol.

XCC Call Media Forking

External applications can request media forking for a call. When the application requests media forking, it must provide the XCC provider with two unique remote RTP ports (nearEndAddr and farEndAddr). The XCC provider identifies the incoming connection of a call, forks both the transmit (TX) and receive (RX) packets, and sends the packets to the targeted RTP ports. The XCC provider uses the nearEndAddr element for the forked TX media stream and the farEndAddr XCC element to record the RX media stream. The two forked media streams are sent from the voice gateway in a "SEND ONLY" direction.

Media forking has the following limitations:

- Supports only voice media stream.
- Supports only IPv4 RTP forked media stream.
- Media mixing on forked media streams is not supported.
- Media negotiation is not supported on the forked media streams. In other words, the codec of the forked media stream cannot be changed. If the targeted media server supports a dynamic codec format in the forked media stream, you must configure a supported codec, such as G.711, in the voice gateway.
- Media renegotiation is not supported.
- Media forking ends when the connection is disconnected.
- Supplementary services are not supported.
- Only one media forking request per session is supported. The XCC provider rejects additional media forking request from the application.

The XCC provider updates the application on the status of the media forking by including one of the following states in the NotifyXccCallData message.

- FORK_FAILED—Setup for media forking failed. Forked RTP connections cannot be established with the targeted RTP addresses.

- FORK_STARTED—Media forking was successful. Both the TX and RX forked RTP connections are established and connected to the targeted RTP addresses.
- FORK_DONE—Media forking has completed. Both the TX and RX forked RTP connections are released.

XCC Connection

The XCC connection describes the relationship in a XCC call and the endpoint or trunk in the call. Figure 1-5 illustrates the connection states.

Figure 1-5 Connection States

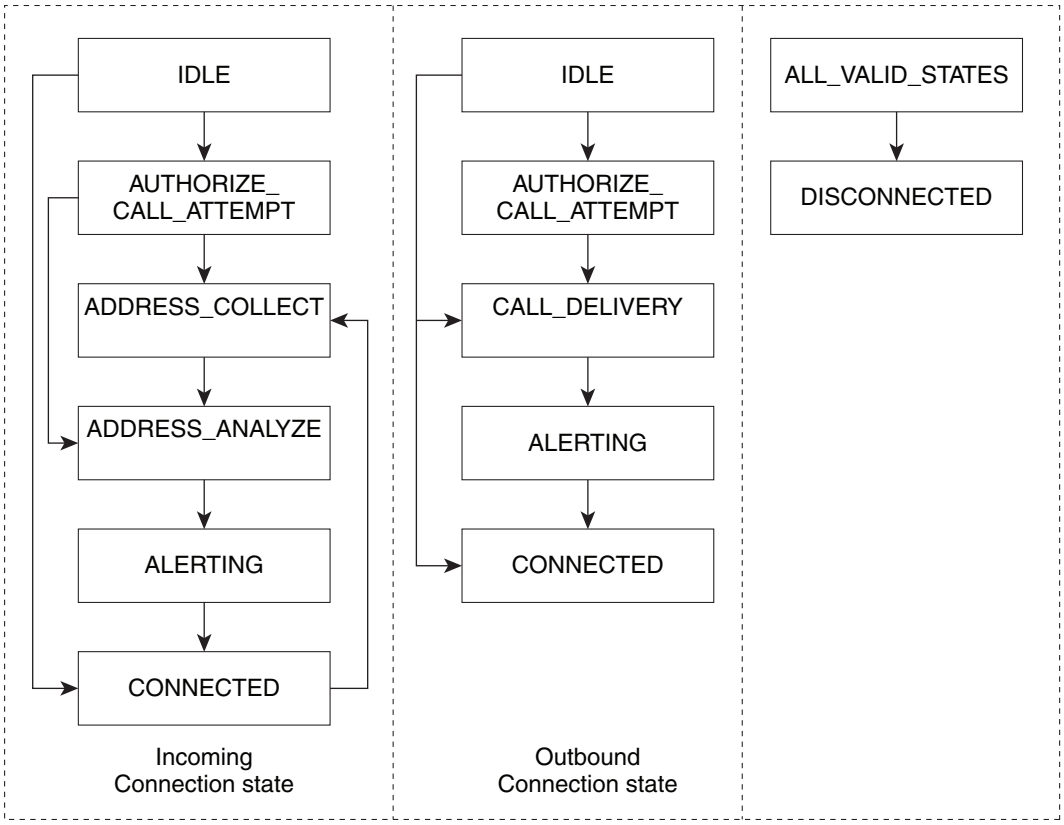


Table 1-6 describes the connection states and the activity and exchanges that can occur between the voice gateway and application when the application sets up event notifications for a particular connection state.

Table 1-6 Connection States

Connection States	Description	Activity and Messages sent between the Voice Gateway and Application
IDLE	Initial state of all new connections. In this state, the connection is not an active part of the call, but references to the call and address are valid.	**Voice Gateway** The voice gateway sends a NotifyXccConnectionData(CREATED) message for inbound calls. No messages are sent for outbound calls.
AUTHORIZE_CALL_ATTEMPT	Originating endpoint is waiting for authorization.	**Voice Gateway** The voice gateway places the call in a suspended state, sends a SolicitXccConnectionAuthorize() message, and waits for a response from the application. **Application** The application sends the ResponseXccConnectionAuthorize() message directing the gateway to either continue processing or release the call.
ADDRESS_COLLECT	Gateway is collecting information from the originating party.	No messages are sent.
ADDRESS_ANALYZE	Gateway has finished collecting the originating party information and is analyzing and translating the information according to a dial plan.	**Voice Gateway** The voice gateway places the call in a suspended state, sends a SolicitXccConnectionAddressAnalyze() message, and waits for a response from the application. **Application** The application sends either the call route back to the gateway or delegates the voice gateway to make the route selection in the ResponseXccConnectionAddressAnalyze () message.
CALL_DELIVERY	On an outbound call, the voice gateway selects the route and sends a request that a call be setup at the specified called endpoint.	No messages are sent for inbound calls. **Voice Gateway** The voice gateway sends a NotifyXccConnectionData(CREATED) and a NotifyXccConnectionData(CALL) DELIVERY) message for outbound calls.
ALERTING	Endpoint is being notified of the incoming call.	**Voice Gateway** The voice gateway sends a NotifyXccConnectionData(ALERTING) message.

Table 1-6 Connection States (continued)

Connection States	Description	Activity and Messages sent between the Voice Gateway and Application
CONNECTED	Connection and address for the call active.	**Voice Gateway** The voice gateway sends a NotifyXccConnectionData (CONNECTED) message.
DISCONNECTED	Connection is no longer active.	**Voice Gateway** The voice gateway sends a NotifyXccConnectionData(DISCONNECTED) message.

The XCC connection API is described in Table 1-7

Table 1-7 XCC Connection API

Connection	Direction	Description
RequestXccConnectionRelease	Application to XCC Provider	Message sent requesting that the connection for a call be released.
ResponseXccConnectionRelease	XCC provider to the application	Message sent in response to the application's connection release request.
SolicitXccConnectionAuthorize	XCC provider to the application	Blocking message sent requesting call authorization from the application.
SolicitXccConnectionAddressAnalyze	XCC provider to the application	Blocking message sent with address information for the application to analyze.
ResponseXccConnectionAuthorize	Application to XCC Provider	Message sent in response to the XCC provider's connection authorization request.
RequestXccConnectionAuthorizeDone	Application to XCC Provider	Message sent instructing the XCC provider to either continue processing the call or to release the call.
ResponseXccConnectionAddressAnalyze	Application to XCC Provider	Response message sent instructing the XCC provider to either continue processing the call or to release the call.
RequestXccConnectionAddressAnalyzeDone	Application to XCC Provider	Message sent when the application has completed the address analysis. The message provides information on how the provider should process the call and lists the connection event filters that the application is interested in monitoring.
ResponseXccConnectionAuthorizeDone	XCC provider to the application	Response message sent when the XCC provider has completed the application's XccConnectionAuthorizeDone request.

Table 1-7 XCC Connection API (continued)

Connection	Direction	Description
ResponseXccConnectionAddressAnalyzeDone	XCC provider to the application	Response message sent when the XCC provider has completed the application's XccConnectionAddressAnalyzeDone request.
NotifyXccConnectionData(connection_state)	XCC provider to the application	Operation-triggered message sent to notify the application of the following connection states: • CREATED • REDIRECTED • ALERTING • CONNECTED • TRANSFERRING • DISCONNECTED • HANDOFFLEAVE • HANDOFFJOIN

XSVC Provider

The extended serviceability provider (XSVC provider) monitors the health of the trunk and provides the application with real-time trunk status.

The XSVC provider can monitor both traditional public switched telephone network (PSTN) trunks and VoIP trunks. You must configure the XSVC provider and install a route listener for XSVC on the interested trunk group to begin monitoring the trunk status. The route listener communicates with the trunk group resource manager to obtain information on the trunks, including alarm information for T1/E1 trunks.

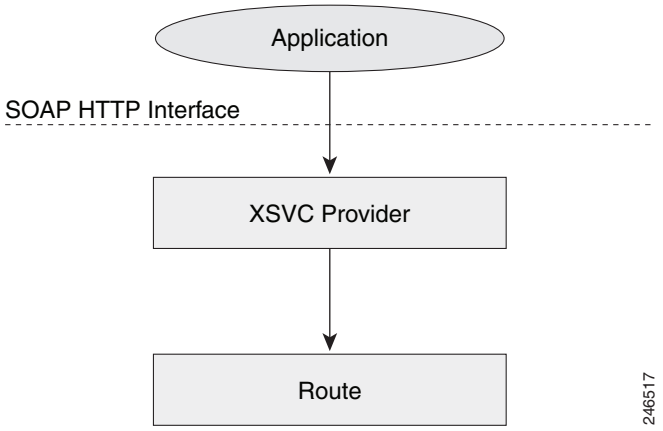
For PSTN trunks, the trunk group is a logical grouping of interfaces with the same signaling characteristics, such as DS1, FXO, or PRI interfaces. The trunk group can have more than one PRI interface and can also support FXO, but you cannot mix FXO and T1/E1 interfaces. The trunk group resource manager supports the logical configuration of trunk groups.

For VoIP trunks, the trunk manager monitors a VoIP trunks by using Internet Control Message Protocol (ICMP) pings. The trunk manager supports up to 1000 trunks.

When the application registers with the XSVC provider, the application obtains a handler that the application uses to receive snapshot information on all the routes or specific routes. The XSVC provider can support up to 8 different applications, with each application able to monitor a particular group of trunks.

Figure 1-6 illustrates the relationship between the application, XSVC route, and XSVC provider.

Figure 1-6 XSVC Provider



Characteristics of the XSVC Provider

The XSVC provider has the following characteristics:

- When the XSVC provider cannot reach the remote application, the XSVC provider discards event information messages.
- The application must register with the XSVC provider or use a snapshot to obtain the most updated trunk information.
- During the registration, the application can configure event filters for a registered session. The event filters only applies for that registered session.
- The XSVC provider reports on the current status of the trunk. The XSVC provider does not report on changes to a trunk configuration until the change has taken effect.

XSVC Provider API

When the application registers with the XSVC provider, a route listener is installed on the trunk interfaces. If filters are not specified in the registration message, the XSVC provider does not filter out any events. For the application to receive the most current trunk configuration, we recommend that you do not filter out the ROUTE_CONF_UPDATED event.

The XSVC provider API is described in Table 1-8. For additional information, see the XSVC Provider API.

Table 1-8 XSVC Provider API

XSVC Provider	Direction	Description
RequestXsvcRegister	Application to XSVC Provider	Registration request sent with event filters settings in the message.
RequestXsvcUnRegister	Application to XSVC Provider	Message sent from the application requesting to be unregistered.

Table 1-8 XSVC Provider API (continued)

XSVC Provider	Direction	Description
ResponseXsvcRegister	XSVC provider to the application	Message sent in response to the application's registration request.
ResponseXsvcUnRegister()	XSVC provider to the application	Message sent in response to the application's unregistration request.
NotifyXsvcStatus	XSVC provider to the application	Operation-triggered message sent to notify the application when the XSVC provider changes state.
SolicitXsvcProbing	XSVC provider to the application	Blocking message sent to keep the registration session alive and to check on the health of the application.
SolicitXsvcProviderUnRegister	XSVC provider to the application	Blocking message sent to inform the application that the XSVC provider has entered the shutdown state and the registration session is now unregistered.
ResponseXsvcProbing	Application to XSVC Provider	Message sent in response to the XSVC probing message.
ResponseXsvcProviderUnRegister	Application to XSVC Provider	Message sent in response to the XSVC unregistered message.

XSVC Route

With the route snapshot API, the application can request and receive a summary from the voice gateway on all the routes that are currently being monitored in a compact format. The application can also set up a filter to listen to specific routes. The application can also request that the XSVC provider send detail information for a specific route. For T1/E1 trunks, the XSVC provider sends additional information, such as channels, total available channels, alarm, and error statistics.

The XSVC Route API is described in Table 1-9.

Table 1-9 XSVC Route API

XSVC Route	Direction	Description
RequestXsvcRouteSetFilter	Application to XSVC Provider	Message specifying the routes that the application is interested in monitoring.
RequestXsvcRouteSnapshot	Application to XSVC Provider	Message requesting compact information on all monitored routes.
RequestXsvcRouteStats	Application to XSVC Provider	Message requesting statistics on specific routes.
RequestXsvcRouteData	Application to XSVC Provider	Message sent requesting detail information on specific routes.
ResponseXsvcRouteSetFilter	XSVC provider to the application	Message sent in response to the application's route filter request message.
ResponseXsvcRouteSnapshot	XSVC provider to the application	Message sent with the compact information (Name and Link information only) on all the routes that are being monitored.
ResponseXsvcRouteStats	XSVC provider to the application	Response message sent with the statistical information on a route.

Table 1-9 XSVC Route API (continued)

XSVC Route	Direction	Description
ResponseXsvcRouteData	XSVC provider to the application	Response message sent with the detailed information on a route.
NotifyXsvcRouteConfiguration	XSVC provider to the application	Operation-triggered message sent when the XSVC option is enabled or disabled on a trunk group, or if the following route configuration changes occur on a trunk group where the XSVC option is enabled: • When a new trunk or VoIP trunk is added • When a trunk or VoIP trunk is deleted • When trunks in an existing trunk group are modified • When a trunk or VoIP trunk is modified
NotifyXsvcRouteStatus	XSVC provider to the application	Operation-triggered message sent to notify the application when there is a route status change, for example when the link status changes from UP to DOWN or vice versa. The information sent is in a compact format. **Note** This event is also triggered when there is a change in the alarm status.

Alarm Definition

Table 1-10 describes the alarm definition that can be found in XSVC route messages.

Table 1-10 Alarm Definition

Alarm	Definition
NoAlarm	No alarm present
RcvFarEndLOF	Far end LOF[1] indication (a.k.a. Yellow Alarm)
XmtFarEndLOF	Near end sending LOF indication
RcvAIS	Far end sending AIS[2]
XmtAIS	Near end sending AIS
LossOfFrame	Near end LOF (a.k.a. Red Alarm)
LossOfSignal	Near end loss of signal
LoopbackState	Near end has a loop back
T16AIS	E1 TS16 AIS
RcvFarEndLOMF	Far end is sending TS16 LOMF[3]
RcvFarEndLOMF	Near end is sending TS16 LOMF
RcvTestCode	Near end detects a test code
OtherFailure	Line status that is not defined here
UnavailSigState	Near end is in an unavailable signal state

Table 1-10 Alarm Definition (continued)

Alarm	Definition
NetEquipOOS	Carrier equipment is out of service
RcvPayloadAIS	DS2 payload AIS
Ds2PerfThreshold	DS2 performance threshold

1. LOF = loss of frame.
2. AIS = alarm indication signal.
3. LOMF = loss of multiframe.

Statistics Definition

Table 1-10 defines the statistics that are collected and can be found in XSVC route messages.

Table 1-11 Statistics Definition

Statistics	Definition
LCV	Line Coding Violation Error Event
PCV	Path Coding Violation Error Event
CSS	Controlled Slip Seconds
SEFS	Severely Errored Frame Seconds
LES	Line Errored Seconds
DM	Degraded Minutes
ES	Errored Seconds
BES	Bursty Errored Seconds
SES	Severely Errored Seconds
UAS	Unavailable Seconds

XCDR Provider

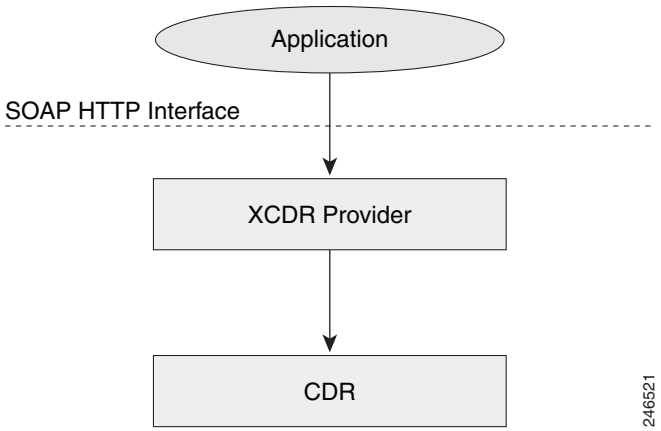
The XCDR provider sends information on a call detail record (CDR) to the registered application when a call ends. The CDR contains statistics on the call and calling party and called party information in a CSV format. The XCDR provider can support up to eight remote applications.

When the application registers with the XCDR provider, it obtains a handler that the application can use to receive CDR records. The application can choose to receive either the compact or detailed CDR format.

Note By default, the XCDR provider sends out the CDR record in a compact format to save bandwidth.

Figure 1-7 illustrates the relationship between the application, CDR, and XCDR provider.

Figure 1-7 XCDR



XCDR Provider API

The XCDR provider API is described in Table 1-12. For additional information, see the XCDR provider API.

Table 1-12 XCDR Provider API

XCDR Provider	Direction	Description
RequestXcdrRegister	Application to XCDR Provider	Registration request message. The application can specify whether it wants to receive the route configuration change notification or route status changes.
RequestXcdrUnRegister	Application to XCDR Provider	Unregistration request message sent from the application to the XCDR provider.
ResponseXcdrRegister	XCDR Provider to application	Message sent in response to the application's registration request.
ResponseXcdrUnRegister	XCDR Provider to application	Message sent in response to the application's unregistration request.
NotifyXcdrStatus	XCDR Provider to application	Operation triggered message to notify the application when the XCDR provider changes state.
SolicitXcdrProbing	XCDR Provider to application	Blocking message sent to keep the registration session alive and to check on the health of the application.
SolicitXcdrProviderUnRegister	XCDR Provider to application	Blocking message sent from the voice gateway informing the application that the XCDR provider has entered the shutdown state and the registration session is now unregistered.
ResponseXcdrProbing	Application to XCDR Provider	Message sent in response to the XCDR probing message.
ResponseXcdrProviderUnRegister	Application to XCDR Provider	Message sent in response to the XCDR unregistered message.

XCDR CDR

XCDR CDR is responsible for collecting CDR information and generating events that are sent to the application. The application can specify whether it wants the CDR record in compact or detailed format by using the RequestXcdrSetAttribute message.

The XCDR CDR API is described in Table 1-12.

Table 1-13 XCDR CDR API

XCDR CDR	Direction	Description
RequestXcdrSetAttribute	Application to XCDR Provider	Request message sent to specify the CDR format.
ResponseXcdrSetAttribute	XCDR Provider to application	Message sent in response to the application's CDR format request.
NotifyXcdrRecord	XCDR Provider to application	Message with the Call Detail Record.

Call Detail Record

For detail information on the name and order of the call detail record fields, see *CDR Accounting for Cisco IOS Voice Gateways*.

Where to Go Next

For more information on the interactions between the providers and the application and examples of messages, see the "Provider and Application Interactions" section on page A-1

For more information on the elements in the API, see the XCC, XCDR, and XSVC Provider API reference guide.

CHAPTER 2

Configuring Cisco Unified Communication IOS Services

This chapter contains the following sections:

Configuring the Router for Cisco Unified Communication IOS Services

This section describes how to configure the router to support the providers on the gateway.

Prerequisite

Cisco IOS Release 15.2(2)T

Configuring Cisco Unified Communication IOS Services on the Router

Perform this procedure to configure Cisco Unified Communication IOS services on the router.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **ip http server**
4. **ip http max-connection** *value*
5. **ip http timeout-policy idle** *seconds* **life** *seconds* **requests** *value*
6. **http client persistent**
7. **http client connection idle timeout** *seconds*
8. **uc wsapi**

9. **message-exchange max-failures** *number*
10. **probing max-failures** *number*
11. **probing interval keepalive** *seconds*
12. **probing interval negative** *seconds*
13. **source-address** *ip-address*
14. **end**

DETAILED STEPSi

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**ip http server** **Example:** `Router(conf)# ip http server`	Enables the HTTP server (web server) on the system.
Step 4	**ip http max-connection** *value* **Example:** `Router(conf)# ip http max-connection 100`	Sets the maximum number of concurrent connections to the HTTP sever that will be allowed. The default value is 5.

	Command or Action	Purpose
Step 5	**ip http timeout-policy idle** *seconds* **life** *seconds* **requests** *value* **Example:** `Router(conf)# ip http timeout-policy idle 600 life 86400 requests 86400`	Sets the characteristics that determine how long a connection to the HTTP server should remain open. The characteristics are: **idle**—The maximum number of seconds the connection will be kept open if no data is received or response data can not be sent out on the connection. Note that a new value may not take effect on any already existing connections. If the server is too busy or the limit on the life time or the number of requests is reached, the connection may be closed sooner. The default value is 180 seconds (3 minutes). **life**—The maximum number of seconds the connection will be kept open, from the time the connection is established. Note that the new value may not take effect on any already existing connections. If the server is too busy or the limit on the idle time or the number of requests is reached, it may close the connection sooner. Also, since the server will not close the connection while actively processing a request, the connection may remain open longer than the specified life time if processing is occurring when the life maximum is reached. In this case, the connection will be closed when processing finishes. The default value is 180 seconds (3 minutes). The maximum value is 86400 seconds (24 hours). **requests**—The maximum limit on the number of requests processed on a persistent connection before it is closed. Note that the new value may not take effect on any already existing connections. If the server is too busy or the limit on the idle time or the life time is reached, the connection may be closed before the maximum number of requests are processed. The default value is 1. The maximum value is 86400.
Step 6	**http client persistent** **Example:** `Router(conf)# http client persistent`	Enables HTTP persistent connections.
Step 7	**http client connection idle timeout** *seconds* **Example:** `Router(conf)# http client idle timeout 600`	Sets the number of seconds that the client waits in the idle state until it closes the connection.
Step 8	**uc wsapi** **Example:** `Router(conf)# uc wsapi`	Enters Cisco Unified Communication IOS Service configuration mode.

	Command or Action	Purpose
Step 9	**message-exchange max-failures** *number* **Example:** `Router(config-uc-wsapi)# message-exchange max failures 2`	Configures the maximum number of failed message exchanges between the application and the provider before the provider stops sending messages to the application. Range is 1 to 3. Default is 1.
Step 10	**probing max-failures** *number* **Example:** `Router(config-uc-wsapi)# probing max-failures 5`	Configures the maximum number of failed probing messages before the router unregisters the application. Range is 1 to 5. Default is 3.
Step 11	**probing interval keepalive** *seconds* **Example:** `Router(config-uc-wsapi)# probing interval 180`	Configures the interval between probing messages, in seconds. Default is 120 seconds.
Step 12	**probing interval negative** *seconds* **Example:** `Router(config-uc-wsapi)# probing interval negative 10`	Configures the interval between negative probing messages, in seconds.
Step 13	**source-addr**ess *ip-address* **Example:** `Router(config-uc-wsapi)# source-address 172.1.12.13`	Configures the IP address (hostname) as the source IP address for the UC IOS service. **Note** The source IP address is used by the provider in the NotifyProviderStatus messages.
Step 14	**end** **Example:** `Router(config-uc-wsapi)# end`	Returns to privileged EXEC mode.

Configuring the XCC Provider on the Router

Perform this procedure to configure the XCC provider on the router.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **uc wsapi**
4. **provider xcc**
5. **no shutdown**
6. **remote-url** *url*
7. **exit**
8. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**uc wsapi** **Example:** `Router(conf)# uc wsapi`	Enters Cisco Unified Communication IOS Service configuration mode.
Step 4	**provider xcc** **Example:** `Router(config-uc-wsapi)# provider xcc`	Enters XCC provider configuration mode.
Step 5	**no shutdown** **Example:** `Router(config-uc-wsapi-xcc)# no shutdown`	Activates XCC provider.
Step 6	**remote-url** *url* **Example:** `Router(config-uc-wsapi-xcc)# remote-url http://209.133.85.47:8090/my_callcontrol`	Specifies the URL (IP address and port number) that the application uses to communicate with XCC provider. The XCC provider uses the IP address and port to authenticate incoming requests.
Step 7	**exit** **Example:** `Router(config-uc-wsapi-xcc)# exit`	Exits XCC configuration mode.
Step 8	**end** **Example:** `Router(config-uc-wsapi)# end`	Returns to privileged EXEC mode.

Configuring the XSVC Provider on the Router

Perform this procedure to configure the XSVC providers on the router.

SUMMARY STEPS

1. **enable**
2. **configure terminal**

3. **uc wsapi**
4. **provider xsvc**
5. **no shutdown**
6. **remote-url** [*url-number*] *url*
7. **exit**
8. **trunk group** *name*
9. **description**
10. **xsvc**
11. **exit**
12. **voip trunk group** ***name***
13. **description**
14. **xsvc**
15. **session target ipv4:***destination-address*
16. **exit**
17. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**uc wsapi** **Example:** `Router(conf)# uc wsapi`	Enters Cisco Unified Communication IOS Service configuration mode.
Step 4	**provider xsvc** **Example:** `Router(config-uc-wsapi)# provider xsvc`	Enters XSVC provider configuration mode.
Step 5	**no shutdown** **Example:** `Router(config-uc-wsapi-xsvc)# no shutdown`	Activates XSVC provider.

	Command or Action	Purpose
Step 6	**remote-url** [*url-number*] *url* **Example:** `Router(config-uc-wsapi-xsvc)# remote-url 1 http://209.133.85.47:8090/my_route_control`	Specifies up to 8 different URLs (IP address and port number) that applications can use to communicate with the XSVC provider. The XSVC provider uses the IP address and port to authenticate incoming requests. The *url-number* identifies the unique url. Range is 1 to 8.
Step 7	**exit** **Example:** `Router(config-uc-wsapi-xsvc)# exit`	Exits XSVC configuration mode.
Step 8	**trunk group** *name* **Example:** `Router(config)# trunk group SJ_PRI`	Enters trunk-group configuration mode to define a trunk group.
Step 9	**description** **Example:** `Router(config)# description IN`	Enter a description for the trunk group. The name is passed to external application as part of XSVC status and XCC connection messages.
Step 10	**xsvc** **Example:** `Router(config-trunk-group)# xsvc`	Enables xsvc monitoring on the trunk group.
Step 11	**exit** **Example:** `Router(config-trunk-group)# exit`	Exits trunk group configuration mode.
Step 12	**voip trunk group** *name* **Example:** `Router(config)# trunk group SJ_SIP`	Enters VOIP trunk-group configuration mode to define a trunk group.
Step 13	**description** **Example:** `Router(config-voip-trk-gp)# description IN`	Enter a description for the VOIP trunk group. The name is passed to external application as part of XSVC status and XCC connection messages.
Step 14	**xsvc** **Example:** `Router(config-voip-trk-gp)# xsvc`	Enables xsvc monitoring on the VOIP trunk group.
Step 15	**session target ipv4:***destination address* **Example:** `Router(config-voip-trk-gp)# session target ipv4:9.10.31.254`	Configures the IP address of the remote router.

	Command or Action	Purpose
Step 16	**exit** **Example:** `Router(config-voip-trk-gp)# exit`	Exits VOIP trunk group configuration mode.
Step 17	**end** **Example:** `Router(config-uc-wsapi)# end`	Returns to privileged EXEC mode.

Configuring the XCDR Provider on the Router

Perform this procedure to configure the XCDR provider on the router.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **uc wsapi**
4. **provider xcdr**
5. **no shutdown**
6. **remote-url** [*url-number*] *url*
7. **exit**
8. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**uc wsapi** **Example:** `Router(conf)# uc wsapi`	Enters Cisco Unified Communication IOS Service configuration mode.
Step 4	**provider xcdr** **Example:** `Router(config-uc-wsapi)# provider xcdr`	Enters XCDR provider configuration mode.

	Command or Action	Purpose
Step 5	**no shutdown** **Example:** `Router(config-uc-wsapi-xcdr)# no shutdown`	Activates XCDR provider.
Step 6	**remote-url [***url-number***]** *url* **Example:** `Router(config-uc-wsapi-xcdr)# remote-url 1 http://209.133.85.47:8090/my_route_control`	Specifies up to eight different URLs (IP address and port number) that applications can use to communicate with the XCDR provider. The XCDR provider uses the IP address and port to authenticate incoming requests. The *url-number* identifies the unique url. Range is 1 to 8.
Step 7	**exit** **Example:** `Router(config-uc-wsapi-xcdr)# exit`	Exits XCDR configuration mode.
Step 8	**end** **Example:** `Router(config-uc-wsapi)# end`	Returns to privileged EXEC mode.

Configuration Example

The following example sets up the router for Cisco Unified Communication IOS Services. It enables the HTTP server and the XCC, XSVC, and XCDR providers. The configuration specifies the address and port that the application uses to communicate with the XCC, XSVC, and XCDR provider. It also identifies the trunk group that XSVC will be monitoring.

Note XSVC and XCDR can support up to eight different remote URLs.

```
ip http server
!
call fallback monitor
call fallback icmp-ping count 1 interval 2 timeout 100
!
uc wsapi
 source-address 10.1.1.1
 provider xcc
  remote-url http://test.com:8090/xcc
 !
 provider xsvc
  remote-url 1 http://test.com:8090/xsvc
!
 provider xcdr
  remote-url 1 http://test.com:8090/xcdr
!
trunk group pri
 xsvc

voip trunk group 1
 xsvc
 session target ipv4: 11.1.1.1
```

```
!
interface Serial0/1/0:23
isdn switch-type primary-ni
isdn incoming-voice voice
 trunk-group pri
```

Verifying and Troubleshooting Cisco Unified Communication IOS Services

Use the following show commands to gather information on the performance of the Cisco Unified Communication IOS Services:

- **show wsapi registration**
- **show wsapi http client**
- **show wsapi http server**
- **show wsapi xsvc routes**

Use the following debug commands to gather troubleshooting information on the service provider:

- **debug wsapi xcc [CR | all | function | default | detail | error | inout | event]**
- **debug wsapi xsvc [CR | all | function | default | detail | error | inout | event]**
- **debug wsapi xcdr [CR | all | function | default | detail | error | inout | event]**
- **debug wsapi infrastructure [CR | all | function | default | detail | error | inout | event]**

Command Reference

This section documents the CLI commands that are used on the router.

debug wsapi

To collect and display traces for the Cisco Unified Communication IOS services application programming interface, use the **debug wsapi** command in privileged EXEC mode. To disable debugging, use the **no** form of this command.

debug wsapi {infrastructure | xcc | xcdr | xsvc } [all | default | detail | error | event | function | inout | messages]

no debug wsapi {infrastructure | xcc | xcdr | xsvc } [all | default | detail | error | event | function | inout | messages

Syntax Description

infrastructure	Enables debugging traces on the infrastructure.
xcc	Enables debugging traces on the xcc provider.
xcdr	Enables debugging traces on the xcdr provider.
xsvc	Enables debugging traces on the xsvc provider.
all	Enables all debugging traces.
default	Enables default debugging traces.
detail	Enables detailed debugging traces.
error	Enables error debugging traces.
event	Enables event debugging traces.
function	Enables function debugging traces.
inout	Enables inout debugging traces.
messages	Enables API message traces.

Command Modes

Privileged EXEC

Command History

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

Use this command to enable debugging traces for the Cisco Unified Communicaion IOS services subsystems.

Examples

The following is the debug output from the **debug wsapi infrastructure** command for an XCC registration.

```
Router# debug wsapi infrastructure

23:25:09: //WSAPI/INFRA/wsapi_https_urlhook:
23:25:09: //WSAPI/INFRA: app_name cisco_xcc in url /cisco_xcc in port 8090
23:25:09: //WSAPI/INFRA/wsapi_https_urlhook: Exit
23:25:09: //WSAPI/INFRA/wsapi_https_post_action:
```

```
23:25:09: wsapi_https_data_read: <soapenv:Envelope
xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"><soapenv:Body><RequestXccRegister
xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0"><applicationData><name>myapp</name><url
>http://sj22lab-as2:8090/xcc</url></applicationData><blockingEventTimeoutSec>1</blockingEv
entTimeoutSec><blockingTimeoutHandle>CONTINUE_PROCESSING</blockingTimeoutHandle><connectio
nEventsFilter>CREATED AUTHORIZE_CALL REDIRECTED ALERTING CONNECTED TRANSFERRED
CALL_DELIVERY DISCONNECTED HANDOFFLEAVE
HANDOFFJOIN</connectionEventsFilter><mediaEventsFilter>MODE_CHANGE DTMF TONE_BUSY
TONE_DIAL TONE_SECOND_DIAL TONE_RINGBACK TONE_OUT_OF_SERVICE
MEDIA_ACTIVITY</mediaEventsFilter><msgHeader><transactionID>txID001</transactionID></msgHe
ader><providerData><url>http://10.1.1.1:8090/cisco_xcc</url></providerData></RequestXccReg
ister></soapenv:Body></soapenv:Envelope>
23:25:09: //WSAPI/INFRA/27/0/wsapi_https_recv:
23:25:09: //WSAPI/INFRA/27/0/txID001/wsapi_ph_request_msg_handle:
23:25:09: //WSAPI/INFRA/27/0/txID001: prov_type 0 msg_type 6 prov_state 1
23:25:09: //WSAPI/INFRA/wsapi_create_common_msg:
23:25:09: //WSAPI/INFRA/wsapi_create_common_msg: Exit
23:25:09: //WSAPI/INFRA/27/0/txID001/wsapi_send_outbound_response:
23:25:09: wsapi_dump_msg: type 8
23:25:09: transactionID txID001
23:25:09: registrationID 50674FC:XCC:myapp:9
23:25:09: ResponseXccRegister:
23:25:09: providerStatus 1
23:25:09: //WSAPI/INFRA/27/0/txID001/wsapi_send_outbound_response: Exit
23:25:09: wsapi_send_ResponseRegister:mem_mgr_mempool_free: mem_refcnt(3CA18B8)=0 -
mempool cleanup
23:25:09: //WSAPI/INFRA/27/0/txID001/wsapi_https_recv: Exit
23:25:09: wsapi_https_data_write: <?xml version="1.0" encoding="UTF-8"?><SOAP:Envelope
xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"><SOAP:Body><ResponseXccRegister
xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0"><msgHeader><transactionID>txID001</tran
sactionID><registrationID>50674FC:XCC:myapp:9</registrationID></msgHeader><providerStatus>
IN_SERVICE</providerStatus></ResponseXccRegister></SOAP:Body></SOAP:Envelope>
23:25:09: //WSAPI/INFRA/wsapi_https_post_action: Exit
```

The following is a partial debug log from the **debug wsapi xcc all** command for a call.

```
Router# debug wsapi xcc all

23:27:20: //WSAPI/XCC/check_xccp_active:177:
23:27:20: //WSAPI/XCC/provider_base_get_state:248:
23:27:20: //WSAPI/XCC/provider_base_get_registration_count:212:
23:27:20: //WSAPI/XCC/check_xccp_active:177:
23:27:20: //WSAPI/XCC/provider_base_get_state:248:
23:27:20: //WSAPI/XCC/provider_base_get_registration_count:212:
23:27:20: //WSAPI/XCC/xccp_sessStore_call_add:271:
23:27:20: //WSAPI/XCC/xccp_sessStore_get_db:145:
23:27:20: //WSAPI/XCC/xccp_session_call_add:353: xcc session successfully added
23:27:20: //WSAPI/XCC/xccp_sessStore_call_add:285: xcc call  successfully added
23:27:20: //WSAPI/XCC/check_xccp_active:177:
23:27:20: //WSAPI/XCC/provider_base_get_state:248:
23:27:20: //WSAPI/XCC/provider_base_get_registration_count:212:
23:27:20: //WSAPI/XCC/xccp_create_outbound_msg_space:677:
23:27:20: //WSAPI/XCC/xccp_sessStore_get_callData:225:
23:27:20: //WSAPI/XCC/xccp_sessStore_get_db:145:
23:27:20: //WSAPI/XCC/xccp_session_get_callData:445:
23:27:20: //WSAPI/XCC/check_xccp_active:177:
23:27:20: //WSAPI/XCC/provider_base_get_state:248:
23:27:20: //WSAPI/XCC/provider_base_get_registration_count:212:
23:27:20: //WSAPI/XCC/xccp_notify_events:434:
23:27:20: //WSAPI/XCC/xccp_queue_events:304:
23:27:20: //WSAPI/XCC/provider_base_event_new:335:
23:27:20: //WSAPI/UNKNOWN/event_base_new:267:
23:27:20: //WSAPI/XCC: magic [0xBABE] state[EVENT_STATE_ACTIVE] owner [0x1148C178]
evSize[56] debFlag[3] evHdlr[0x894D834] evHdlFree[0x894DB00]
```

```
23:27:20: //WSAPI/UNKNOWN/event_base_new:292: event base new succ
23:27:20: //WSAPI/XCC/provider_base_event_new:360: provider base eventNew success
23:27:20: //WSAPI/XCC/provider_base_add_ev_to_q:393:
23:27:20: //WSAPI/XCC/check_xccp_active:177:
23:27:20: //WSAPI/XCC/provider_base_get_state:248:
23:27:20: //WSAPI/XCC/provider_base_get_registration_count:212:
23:27:20: //WSAPI/XCC/xccp_create_outbound_msg_space:677:
23:27:20: //WSAPI/XCC/xccp_sessStore_get_callData:225:
23:27:20: //WSAPI/XCC/xccp_sessStore_get_db:145:
23:27:20: //WSAPI/XCC/xccp_session_get_callData:445:
23:27:20: //WSAPI/XCC/check_xccp_active:177:
23:27:20: //WSAPI/XCC/provider_base_get_state:248:
23:27:20: //WSAPI/XCC/provider_base_get_registration_count:212:
23:27:20: //WSAPI/XCC/xccp_solicit_events:359:
23:27:20: //WSAPI/XCC/xccp_queue_events:304:
23:27:20: //WSAPI/XCC/provider_base_event_new:335:
23:27:20: //WSAPI/UNKNOWN/event_base_new:267:
23:27:20: //WSAPI/XCC: magic [0xBABE] state[EVENT_STATE_ACTIVE] owner [0x1148C178]
evSize[56] debFlag[3] evHdlr[0x894D834] evHdlFree[0x894DB00]
23:27:20: //WSAPI/UNKNOWN/event_base_new:292: event base new succ
23:27:20: //WSAPI/XCC/provider_base_event_new:360: provider base eventNew success
23:27:20: //WSAPI/XCC/provider_base_add_ev_to_q:393:
23:27:20: //WSAPI/XCC/provider_base_process_events:444:
23:27:20: //WSAPI/XCC/xccp_handle_events:153:
23:27:20: //WSAPI/INFRA/wsapi_send_outbound_message:
23:27:20: //WSAPI/INFRA/wsapi_send_outbound_message_by_provider_info:
23:27:20: //WSAPI/XCC/wsapi_xcc_encode_outbound_msg:
23:27:20: //WSAPI/XCC/wsapi_xcc_encode_outbound_msg: Exit
23:27:20: //WSAPI/INFRA/0/1527/50875A4:319:out_url http://sj22lab-as2:8090/xcc
23:27:20: wsapi_send_outbound_message_by_provider_info: <?xml version="1.0"
encoding="UTF-8"?><SOAP:Envelope
xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"><SOAP:Body><NotifyXccConnectionData
xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0"><msgHeader><transactionID>50875A4:319</
transactionID><registrationID>50674FC:XCC:myapp:9</registrationID></msgHeader><callData><c
allID>9</callID><state>ACTIVE</state></callData><connData><connID>1527</connID><state>IDLE
</state></connData><event><created><connDetailData><connData><connID>1527</connID><state>I
DLE</state></connData><guid>7A1E678F-8259-11E0-8FF1-D29982DCA129</guid><callingAddrData><t
ype>E164</type><addr>5522101</addr></callingAddrData><calledAddrData><type>E164</type><add
r>6001</addr></calledAddrData><origCallingAddrData><type>E164</type><addr>5522101</addr></
origCallingAddrData><origCalledAddrData><type>E164</type><addr>6001</addr></origCalledAddr
Data><connIntfType>CONN_SIP</connIntfType><mediaData><type>VOICE</type></mediaData><connIn
tf>1.3.45.2</connIntf><connDirectionType>INCOMING</connDirectionType></connDetailData></cr
eated></event></NotifyXccConnectionData></SOAP:Body></SOAP:Envelope>
23:27:20: //WSAPI/INFRA/0/1527/50875A4:319/wsapi_send_outbound_message_by_provider_info:
Exit
.
.
.
```

message-exchange max-failures

To configure the maximum number of failed message that is exchanged between the application and the provider before the provider stops sending messages to the application, use the **message-exchange max-failures** command. To reset the maximum to the default number, use the **no** form of this command.

message-exchange max-failures *number*

no message-exchange max-failures *number*

Syntax Description

number	Maximum number of messages allowed before the service provider stops sending messages to the application. Range is from 1 to 3. Default is 1.

Command Default

The default is 1.

Command Modes

uc wsapi configuration mode

Command History

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

Use this command to set the maximum number of messages that can fail before the system determines that the application is unreachable and the service provider stops sending messages to the application.

Examples

The following example sets the maximum number of failed messages to 2.

```
Router(config)# uc wsapi
Router(config-uc-wsapi)# message-exchange max-failures 2
```

Related Commands

Command	Description
probing interval	Sets the time interval between probing messages.
probing max-failure	Sets the number of messages that the system will send without receiving a reply before the system unregisters the application.

probing interval

To configure the time interval between probing messages sent by the router, use the **probing interval** command. To reset the time interval to the default number, use the **no** form of this command.

probing interval [keepalive | negative] *seconds*

no probing interval keepalive [negative] *seconds*

Syntax Description

keepalive	(optional) Configures the time interval between probing messages when the session is in a keepalive state. Range is from 1 to 255 seconds. Default is 5 seconds.
negative	(optional) Configures the time interval between probing messages when the session is in a negative state. Range is from 1 to 20 seconds. Default is 5 seconds.
seconds	Number of seconds between probing message.

Defaults

The default is 120 seconds between probing messages when the session is in a normal state and 5 seconds between probing messages when the session is in a negative state.

Command Modes

uc wsapi configuration mode.

Command History

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

Use this command to configure the time interval between probing messages sent by the router.

Examples

The following example sets an interval of 180 seconds during a normal session and 10 seconds when the session is in a negative state:

```
Router(config)# uc wsapi
Router(config-uc-wsapi)# probing interval keepalive 180
Router(config-uc-wsapi)# probing interval negative 10
```

Related Commands

Command	Description
message-exchange	Sets the maximum number of failed message responses before the provider stops sending messages.
probing max-failure	Sets the number of messages that the system will send without receiving a reply before the system unregisters the application.

probing max-failures

To configure the maximum number of probing messages that the application fails to respond to before the system stops the session and unregisters the application, use the **probing max-failures** command. To reset the maximum to the default number, use the **no** form of this command.

probing max-failures *number*

no probing max-failures *number*

Syntax Description

number	Maximum number of messages allowed before the system stops the session and unregisters the application. Range is from 1 to 5. Default is 3.

Command Default

The default is 3.

Command Modes

uc wsapi configuration mode.

Command History

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

Use this command to set the maximum number of probing messages sent by the system that the application does not respond to before the system stops the session and unregisters the application session.

Examples

The following example sets the maximum number of failed messages to 5.

```
Router(config)# uc wsapi
Router(config-uc-wsapi)# probing max-failures 5
```

Related Commands

Command	Description
message-exchange	Sets the maximum number of failed message responses before the provider stops sending messages.
probing interval	Sets the time interval between probing messages.

provider

To configure and enable a service provider, use the **provider** command. To remove the provider, use the **no** form of this command.

provider [XCC | XSVC | XCDR]

no provider [XCC | XSVC | XCDR]

Syntax Description

XCC	(optional) Enables the XCC service provider.
XSVC	(optional) Enables the XSVC service provider.
XCDR	(optional) Enables the XCDR service provider.

Defaults

No default behavior.

Command Modes

uc wsapi configuration mode

Command History

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

Use this command to enable the service provider.

Examples

The following example enables the XCC service provider.

```
Router(config)# uc wsapi
Router(config-uc-wsapi)# provider xcc
Router(config-uc-wsapi-xcc)# no shutdown
```

Related Commands

Command	Description
remote-url	Specifies the URL of the application.
source-address	Specifies the IP address of the provider.
uc wsapi	Enters Cisco Unified Communication IOS services configuration mode.

remote-url

To configure the url of the application that will be used by the service provider, use the **remote-url** command. The provider will use this url to authenticate and commnunicate with the application. To delete the configured url, use the **no** form of this command.

remote-url [*url-number*] *url*

no remote-url [*url-number*] *url*

Syntax Description

url-number	(optional) URL number. Range is from 1 to 8.
url	Specifies the URL that the service provider will be using in the messages.

Command Default

None

Command Modes

uc wsapi configuration mode.

Command History

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

Use this command to configure the remote URL (application) that the service provider uses in messages.

Examples

The following exampleconfigures the remote url that the the xcc service provider will use in messages.

```
Router(config)# uc wsapi
Router(config-uc-wsapi)# provider xcc
Router(config-uc-wsapi-xcc)# no shutdown
Router(config-uc-wsapi-xcc)# remote-url 1 http://209.133.85.47:8090/my_route_control
```

Related Commands

Command	Description
provider	Enables a provider service.
source-address	Specifies the IP address of the provider.
uc wsapi	Enters Cisco Unified Communication IOS services configuration mode.

show call media forking

To display currently active media forking sessions, use the **show call media forking** command in user EXEC or privileged EXEC mode.

show call media forking

Syntax Description

This command has no arguments or keywords.

Command Modes

User EXEC (>)

Privileged EXEC (#)

Command History

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

Use this command to verify that media forking was successful for relevant anchor legs.

Examples

The following example is a sample output from the **show call media forking** command.

```
Router# show call media forking

Warning: Output may be truncated if sessions are added/removed concurrently!

Session    Call     n/f  Destination (port address)
7          6        far  1234 1.5.35.254
8          6        near 5678 1.5.35.254
```

Table 2-1 describes the fields that are displayed.

Table 2-1 Show Call Media Forking Field Descriptions

Field	Description
Session	Session Identifier.
Call	Call Leg identifier in hexadecimal. It must match the Call ID from the show call leg active command.
n/f	Direction (Near End or Far End) of the voice stream that was forked.
Destination (port address)	Destination for the forked packets. It consists of the folllowing: • RTP Port • IP Address

show voip trunk group

To display the internal list of voip trunk groups, use the **show voip trunk group** command in user EXEC or privileged EXEC mode.

show voip trunk group

Syntax Description

This command has no arguments or keywords.

Command Modes

User EXEC (>)

Privileged EXEC (#)

Command History

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

Use this command to display VOIP trunk groups.

Examples

The following example is a sample output from the **show voip trunk group** command.

```
Router# show voip trunk group
=====================================================
name:           1
protocol:       cisco
ip:             1.3.45.2
xsvc:           TRUE
```

show wsapi

To display information on the Cisco Unified Communication IOS services, including registration, statistics, and route information, use the **show wsapi** command in user EXEC or privileged EXEC mode.

show wsapi {http-client | http-server | registration {all | xcc | xcdr | xsvc } | xsvc route }

Syntax Description

http-client	Displays the statistics that have been collected on the http client interface.
http-server	Displays the statistics that have been collected on the http server interface.
registration	Displays the currently registered applications on the WSAPI subsystem.
all	Displays all registered applications.
xcc	Displays the applications that are registered to the XCC provider.
xcdr	Displays the applications that are registered to the XCDR provider.
xsvc	Displays the applications that are registered to the XSVC provider.
xsvc route	Displays the internal route information in the XSVC provider.

Command Modes

User EXEC

Privileged EXEC

Command History

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

Use this command to display information on the Cisco Unified Communication IOS services.

Examples

The following example is a sample output from the **show wsapi http-client** command.

```
Router# show wsapi http-client

WSAPI Outgoing Notify/Solicit Message Statistics
========================================

wsapi_show_httpc_callback_context_invalid: 0
wsapi_show_httpc_callback_context_error: 0
wsapi_show_httpc_callback_no_reg: 5
wsapi_show_httpc_callback_notify_OK: 85
wsapi_show_httpc_callback_notify_error: 0
wsapi_show_httpc_callback_client_error: 0
wsapi_show_httpc_callback_error: 7
wsapi_show_httpc_callback_client_error: 0
wsapi_show_httpc_callback_decode_error: 28
wsapi_show_httpc_callback_no_txID: 0
wsapi_show_httpc_callback_OK: 655
wsapi_show_httpc_create_msg_error: 0
wsapi_show_httpc_context_active: 0
wsapi_tx_context_freeq depth: 4
```

The following example is a sample output from the **show wsapi http-server** command.

```
Router# show wsapi http-server

WSAPI Incoming Request Message Statistics
=========================================

wsapi_show_https_urlhook: 23
wsapi_show_https_post_action: 23
wsapi_show_https_post_action_fail: 0
wsapi_show_https_xml_fault: 0
wsapi_show_https_post_action_done: 23
wsapi_show_https_service_timeout: 0
wsapi_show_https_send_error: 0
wsapi_show_https_invalid_context: 0
wsapi_show_https_data_active: 0
wsapi_https_data_q depth: 1
wsapi_show_https_internal_service_error: 0
wsapi_show_https_service_unavailable_503: 0
wsapi_show_https_not_found_404: 0
wsapi_show_https_registration_success: 9
wsapi_show_https_not_registered: 0
wsapi_show_https_registration_auth_fail: 1
wsapi_show_https_registration_fail: 0
wsapi_show_https_un_registered: 0
```

The following example is a sample output from the **show wsapi registration all** command.

```
Router# show wsapi registration all

Provider XCC
=====================================================
registration
  id: 4FA11CC:XCC:myapp:5
  appUrl:http://sj22lab-as2:8090/xcc
  appName: myapp
  provUrl: http://10.1.1.1:8090/cisco_xcc
  prober state: STEADY
  connEventsFilter:
CREATED|AUTHORIZE_CALL|ADDRESS_ANALYZE|REDIRECTED|ALERTING|CONNECTED|TRANSFERRED|CALL_DELI
VERY|DISCONNECTED|HANDOFF_JOIN|HANDOFF_LEAVE
  mediaEventsFilter:
DTMF|MEDIA_ACTIVITY|MODE_CHANGE||TONE_DIAL|TONE_OUT_OF_SERVICE|TONE_RINGBACK|TONE_SECOND_D
IAL
  blockingEventTimeoutSec: 1
  blockingTimeoutHandle: CONTINUE_PROCESSING


Provider XSVC
=====================================================
registration index: 2
  id: 4FA0F8C:XSVC:myapp:3
  appUrl:http://sj22lab-as2:8090/xsvc
  appName: myapp
  provUrl: http://10.1.1.1:8090/cisco_xsvc
  prober state: STEADY
  route filter:
  event filter: off

Provider XCDR
=====================================================
registration index: 1
  id: 4FA10A0:XCDR:myapp:1
  appUrl:http://sj22lab-as2:8090/xcdr
```

```
  appName: myapp
  provUrl: http://10.1.1.1:8090/cisco_xcdr
  prober state: STEADY
  cdr format: COMPACT
  event filter: off
```

The following example is a sample output from the **show wsapi xsvc route** command.

```
Router# show wsapi xsvc route

Route SANJOSE_SIP
=====================================================
 Type: VOIP
 Description: OUT
 Filter:
 Trunk:
        Trunk Name:     1.3.45.2
        Trunk Type:     SIPV2
        Trunk Status:   UP

Route SANJOSE_PRI
=====================================================
 Type: PSTN
 Description: IN
 Filter:
 Trunk:
        Trunk Name:     Se0/1/0:23
        Trunk Type:     ISDN PRI
        Trunk Status:   UP
        Total channels  2
        Channel bitmap  0x01FFFFFE 1-24
        Link bitmap     0x00000006
       Alarm   0x00000001
        Time elapsed    516
        Interval       92
        CurrentData
     0 Line Code Violations, 0 Path Code Violations
     0 Slip Secs, 0 Fr Loss Secs, 0 Line Err Secs, 0 Degraded Mins
     0 Errored Secs, 0 Bursty Err Secs, 0 Severely Err Secs, 0 Unavail Secs
        TotalData

     49 Line Code Violations, 7 Path Code Violations,
     0 Slip Secs, 1 Fr Loss Secs, 1 Line Err Secs, 0 Degraded Mins,
     0 Errored Secs, 0 Bursty Err Secs, 0 Severely Err Secs, 2 Unavail Secs

        Trunk Name:     Se0/1/1:23
        Trunk Type:     ISDN PRI
        Trunk Status:   UP
        Total channels  2
        Channel bitmap  0x01FFFFFE 1-24
        Link bitmap     0x00000006
       Alarm   0x00000001
        Time elapsed    516
       Interval        92
        CurrentData
     0 Line Code Violations, 0 Path Code Violations
     0 Slip Secs, 0 Fr Loss Secs, 0 Line Err Secs, 0 Degraded Mins
     0 Errored Secs, 0 Bursty Err Secs, 0 Severely Err Secs, 0 Unavail Secs
        TotalData

     42 Line Code Violations, 4 Path Code Violations,
     0 Slip Secs, 1 Fr Loss Secs, 1 Line Err Secs, 0 Degraded Mins,
     0 Errored Secs, 0 Bursty Err Secs, 0 Severely Err Secs, 2 Unavail Secs
```

source-address (uc-wsapi)

To specify the source IP address or hostname for the Cisco Unified Communication IOS services in the NotifyProviderStatus message, use the **source-address** command in uc wsapi configuration mode. To disable the router from sending NotifyProviderStatus message, use the **no** form of this command.

source-address *ip-address*

no source-address

Syntax Description

ip-address	IP address identified as the source address by the service provider in the NotifyProviderStatus message.

Defaults

No IP address.

Command Modes

uc wsapi

CommandHistory

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

This command enables the service provider on the router to send messages to the application via the NotifyProvicerStatus message.

Examples

The following example shows how to set the IP source address and port:

```
Router(config)# uc wsapi
Router(config-register-global)# source-address 172.1.12.13
```

Related Commands

Command	Description
provider	Enables a provider service.
remote-url	Specifies the URL of the application.
uc wsapi	Enters Cisco Unified Communication IOS services configuration mode.

uc wsapi

To configure the Cisco Unified Communication IOS services environment for a specific application, use the **uc wsapi** command.

uc wsapi

Syntax Description

This command has no arguments or keywords.

Command Default

None

Command Modes

EXEC mode.

Command History

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

Use this command to enter the Cisco Unified Communication IOS services configuration environment.

Examples

The following example enters the Cisco Unified Communication IOS services configuration environment.

```
Router(config)# uc wsapi
Router(config-uc-wsapi)#
```

Related Commands

Command	Description
provider	Enables a provider service.

voip trunk group

To define or modify a VOIP trunk group and to enter trunk group configuration mode, use the **voip trunk group** command in global configuration mode. To delete the VOIP trunk group, use the **no** form of this command.

voip trunk group *name*

no voip trunk group *name*

Syntax Description

name	Name of the voip trunk group. Valid names contain a maximum of 63 alphanumeric characters.

Command Default

No voip trunk group is defined.

Command Modes

Global configuration

Command History

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

Use the **voip trunk group** command to define the VOIP trunk and extend serviceability to the trunk. By default, the session protocol of the IP trunk is h323. Up to 1000 trunk groups can be configured on the gateway provided that the gateway has sufficient memory to store the profiles

Examples

The following example enables creates a VOIP trunk group and enables monitoring.

```
Router(config)# voip trunk group siptrk1
Router(config-voip-trk)# session protocol sipv2
Router(config-voip-trk)# target ipv4: 10.1.1.15
Router(config-voip-trk)# xsvc
```

Command	Description
show voip trunk group	Displays the internal list of voip trunk groups.
xsvc	Enables monitoring on the trunk.

xsvc

To add support for extended serviceability (xsvc) on TDM, (ISDN-PRI/BRI, DS0-group, analog voice-port) voice interfaces, which are defined as a trunk group, use the **xsvc** command. To disable support for extended serviceability, use the **no** form of this command.

xsvc

no xsvc

Syntax Description

This command has no arguments or keywords.

Command Default

Extended serviceability is disabled on trunk groups.

Command Modes

Trunk group configuration

Command History

Release	Modification
15.2(2)T	This command was introduced.

Usage Guidelines

Use this command to add support for extended serviceability on voice interfaces which are defined as a trunk group.

Examples

The following example enables monitoring on a trunk group.

```
Router(config)# trunk group tdm-tg1
Router(config-trunk-group)# xsvc
```

Related Commands

Command	Description
provider	Enables a provider service.

APPENDIX A

Provider and Application Interactions

This section describes the interaction and sequence of messages that take place between the providers on the voice gateway and the application.

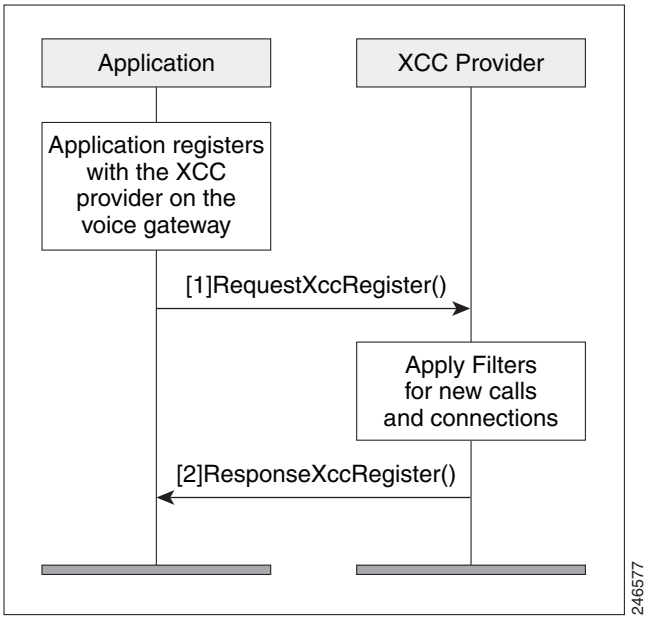
XCC

This section describes some of the interactions that takes place between the XCC provider and the application.

Interaction Between the XCC Provider and Application

Figure A-1 shows the interaction and the sequence of messages that are exchanged between the application and the XCC provider during registration.

Figure A-1 Message interaction when the application registers with XCC Provider



Message Examples

This section provides examples of message exchanges between the application and the XCC provider.

Example of a Registration Message Exchange

The following is an example of the RequestXccRegister message sent by the application requesting registration and setting up the connection event and media event filters.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Body>
    <RequestXccRegister xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <applicationData>
        <name>myapp</name>
        <url>http://test.com:8090/xcc</url>
      </applicationData>
      <blockingEventTimeoutSec>1</blockingEventTimeoutSec>
      <blockingTimeoutHandle>CONTINUE_PROCESSING</blockingTimeoutHandle>
      <connectionEventsFilter>CREATED AUTHORIZE_CALL ADDRESS_ANALYZE REDIRECTED ALERTING
CONNECTED TRANSFERRED CALL_DELIVERY DISCONNECTED HANDOFFLEAVE
HANDOFFJOIN</connectionEventsFilter>
      <mediaEventsFilter>MODE_CHANGE DTMF TONE_BUSY TONE_DIAL TONE_SECOND_DIAL
TONE_RINGBACK TONE_OUT_OF_SERVICE MEDIA_ACTIVITY</mediaEventsFilter>
      <msgHeader>
        <transactionID>11111d</transactionID>
      </msgHeader>
      <providerData>
        <url>http://10.1.1.1:8090/cisco_xcc</url>
      </providerData>
    </RequestXccRegister>
  </soapenv:Body>
</soapenv:Envelope>
```

The following is an example of a ResponseXccRegister message sent from the XCC provider in response to the application's registration request. The registration ID is used in all messages during the registered session:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <ResponseXccRegister xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>11111d</transactionID>
        <registrationID>152E034C:XCC:myapp:5</registrationID>
      </msgHeader>
      <providerStatus>IN_SERVICE</providerStatus>
    </ResponseXccRegister>
  </SOAP:Body>
</SOAP:Envelope>
I/O warning : failed to load external entity "ResponseXCCRegister.txt"
mcebu-reg-ex2:428>  xmllint --format xmlResponseXCCRegister > ResponseXCCRegister.txt
mcebu-reg-ex2:429> more ResponseXCCRegister.txt
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <ResponseXccRegister xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>11111d</transactionID>
        <registrationID>152E034C:XCC:myapp:5</registrationID>
      </msgHeader>
      <providerStatus>IN_SERVICE</providerStatus>
    </ResponseXccRegister>
```

```
  </SOAP:Body>
</SOAP:Envelope
```

Example of a Change in Service Message

The following is an example of a NotifyXccStatus message sent from the gateway when the XCC shuts down.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <NotifyXccProviderStatus xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>43257C78:F4</transactionID>
      </msgHeader>
      <applicationData>
        <url>http://mcebu-reg-ex2.cisco.com:8090/xcc</url>
      </applicationData>
      <providerData>
        <url>http://172.19.149.185:8090/cisco_xcc</url>
      </providerData>
      <providerStatus>SHUTDOWN</providerStatus>
    </NotifyXccProviderStatus>
  </SOAP:Body>
</SOAP:Envelope>
```

Example of the Application Requesting to be Unregister

The following is an example of a RequestXccUnRegister message sent from an application when it no longer needs the provider's services.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <SolicitXccProviderUnRegister xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>152EF0C4:8F</transactionID>
        <registrationID>152E034C:XCC:myapp:5</registrationID>
      </msgHeader>
    </SolicitXccProviderUnRegister>
  </SOAP:Body>
</SOAP:Envelope>
```

Example of a Keepalive Probing Message

The following is an example of the SolicitXccProbing message sent from the XCC provider to maintain an active registration session.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <SolicitXccProbing xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>152EC69C:89</transactionID>
        <registrationID>152E034C:XCC:myapp:5</registrationID>
      </msgHeader>
      <sequence>1</sequence>
      <interval>5</interval>
      <failureCount>0</failureCount>
      <registered>true</registered>
      <providerStatus>IN_SERVICE</providerStatus>
    </SolicitXccProbing>
  </SOAP:Body>
</SOAP:Envelope>
```

The following is an example of the ResponseXccProbing message sent from the application responding to the XCC provider probing message.

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Header/>
  <soapenv:Body>
    <ResponseXccProbing xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <registrationID>62199E50:XCC:myapp:34</registrationID>
        <transactionID>621B7310:346</transactionID>
      </msgHeader>
      <sequence>1</sequence>
    </ResponseXccProbing>
  </soapenv:Body>
</soapenv:Envelope>
```

Example of the Provider Shutting Down

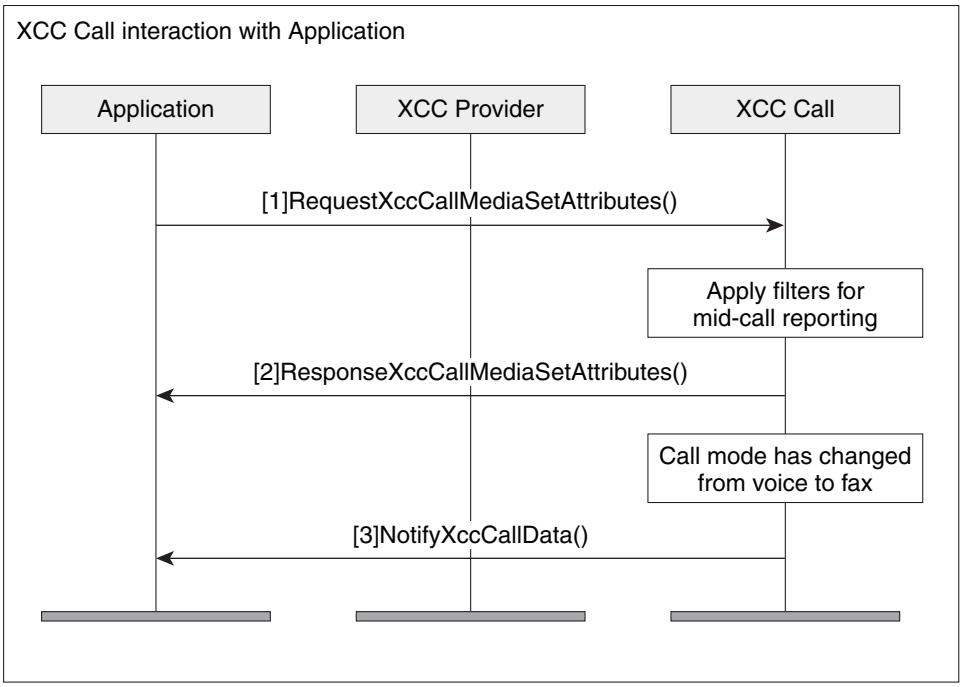
The following is an example of the SolicitXccProviderUnRegister message sent from the XCC provider when it enters the shutdown state.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <SolicitXccProviderUnRegister xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>152EF0C4:8F</transactionID>
        <registrationID>152E034C:XCC:myapp:5</registrationID>
      </msgHeader>
    </SolicitXccProviderUnRegister>
  </SOAP:Body>
</SOAP:Envelope>
```

Interaction Between the Application, XCC Provider, and XCC Call

Figure A-2 shows the interaction between the application, XCC provider, and XCC call for a call and the sequence of messages that are exchanged between the application and the XCC provider.

Figure A-2 Message interaction when a call comes in



Message Examples

This section provides examples of message exchanges between the application and the XCC provider during a call.

Example of the Application Setting Call Media Attributes.

The following is an example of a RequestXccCallMediaSetAttributes message sent from application notifying the provider of the media attributes for a call.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Body>
    <RequestXccCallMediaSetAttributes xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <callID>6</callID>
      <mediaForking>
        <farEndAddr>
          <ipv4>1.3.45.155</ipv4>
          <port>32599</port>
        </farEndAddr>
        <nearEndAddr>
          <ipv4>1.3.45.155</ipv4>
          <port>32598</port>
        </nearEndAddr>
      </mediaForking>
      <msgHeader>
        <registrationID>D3868:XCC:myapp:5</registrationID>
        <transactionID>D5494:5B</transactionID>
      </msgHeader>
    </RequestXccCallMediaSetAttributes>
  </soapenv:Body>
</soapenv:Envelope>
```

The following is an example of the ResponseXccCallMediaSetAttributes message sent from the a XCC provider in response to the application's media set attribute request.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <ResponseXccCallMediaSetAttributes xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>D5494:5B</transactionID>
        <registrationID>D3868:XCC:myapp:5</registrationID>
      </msgHeader>
    </ResponseXccCallMediaSetAttributes>
  </SOAP:Body>
</SOAP:Envelope>
```

Example of a Change in Call Mode

The following is an example of a NotifyXccCallData message sent from the XCC provider notifying the application that the call mode has changed from modem to fax mode.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http:// www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <NotifyXccCallData>
      <msgHeader>
        <transactionID>2336EF94:BC</transactionID>
        <registrationID>23362C88:XCC:myapp:7</registrationID>
      </msgHeader>
      <callData>
        <callID>8</callID>
        <state>ACTIVE</state>
      </callData>
      <mediaEvent>
        <modeChange>
          <old>MODEM</old>
          <new>FAX</new>
        </modeChange>
      </mediaEvent>
    </NotifyXccCallData>
  </SOAP:Body>
</SOAP:Envelope>
```

Example of a DTMF Detection

The following is an example of a NotifyXccCallData message sent from the XCC provider notifying the application that the number 1 digit on the keypad has been pressed.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <NotifyXccCallData>
      <msgHeader>
        <transactionID>491100E4:2E5</transactionID>
        <registrationID>4910E328:XCC:myapp:29</registrationID>
      </msgHeader>
      <callData>
        <callID>38</callID>
        <state>ACTIVE</state>
      </callData>
      <mediaEvent>
        <DTMF>
          <digit>1</digit>
          <dateTime>*01:35:04.111 UTC Sun Oct 4 1970</dateTime>
```

```
        </DTMF>
      </mediaEvent>
    </NotifyXccCallData>
  </SOAP:Body>
</SOAP:Envelope>
```

Example of Call Media Forking

The following is an example of a RequestXccCallMediaForking message sent from the application requesting that the media stream for the call session be forked. The application must include two unique RTP ports—nearEndAddr element for the forked TX media stream and the farEndAddr XCC element for the RX media stream

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Body>
    <RequestXccCallMediaForking xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <action>
        <enableMediaForking>
          <farEndAddr>
            <ipv4>1.3.45.155</ipv4>
            <port>32599</port>
          </farEndAddr>
          <nearEndAddr>
            <ipv4>1.3.45.155</ipv4>
            <port>32598</port>
          </nearEndAddr>
        </enableMediaForking>
      </action>
      <callID>8</callID>
      <msgHeader>
        <registrationID>4C21504:XCC:myapp:3</registrationID>
        <transactionID>4C23C6C:2FE</transactionID>
      </msgHeader>
    </RequestXccCallMediaForking>
  </soapenv:Body>
</soapenv:Envelope>
```

The following is an example of the NotifyXccCallData message sent from the XCC provider to the application with information on the status of the media forking.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <NotifyXccCallData>
      <msgHeader>
        <transactionID>4C252A4:2FF</transactionID>
        <registrationID>4C21504:XCC:myapp:3</registrationID>
      </msgHeader>
      <callData>
        <callID>8</callID>
        <state>ACTIVE</state>
      </callData>
      <mediaEvent>
        <mediaForking>
          <mediaForkingState>STARTED</mediaForkingState>
        </mediaForking>
      </mediaEvent>
    </NotifyXccCallData>
  </SOAP:Body>
</SOAP:Envelope>
```

The following is an example of the ResponseXccCallMediaForking message sent from the XCC provider in response to the application's media forking request.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <ResponseXccCallMediaForking xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>4C23C6C:2FE</transactionID>
        <registrationID>4C21504:XCC:myapp:3</registrationID>
      </msgHeader>
    </ResponseXccCallMediaForking>
  </SOAP:Body>
</SOAP:Envelope
```

Interaction Between the Application and XCC Connection

The following section describes the interaction between the application, XCC provider and XCC Connection.

Examples of XCC Message Exchange in the Connection State

The following is an example of a notification message sent from the XCC provider notifying the application of a connection creation event.

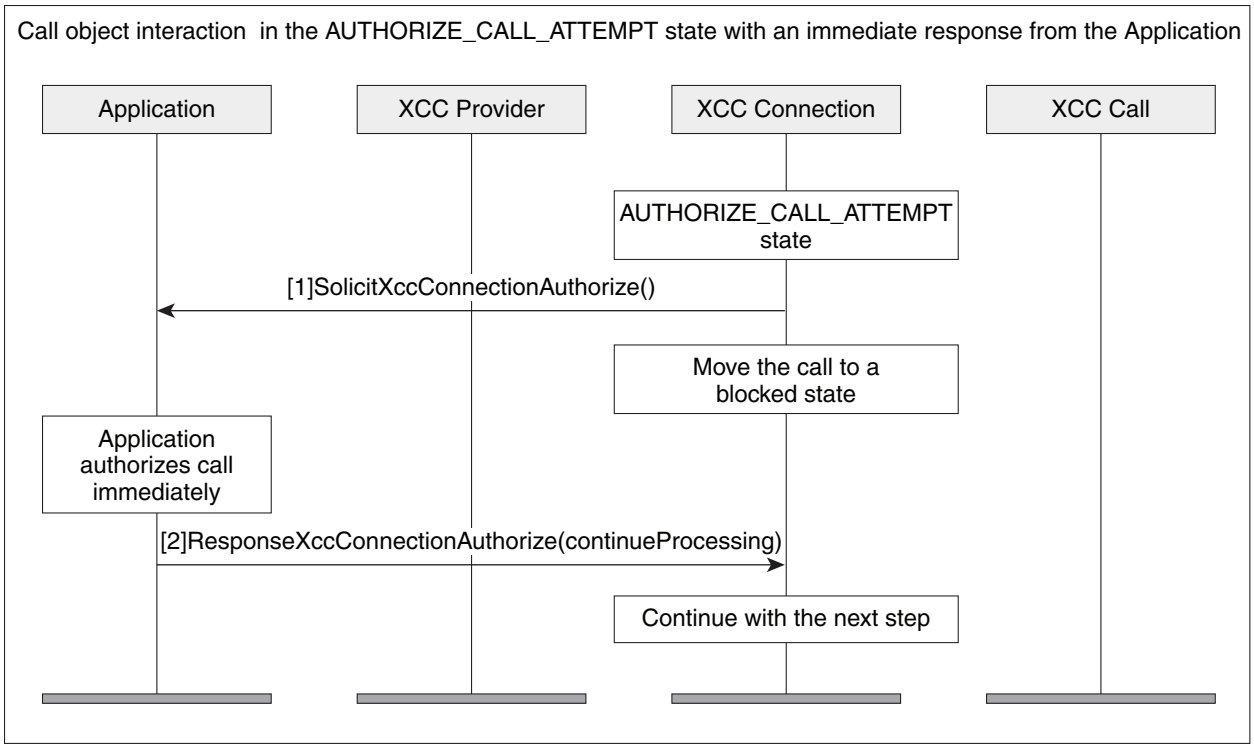
```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <NotifyXccConnectionData xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>152E6854:69</transactionID>
        <registrationID>152E034C:XCC:myapp:5</registrationID>
      </msgHeader>
      <callData>
        <callID>5</callID>
        <state>ACTIVE</state>
      </callData>
      <connData>
        <connID>30</connID>
        <state>IDLE</state>
      </connData>
      <event>
        <created>
          <connDetailData>
            <connData>
              <connID>30</connID>
              <state>IDLE</state>
            </connData>
            <guid>DDAAE040-7F44-11E0-831A-D2E9BAD25129</guid>
            <callingAddrData>
              <type>E164</type>
              <addr>3901</addr>
            </callingAddrData>
            <calledAddrData>
              <type>E164</type>
              <addr>2002</addr>
            </calledAddrData>
            <origCallingAddrData>
              <type>E164</type>
```

```
            <addr>3901</addr>
          </origCallingAddrData>
          <origCalledAddrData>
            <type>E164</type>
            <addr>2002</addr>
          </origCalledAddrData>
          <connIntfType>CONN_SIP</connIntfType>
          <mediaData>
            <type>VOICE</type>
          </mediaData>
          <connIntf>9.10.31.254</connIntf>
          <routeName>SANJOSE_SIP</routeName>
          <routeDescription>IN</routeDescription>
          <connDirectionType>INCOMING</connDirectionType>
        </connDetailData>
      </created>
    </event>
  </NotifyXccConnectionData>
 </SOAP:Body>
</SOAP:Envelope
```

Interaction for Call Authorization with an Immediate Response

Figure A-3 illustrates the call interaction when an application responds immediately to a call authorization solicit message from the XCC provider.

Figure A-3 Call Interaction when the application responds immediately to a call



The following example is the SolicitXccConnectionAuthorize message sent from the XCC provider asking for authorization to continue processing the call.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <SolicitXccConnectionAuthorize xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>152E6854:6A</transactionID>
        <registrationID>152E034C:XCC:myapp:5</registrationID>
      </msgHeader>
      <callData>
        <callID>5</callID>
        <state>ACTIVE</state>
      </callData>
      <connDetailData>
        <connData>
          <connID>30</connID>
          <state>AUTHORIZE_CALL_ATTEMPT</state>
        </connData>
        <guid>DDAAE040-7F44-11E0-831A-D2E9BAD25129</guid>
        <callingAddrData>
          <type>E164</type>
          <addr>3901</addr>
        </callingAddrData>
        <calledAddrData>
          <type>E164</type>
          <addr>2002</addr>
        </calledAddrData>
        <origCallingAddrData>
          <type>E164</type>
          <addr>3901</addr>
        </origCallingAddrData>
        <origCalledAddrData>
          <type>E164</type>
          <addr>2002</addr>
        </origCalledAddrData>
        <connIntfType>CONN_SIP</connIntfType>
        <mediaData>
          <type>VOICE</type>
        </mediaData>
        <connIntf>9.10.31.254</connIntf>
            <routeName>SANJOSE_SIP</routeName>
            <routeDescription>IN</routeDescription>
        <connDirectionType>INCOMING</connDirectionType>
      </connDetailData>
    </SolicitXccConnectionAuthorize>
  </SOAP:Body>
</SOAP:Envelope>
```

Upon authentication, the application immediately sends a response. The following example is the response message (ResponseXccConnectionAuthorize) from the application to continue processing the call.

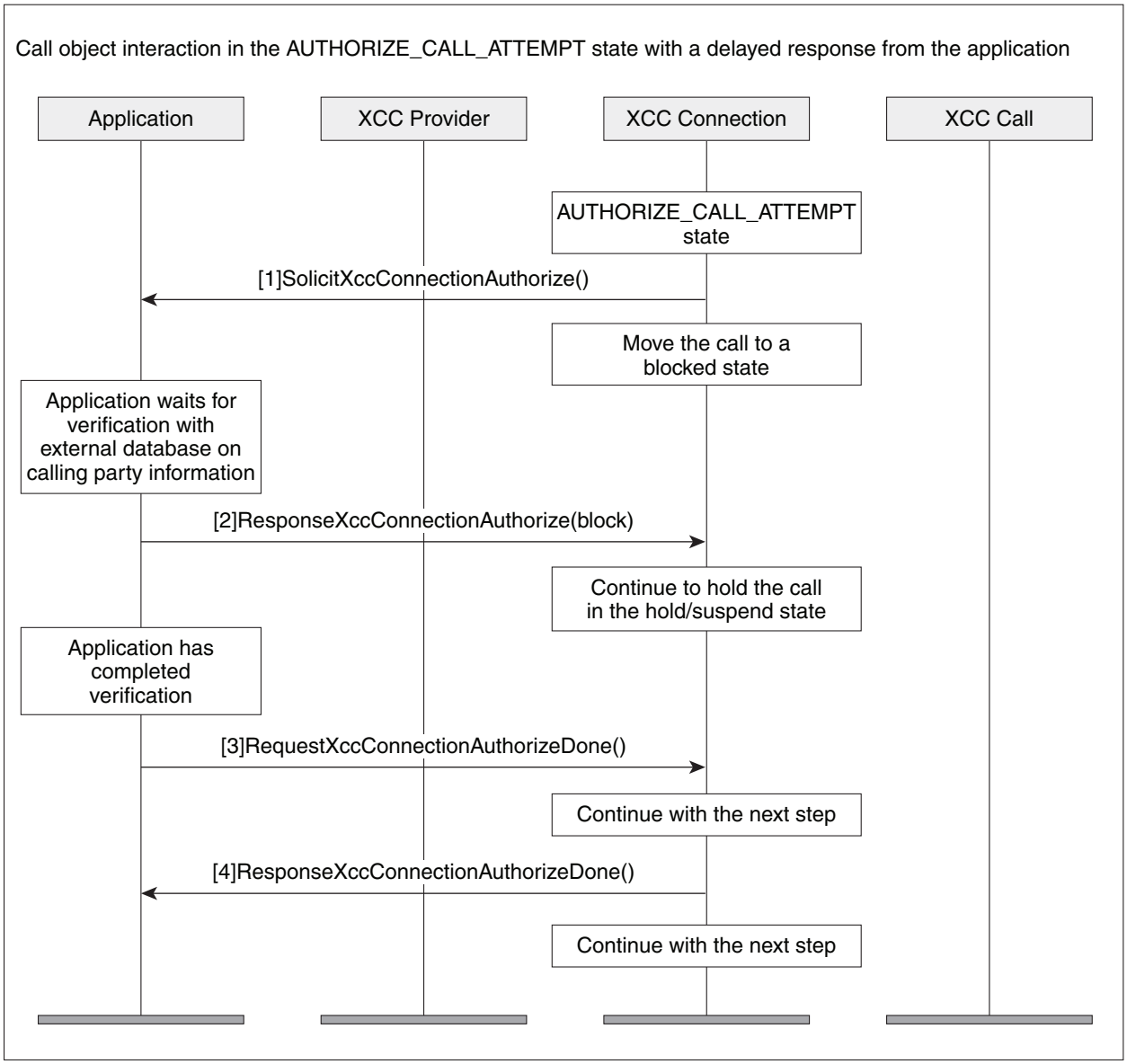
```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Header/>
  <soapenv:Body>
    <ResponseXccConnectionAuthorize xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <action>continueProcessing</action>
      <msgHeader>
        <registrationID>152E034C:XCC:myapp:5</registrationID>
        <transactionID>152E6854:6A</transactionID>
      </msgHeader>
```

```
    </ResponseXccConnectionAuthorize>
  </soapenv:Body>
</soapenv:Envelope>
```

Interaction for Call Authorization with a Delayed Response

Figure A-4 illustrates the call interaction when an application cannot respond immediately to a call authorization solicit message from the XCC provider. The application can request that the XCC provider temporarily block the call.

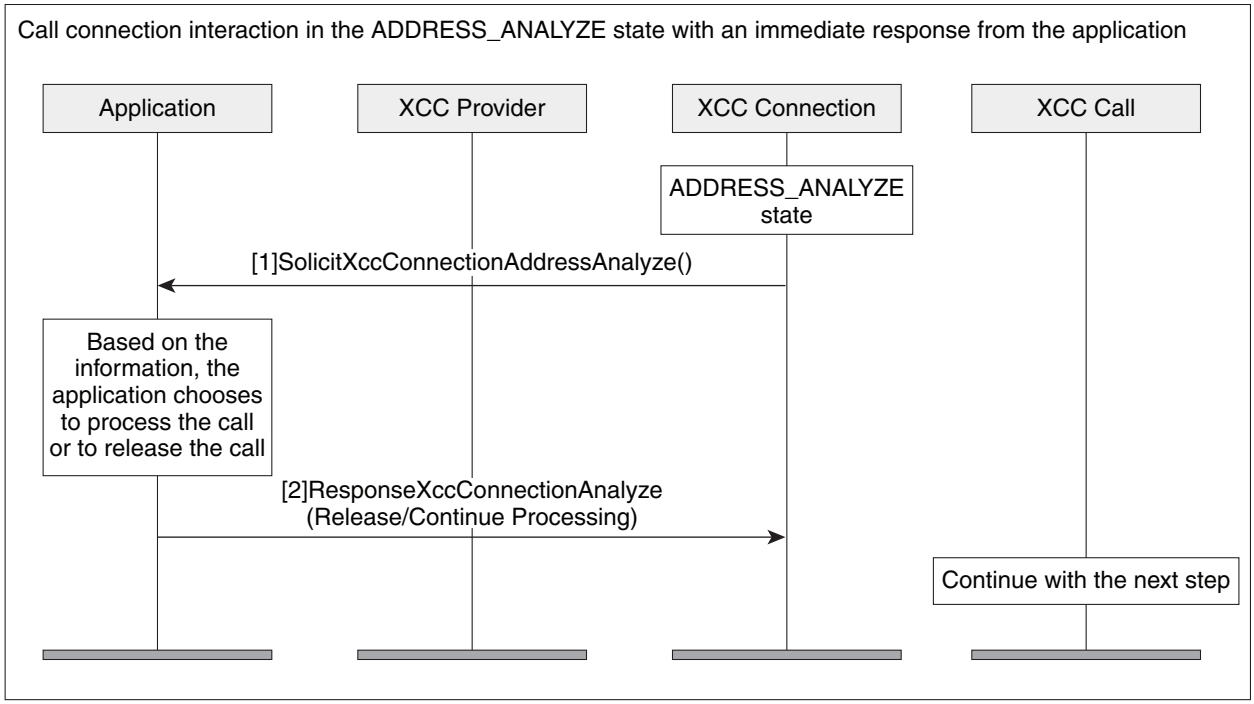
Figure A-4 Call Interaction when the application has a delayed response



Interaction During Digit Collection with an Immediate Response

Figure A-5 shows the call interaction after an application has sent a message to the XCC provider to continue the call and begin collecting digits. The application is able to respond immediately.

Figure A-5 Call Interaction when the application responds immediately upon digit collection



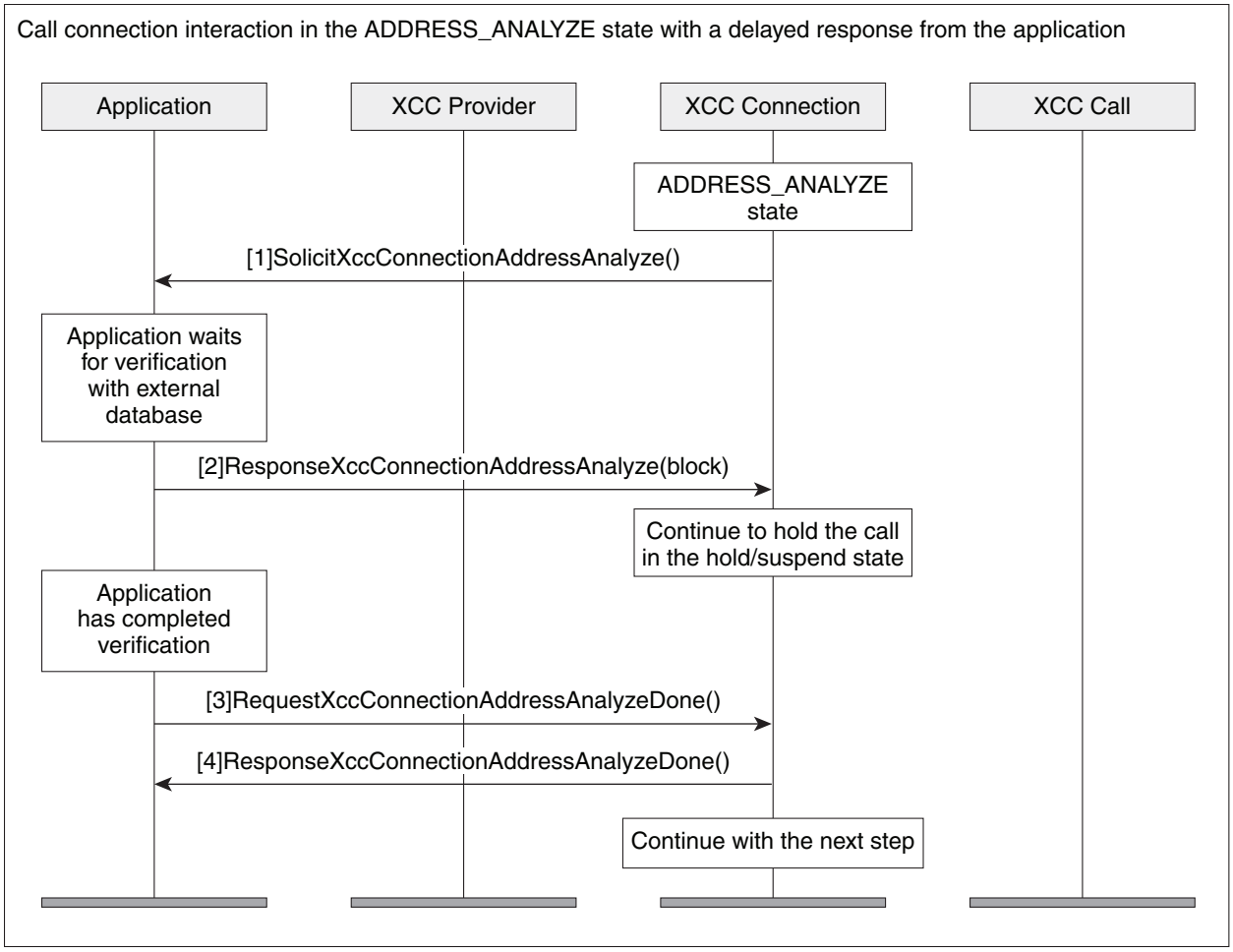
The following example is the SolicitXccConnectionAddressAnalyze message sent from the XCC provider with call information for the application.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <SolicitXccConnectionAddressAnalyze
xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>152E6B18:6B</transactionID>
        <registrationID>152E034C:XCC:myapp:5</registrationID>
      </msgHeader>
      <callData>
        <callID>5</callID>
        <state>ACTIVE</state>
      </callData>
      <connData>
        <connID>30</connID>
        <state>ADDRESS_ANALYZE</state>
      </connData>
      <collectAddress>
        <type>E164</type>
        <addr>2002</addr>
      </collectAddress>
    </SolicitXccConnectionAddressAnalyze>
  </SOAP:Body>
</SOAP:Envelope>
```

Interaction During Digit Collection with a Delayed Response

Figure A-6 shows the call interaction after an application has sent a message to the XCC provider to continue to begin collecting digits, but the application is unable to respond immediately.

Figure A-6 Call Interaction when the application has a delayed response to digit collections



Notification Examples

The following example is the NotifyXccConnection message sent from the XCC provider letting the application know that an outgoing call is being connected.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <NotifyXccConnectionData xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>490D843C:2C9</transactionID>
        <registrationID>490D710C:XCC:myapp:28</registrationID>
      </msgHeader>
      <callData>
        <callID>37</callID>
        <state>ACTIVE</state>
      </callData>
```

```
      <connData>
        <connID>280</connID>
        <state>CONNECTED</state>
      </connData>
      <event>
        <CONNECTED>
          <connDetailData>
            <connData>
              <connID>280</connID>
              <state>CONNECTED</state>
            </connData>
            <guid>B64EC537-872C-11E0-8FBC-A55F7D9A8E13</guid>
            <callingAddrData>
              <type>E164</type>
              <addr>2001</addr>
            </callingAddrData>
            <calledAddrData>
              <type>E164</type>
              <addr>3001</addr>
            </calledAddrData>
            <origCallingAddrData>
              <type>E164</type>
              <addr>2001</addr>
            </origCallingAddrData>
            <origCalledAddrData>
              <type>E164</type>
              <addr>3001</addr>
            </origCalledAddrData>
            <connIntfType>CONN_SIP</connIntfType>
            <mediaData>
              <type>VOICE</type>
              <coderType>g711ulaw</coderType>
              <coderByte>160</coderByte>
            </mediaData>
            <connIntf>9.10.21.254</connIntf>
            <connDirectionType>INCOMING</connDirectionType>
          </connDetailData>
        </CONNECTED>
      </event>
    </NotifyXccConnectionData>
  </SOAP:Body>
</SOAP:Envelope>
```

The following example is the NotifyXccConnection message sent from the XCC provider letting the application know that a transferred event has occurred.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <NotifyXccConnectionData xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>48EE6610:2B2</transactionID>
        <registrationID>48EDDDC8:XCC:myapp:27</registrationID>
      </msgHeader>
      <callData>
        <callID>36</callID>
        <state>ACTIVE</state>
      </callData>
      <connData>
        <connID>274</connID>
        <state>DISCONNECTED</state>
      </connData>
      <event>
        <disconnected>
```

```
            <mediaData>
              <type>VOICE</type>
              <coderType>g711ulaw</coderType>
              <coderByte>160</coderByte>
            </mediaData>
            <statsData>
              <callDuration>PT7.46S</callDuration>
              <TxPacketsCount>365</TxPacketsCount>
              <TxBytesCount>58400</TxBytesCount>
              <TxDurationMSec>0</TxDurationMSec>
              <TxVoiceDurationMSec>0</TxVoiceDurationMSec>
              <RxPacketsCount>359</RxPacketsCount>
              <RxBytesCount>57440</RxBytesCount>
              <RxDurationMSec>0</RxDurationMSec>
              <RxVoiceDurationMSec>0</RxVoiceDurationMSec>
            </statsData>
            <discCause>16</discCause>
            <jitterData>
              <routeTripDelayMSec>0</routeTripDelayMSec>
              <onTimeRvPlayMSec>0</onTimeRvPlayMSec>
              <gapFillWithPredicationMSec>0</gapFillWithPredicationMSec>
              <gapFillWithInterpolationMSec>0</gapFillWithInterpolationMSec>
              <gapFillWithRedundancyMSec>0</gapFillWithRedundancyMSec>
              <lostPacketsCount>0</lostPacketsCount>
              <earlyPacketsCount>0</earlyPacketsCount>
              <latePacketsCount>0</latePacketsCount>
              <receiveDelayMSec>0</receiveDelayMSec>
              <loWaterPlayoutDelayMSec>0</loWaterPlayoutDelayMSec>
              <hiWaterPlayoutDelayMSec>0</hiWaterPlayoutDelayMSec>
            </jitterData>
          </disconnected>
        </event>
      </NotifyXccConnectionData>
    </SOAP:Body>
</SOAP:Envelope>
```

The following example is the NotifyXccConnection message sent from the XCC provider letting the application know that a transfer handoff leave event has occurred.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <NotifyXccConnectionData xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>48EE65AC:2AC</transactionID>
        <registrationID>48EDDDC8:XCC:myapp:27</registrationID>
      </msgHeader>
      <callData>
        <callID>35</callID>
        <state>ACTIVE</state>
      </callData>
      <connData>
        <connID>272</connID>
        <state>CONNECTED</state>
      </connData>
      <event>
        <handoffLeave/>
      </event>
    </NotifyXccConnectionData>
  </SOAP:Body>
</SOAP:Envelope>
```

The following example is the NotifyXccConnection message sent from the XCC provider letting the application know that a transfer handoff join event has occurred.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <NotifyXccConnectionData xmlns="http://www.cisco.com/schema/cisco_xcc/v1_0">
      <msgHeader>
        <transactionID>48EE65AC:2AD</transactionID>
        <registrationID>48EDDDC8:XCC:myapp:27</registrationID>
      </msgHeader>
      <callData>
        <callID>36</callID>
        <state>ACTIVE</state>
      </callData>
      <connData>
        <connID>272</connID>
        <state>CONNECTED</state>
      </connData>
      <event>
        <handoffJoin>
          <connDetailData>
            <connData>
              <connID>272</connID>
              <state>CONNECTED</state>
            </connData>
            <guid>99CFA037-F5F2-11B2-8255-AC403F9877FF</guid>
            <callingAddrData>
              <type>E164</type>
              <addr>2001</addr>
            </callingAddrData>
            <calledAddrData>
              <type>E164</type>
              <addr>3001</addr>
            </calledAddrData>
            <origCallingAddrData>
              <type>E164</type>
              <addr>2001</addr>
            </origCallingAddrData>
            <origCalledAddrData>
              <type>E164</type>
              <addr>3001</addr>
            </origCalledAddrData>
            <connIntfType>CONN_SIP</connIntfType>
            <mediaData>
              <type>VOICE</type>
              <coderType>g711ulaw</coderType>
              <coderByte>160</coderByte>
            </mediaData>
            <connIntf>9.10.31.254</connIntf>
            <routeName>SANJOSE_SIP</routeName>
            <routeDescription>IN</routeDescription>
            <connDirectionType>OUTGOING</connDirectionType>
          </connDetailData>
        </handoffJoin>
      </event>
    </NotifyXccConnectionData>
  </SOAP:Body>
</SOAP:Envelope>
```

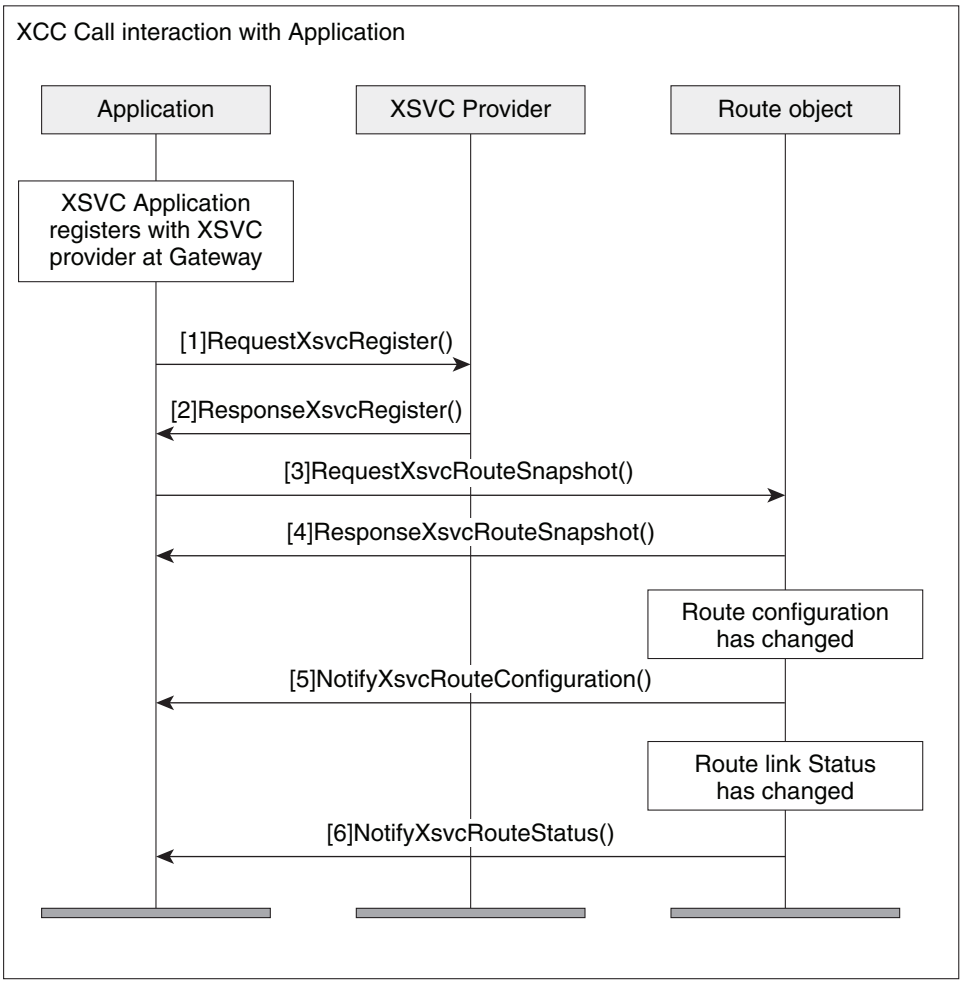
XSVC

This section describes the some of the interactions that take place between the XSVC provider and the application.

Interaction Between the XSVC Provider, Application, and Route Object

Figure A-7 shows the interaction and the sequence of messages that are exchanged between the applicatio, XSVC provider, and the route object during registration.

Figure A-7 Interaction between an applicaton , XSVC provider, and route object



Message Examples

This section provides examples of message exchanges between the application and the XSVC provider.

Example of a Registration Message Exchange

The following is an example of a RequestXsvcRegister message sent from the application requesting registration and setting route event filters.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Body>
    <RequestXsvcRegister xmlns="http://www.cisco.com/schema/cisco_xsvc/v1_0">
      <applicationData>
        <name>myapp</name>
        <url>http://test.com:8090/xsvc</url>
      </applicationData>
      <msgHeader>
        <transactionID>txID001</transactionID>
      </msgHeader>
      <providerData>
        <url>http://10.1.1.1:8090/cisco_xsvc</url>
      </providerData>
      <routeEventsFilter>ROUTE_CONF_UPDATED ROUTE_STATUS_UPDATED</routeEventsFilter>
    </RequestXsvcRegister>
  </soapenv:Body>
</soapenv:Envelope>
```

The following is an example of a ResponseXsvcRegister message sent from the XSVC provider in response to the application's registration request.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <ResponseXsvcRegister xmlns="http://www.cisco.com/schema/cisco_xsvc/v1_0">
      <msgHeader>
        <transactionID>txID001</transactionID>
        <registrationID>2F2EEC:XSVC:myapp:1</registrationID>
      </msgHeader>
      <providerStatus>IN_SERVICE</providerStatus>
    </ResponseXsvcRegister>
  </SOAP:Body>
</SOAP:Envelope>
```

The following is an example of a NotifyXsvcStatusmessage sent from the XSVC provider when it enters the shutdown state.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <NotifyXsvcProviderStatus xmlns="http://www.cisco.com/schema/cisco_xsvc/v1_0">
      <msgHeader>
        <transactionID>6A89EC:B</transactionID>
      </msgHeader>
      <applicationData>
        <url>http://test.com:8090/xsvc</url>
      </applicationData>
      <providerData>
        <url>http://10.1.1.1:8090/cisco_xsvc</url>
      </providerData>
      <providerStatus>SHUTDOWN</providerStatus>
    </NotifyXsvcProviderStatus>
  </SOAP:Body>
</SOAP:Envelope>
```

Example of a Snapshot Reponse Message

The following is an example of a ResponseXsvcRouteSnapshot message sent from XSVC provider with route information.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
```

```
  <SOAP:Body>
    <ResponseXsvcRouteSnapshot xmlns="http://www.cisco.com/schema/cisco_xsvc/v1_0">
      <msgHeader>
        <transactionID>txID002</transactionID>
        <registrationID>77F9EC:XSVC:myapp:5</registrationID>
      </msgHeader>
      <routeList>
        <route>
          <routeName>pri</routeName>
          <routeType>PSTN</routeType>
          <trunkList>
            <trunkData>
              <name>Se0/1/0:23</name>
              <type>ISDN_PRI</type>
              <status>UP</status>
            </trunkData>
          </trunkList>
        </route>
        <route>
          <routeName>1</routeName>
          <routeType>VOIP</routeType>
          <trunkList>
            <trunkData>
              <name>11.1.1.1</name>
              <type>H323</type>
              <status>UP</status>
            </trunkData>
          </trunkList>
        </route>
      </routeList>
    </ResponseXsvcRouteSnapshot>
  </SOAP:Body>
</SOAP:Envelope>
```

Example of a Route Configuration Change

The following is an example of a NotifyXsvcRouteConfiguration message sent from XSVC provider notifying the application that the route list has been modified.

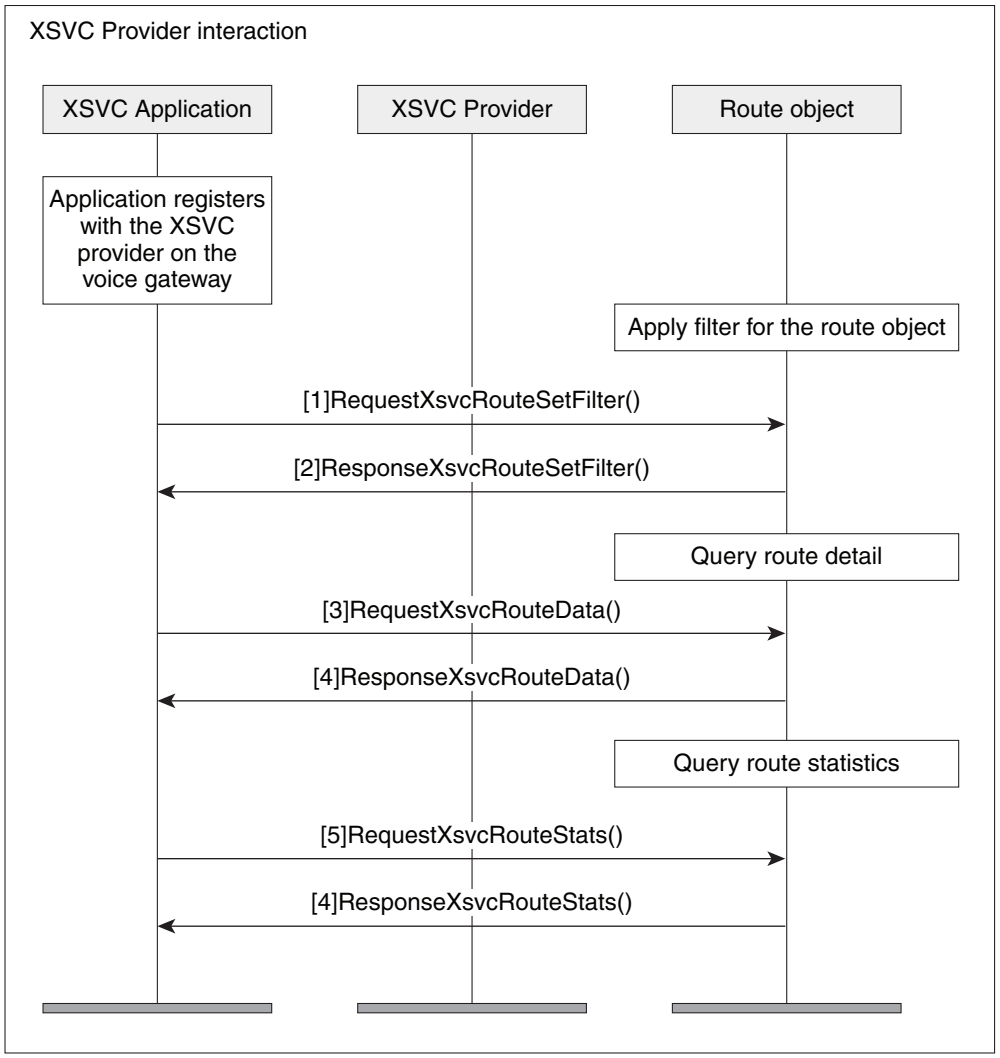
```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <NotifyXsvcRouteConfiguration xmlns="http://www.cisco.com/schema/cisco_xsvc/v1_0">
      <msgHeader>
        <transactionID>7FFC8C:1C</transactionID>
        <registrationID>7E4130:XSVC:myapp:6</registrationID>
      </msgHeader>
      <type>MODIFIED</type>
      <routeList>
        <route>
          <routeName>pri</routeName>
          <routeType>PSTN</routeType>
          <trunkList>
            <trunkData>
              <name>Se0/1/0:23</name>
              <type>ISDN_PRI</type>
              <status>UP</status>
            </trunkData>
          </trunkList>
        </route>
      </routeList>
    </NotifyXsvcRouteConfiguration>
```

```
  </SOAP:Body>
</SOAP:Envelope>
```

Interaction between the Application and the XSVC Provider

Figure A-8 illustrates the call interaction when an application responds immediately to a call authorization solicit message from the XSVC provider.

Figure A-8 Interaction between the application, XSVC provider, and route object when new filters are applied



Example of a Route Data Message

The following is an example of a ResponseXsvcRouteStats message sent from XSVC provider with route statistics.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <ResponseXsvcRouteStats xmlns="http://www.cisco.com/schema/cisco_xsvc/v1_0">
```

```
<msgHeader>
  <transactionID>txID003</transactionID>
  <registrationID>77F9EC:XSVC:myapp:5</registrationID>
</msgHeader>
<routeList>
  <route>
    <routeName>pri</routeName>
    <routeType>PSTN</routeType>
    <trunkList>
      <trunkData>
        <name>Se0/1/0:23</name>
        <type>ISDN_PRI</type>
        <status>UP</status>
        <currentStatics>
          <LCV>0</LCV>
          <PCV>0</PCV>
          <CSS>0</CSS>
          <SEFS>0</SEFS>
          <LES>0</LES>
          <DM>0</DM>
          <ES>0</ES>
          <BES>0</BES>
          <SES>0</SES>
          <UAS>0</UAS>
        </currentStatics>
        <totalStatics>
          <LCV>47</LCV>
          <PCV>6</PCV>
          <CSS>1</CSS>
          <SEFS>1</SEFS>
          <LES>1</LES>
          <DM>0</DM>
          <ES>0</ES>
          <BES>0</BES>
          <SES>0</SES>
          <UAS>2</UAS>
        </totalStatics>
      </trunkData>
    </trunkList>
  </route>
</routeList>
</ResponseXsvcRouteStats>
</SOAP:Body>
</SOAP:Envelope>
```

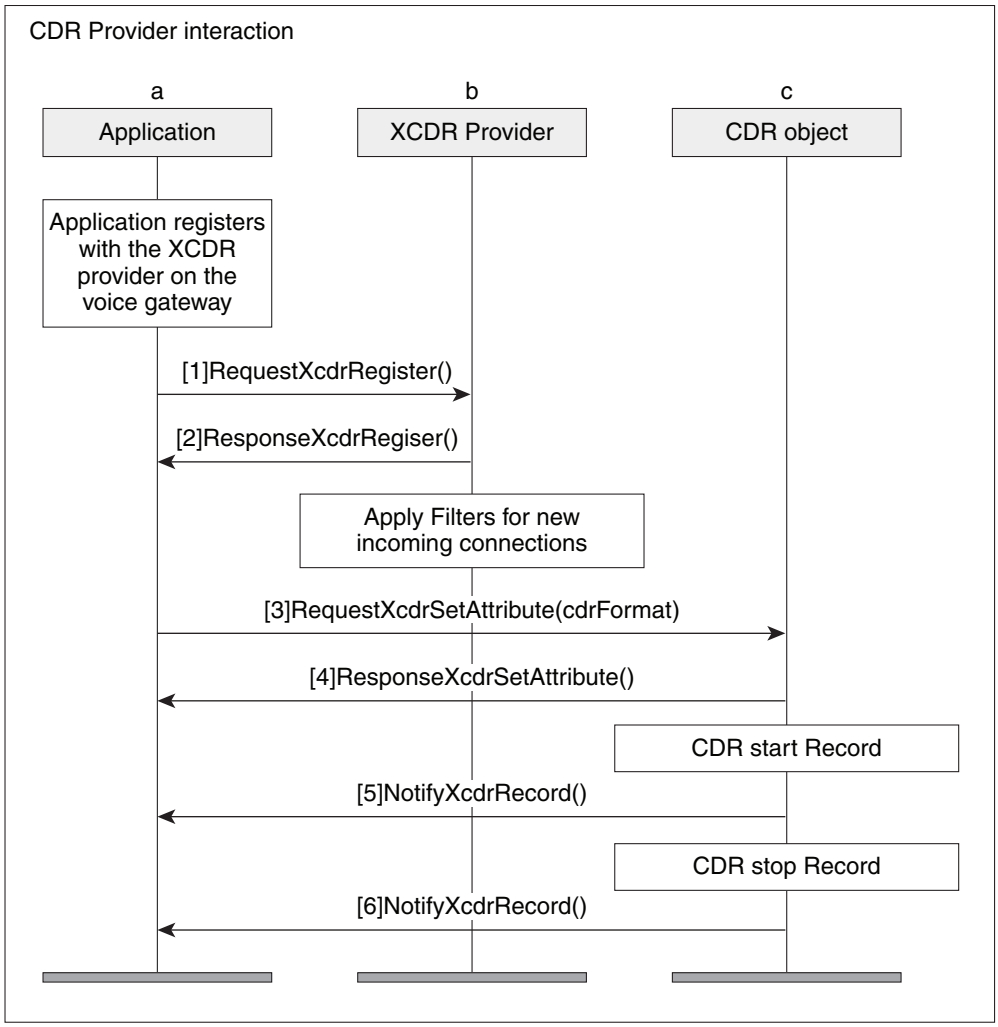
XCDR

This section describes some of the interactions that takes place etween the XCDR provider and the application.

Interaction Between the XCDR Provider and Application

Figure A-9 shows the interaction and the sequence of messages that are exchanged between the application and the XCDR provider during registration.

Figure A-9 Messae interaction when the application registers with the XCDR provider



Message Examples

This section provides examples of message exchanges between the application and the XCDR provider.

Example of a Registration Message Exchange

The following is an example of a RequestXcdrRegister message sent from the application requesting registration and specifying the type of records that it expects to receive.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Body>
    <RequestXcdrRegister xmlns="http://www.cisco.com/schema/cisco_xcdr/v1_0">
      <applicationData>
        <name>myapp</name>
        <url>http://test.com:8090/xcdr</url>
      </applicationData>
      <msgHeader>
        <transactionID>txID001</transactionID>
      </msgHeader>
```

```
      <providerData>
        <url>http://10.1.1.1:8090/cisco_xcdr</url>
      </providerData>
    </RequestXcdrRegister>
  </soapenv:Body>
</soapenv:Envelope>
```

The following is an example of a ResponseXcdrRegister message sent from the XCDR provider in response to the application's registration request.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
  <SOAP:Body>
    <ResponseXcdrRegister xmlns="http://www.cisco.com/schema/cisco_xcdr/v1_0">
      <msgHeader>
        <transactionID>txID001</transactionID>
        <registrationID>152E0204:XCDR:myapp:5</registrationID>
      </msgHeader>
      <providerStatus>IN_SERVICE</providerStatus>
    </ResponseXcdrRegister>
  </SOAP:Body>
</SOAP:Envelope>
```

INDEX