Cisco ICM Software CTI OS
Developer’s Guide
CTI Software Release 6.0(0)
May 2004

Corporate 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: 408 526-4100

Customer Order Number:
Text Part Number:
About This Guide  xxii

Purpose  xxii
Audience  xxii
Conventions  xxii
Organization  xxii
Other Publications  xxiii

Obtaining Documentation  xxiv
Cisco.com  xxiv
Documentation CD-ROM  xxiv
Ordering Documentation  xxv
Documentation Feedback  xxv

Obtaining Technical Assistance  xxv
Cisco.com  xxvi
Technical Assistance Center  xxvi

Obtaining Additional Publications and Information  xxviii

CHAPTER 1

Introduction  1-1

Introduction to CTI  1-1
What is a CTI-Enabled Application?  1-2
  Screen Pop  1-2
  Agent State Control  1-2
  Third-Party Call Control  1-3
Leveraging CTI Application Event Flow  1-3
  Asynchronous Events  1-3
Contents

Request-Response Paradigm 1-4
Overview of CTI OS 1-5
Advantages of CTI OS as Interface to ICM 1-6
Key Benefits of CTI OS for CTI Application Developers 1-7

Chapter 2
CTI OS Client Interface Library Architecture 2-1
Object Server Architecture 2-1
Client Interface Library Architecture 2-2
  Connection Layer 2-3
  Service Layer 2-3
  Object Interface Layer 2-4
  Custom Application 2-4
CIL Object Model 2-4
  Session Object 2-5
  Agent Object 2-6
  Call Object 2-6
  SkillGroup Object 2-7
  Object Creation and Lifetime 2-7
  Reference Counting 2-7
Where To Go From Here 2-9

Chapter 3
CIL Coding Conventions 3-1
CTIOS CIL Data Types 3-2
Asynchronous Program Execution 3-3
CIL Error Codes 3-4
COM Error Codes 3-9
Generic Interfaces 3-10
  Arguments 3-11
  Accessing Properties and Parameters with GetValue 3-12
Contents

Setting Request Parameters with AddItem 3-13
Setting Object Properties with SetValue 3-13
UniqueObjectID 3-14
Obtaining Objects from UniqueObjectIDs 3-16
Using Button Enablement Masks 3-17

CHAPTER 4
Building Your Application 4-1
Integrating your Application with CTI OS via the CIL 4-2
Planning and Designing Your Integration 4-2
What Language and Interface to Use 4-3
Testing CTI Applications 4-4
Developing a Test Plan 4-4
Test Environment 4-5
Using the Samples 4-5
Using The CTI OS ActiveX Controls 4-7
Building a Simple Softphone with ActiveX Controls 4-7
Adding a Hook for Screenpops 4-10
Using the COM CIL in Visual Basic 4-13
Referencing COM Components in Visual Basic 4-14
Registering for Events in Visual Basic 4-15
Next Steps 4-15
Using the COM CIL in C++ 4-16
Adding COM Support to your Application 4-16
Using the CIL Dynamic Link Libraries 4-17
Creating an Instance of a COM Object 4-18
Subscribing and Unsubscribing to COM Events in C++ 4-18
Next Steps 4-19
Using the C++ CIL and static libraries 4-19
Header Files and Libraries 4-19

Cisco ICM Software CTI OS Developer's Guide Release 6.0(0)
Contents

How to Set Up an Agent Application to Receive Agent Statistics  4-52
How to Set Up a Monitor-Mode Application to Receive Agent Statistics  4-53
Accessing Agent Statistics  4-57
Changing Which Agent Statistics are Sent  4-58
Agent Statistics Computed by the Sample CTI OS Desktop  4-58

Working with Skill Group Statistics  4-59
Overview  4-59
How to Set Up a Monitor-Mode Application to Receive Skill Group Statistics  4-60
Accessing Skill Group Statistics  4-62
Changing Which Skill Group Statistics are Sent  4-63
Skill Group Statistics Computed by the Sample CTI OS Desktop  4-63

Enabling Silent Monitor in your Application  4-65
Creating a Silent Monitor Object  4-65
Setting the Session Mode  4-66
Initiating and Ending a Silent Monitor Session  4-67
Shutting Down Silent Monitor Manager  4-69

Deployment of Custom CTI OS Applications  4-69
Applications Using the ActiveX Controls  4-69
Applications Using COM (But Not ActiveX Controls)  4-73
Deployment of C++ Applications  4-73

CHAPTER 5

CTI OS ActiveX Softphone Controls  5-1
Property Pages  5-3
Button Controls and Grid Controls  5-4
  Button Controls  5-4
  Grid Controls  5-4
CTI OS ActiveX Control Descriptions  5-5
  AgentStateCtl  5-5
  AgentSelectCtl  5-9
AgentStatisticsCtl 5-10
AlternateCtl 5-10
AnswerCtl 5-10
BadLineCtl 5-11
CallAppearanceCtl 5-11
ChatCtl 5-13
ConferenceCtl 5-13
EmergencyAssistCtl 5-15
HoldCtl 5-16
MakeCallCtl 5-17
ReconnectCtl 5-18
SkillgroupStatisticsCtl 5-19
StatusBarCtl 5-19
SupervisorOnlyCtl 5-20
RecordCtl 5-22
TransferCtl 5-22
The Silent Monitor StandAlone Com Object 5-24
Connect 5-25
Disconnect 5-25
StartMonitoring 5-25
StopMonitoring 5-26
SilentMonitor Com Object Events 5-26
Deployment 5-28
Sample Usage in Visual Basic 5-28
AllagentsMonitor Sample 5-29

CHAPTER 6

Event Interfaces and Events 6-1
Event Publication Model 6-1
ISessionEvents Interface 6-2
OnCallTranslationRouteEvent 6-3
OnConnection 6-5
OnConnectionClosed 6-5
OnConnectionFailure 6-6
OnConnectionRejected 6-6
OnCTIOSFailure 6-7
OnCurrentAgentReset 6-9
OnCurrentCallChanged 6-9
OnFailureEvent 6-10
OnGlobalSettingsDownloadConf 6-10
OnHeartbeat 6-15
OnMissingHeartbeat 6-16
OnMonitorMode Established 6-16
OnSnapshotDeviceConf 6-17
OnSnapshotSkillGroupList 6-18

ICallEvents Interface 6-19
OnAlternateCallConf 6-19
OnAnswerCallConf 6-20
OnCallBegin 6-20
OnCallEnd 6-23
OnCallDataUpdate 6-24
OnCallDelivered 6-28
OnCallEstablished 6-30
OnCallHeld 6-32
OnCallRetrieved 6-33
OnCallCleared 6-33
OnCallConnectionCleared 6-34
OnCallOriginated 6-35
OnCallFailed 6-37
OnCallTransferred 6-38
OnCallConferenced 6-41
OnCallDiverted  6-43
OnAgentPrecallEvent  6-45
OnAgentPreCallAbortEvent  6-45
TOnCallServiceInitiatedEvent  6-45
OnCallQueuedEvent  6-47
OnCallDequeuedEvent  6-49
OnCallReachedNetworkEvent  6-50
OnCallStartRecordingConf  6-52
OnCallStopRecordingConf  6-52
OnClearCallConf  6-53
OnClearConnectionConf  6-53
OnConferenceCallConf  6-54
OnControlFailureConf  6-55
OnConsultationCallConf  6-56
OnHoldCallConf  6-57
OnMakePredictiveCallConf  6-57
OnReconnectCallConf  6-57
OnReleaseCallConf  6-58
OnRetrieveCallConf  6-58
OnSendDTMFCbf  6-59
OnSnapshotCallConf  6-59
OnTransferCallConf  6-62
IAgentEvents Interface  6-63
OnAgentStateChange  6-63
OnAgentStatistics  6-66
OnChatMessage  6-76
OnControlFailureConf  6-77
OnMakeCallConf  6-85
OnPostLogout  6-85
OnPreLogout  6-87
OnSetAgentModeEvent 6-89
OnSetAgentStateConf 6-90
OnLogoutFailed 6-91
OnNewAgentTeamMember 6-91
OnEmergencyCall 6-93
OnStartMonitoringAgent 6-94
OnStopMonitoringAgent 6-95
OnQueryAgentStateConf 6-96
OnAgentInfoEvent 6-99
OnSilentMonitorStartRequest 6-100
OnSilentMonitorStopRequest 6-101
OnUserMessageConf 6-102
SkillGroupEvents Interface 6-102
  OnSkillGroupStatisticsUpdated 6-102
IButtonEnablementEvents 6-103
  OnButtonEnablementChange 6-103
  OnSupervisorButtonChange 6-106
ISilentMonitorEvents 6-107
  OnSilentMonitorStartRequestedEvent 6-107
  OnSilentMonitorStartedEvent 6-109
  OnSilentMonitorStopRequestedEvent 6-111
  OnSilentMonitorStatusReportEvent 6-112
  OnStartSilentMonitorConf 6-114
  OnStopSilentMonitorConf 6-116
  OnSilentMonitorSessionDisconnected 6-117
  OnRTPStreamTimedoutEvent 6-118
IMonitoredAgentEvents Interface 6-119
IMonitoredCallEvents Interface 6-119
Events in Java CIL 6-120
CHAPTER 7  CtiOs Object  7-1

Methods  7-2
  DumpProperties  7-2
  GetAllProperties  7-3
  GetElement  7-4
  GetLastError (Java only)  7-5
  GetNumProperties  7-5
  GetPropertyName  7-6
  GetPropertyType  7-7
  GetValue  7-8
  GetValueArray  7-9
  GetValueBoolObj (Java only)  7-10
  GetValueInt  7-10
  GetValueIntObj (Java only)  7-11
  GetValueShortObj (Java only)  7-12
  GetValueString  7-12
  GetValueUIntObj (Java only)  7-13
  GetValueUShortObj (Java only)  7-14
  IsValid  7-14
  ReportError (Java only)  7-15
  SetValue (Java only)  7-16

CHAPTER 8  Session Object  8-1

Session Object Properties  8-2

Methods  8-4
  AddEventListener (Java only)  8-6
  AddListener Methods (C++ only)  8-7
  Connect  8-8
  CreateSilentMonitorManager  8-10
  CreateWaitObject (C++ and Java only)  8-11
DestroySilentMonitorManager 8-12
DestroyWaitObject (C++ only) 8-13
DisableSkillGroupStatistics (C++ and Java only) 8-14
Disconnect 8-15
DumpProperties 8-15
EnableSkillGroupStatistics (C++ and Java only) 8-16
GetAllAgents 8-17
GetAllCalls 8-17
GetAllProperties 8-18
GetAllSkillGroups 8-18
GetCurrentAgent 8-19
GetCurrentCall 8-20
GetCurrentSilentMonitor 8-21
GetElement 8-21
GetNumProperties 8-21
GetObjectFromObjectID 8-21
GetPropertyName 8-22
GetPropertyType 8-23
GetSystemStatus (Java and C++ only) 8-23
GetValue Methods 8-23
IsAgent 8-23
IsSupervisor 8-24
IsValid 8-25
LogToServer 8-25
RemoveEventListener (Java only) 8-26
RemoveListener Methods (C++ only) 8-26
RequestDesktopSettings 8-27
SetAgent 8-28
SetCurrentCall 8-30
SetCurrentSilentMonitor 8-30
SetMessageFilter  8-31

Notes On Message Filters  8-33
  Message Filter Syntax  8-33
  A Simple Example  8-34
  General Form of Filter Syntax  8-34
  Combining Filters  8-35
  Filtering for Specific Events  8-35
  Filtering Skillgroup Statistics  8-37

Agent Object  9-1
  Agent Object Properties  9-1
  Agent Statistics  9-3
  Methods  9-12
    Arguments Parameters  9-14
    DisableAgentStatistics  9-15
    DisableSkillGroupStatistics  9-16
    EnableAgentStatistics  9-16
    EnableSkillGroupStatistics  9-17
    GetAgentState  9-18
    GetAllProperties  9-19
    GetElement  9-19
    GetMonitoredAgent  9-19
    GetMonitoredCall  9-20
    GetNumProperties  9-21
    GetPropertyName  9-21
    GetPropertyType  9-21
    GetSkillGroups  9-21
    GetValue Methods  9-23
    IsAgent  9-24
Contents

IsSupervisor  9-24
Login  9-25
Logout  9-27
MakeCall  9-29
MakeEmergencyCall  9-34
QueryAgentState  9-36
ReportBadCallLine  9-37
RequestAgentTeamList  9-38
RequestSupervisorAssist  9-39
SendChatMessage  9-40
SetAgentState  9-41
SetValue  9-43
StartMonitoringAgent  9-44
StartMonitoringAgentTeam  9-45
StartMonitoringAllAgentTeams  9-46
StartMonitoringCall  9-47
StopMonitoringAgent  9-48
StopMonitoringAgentTeam  9-49
StopMonitoringAllAgentTeams  9-50
SuperviseCall  9-51

CHAPTER 10  Call Object  10-1
Current Call Concept  10-1
Accessing ECC Variables  10-2
Retrieving ECC Variable Values  10-2
Adding ECC Values  10-4
Properties  10-5
Methods  10-8
    Arguments Parameters  10-10
Contents

Alternate 10-10
Answer 10-11
Clear 10-12
ClearConnection 10-14
Conference 10-15
GetCallContext 10-17
GetCallData 10-18
GetLastError (Java only) 10-20
GetValue Methods 10-20
GetValueBool Obj (Java only) 10-20
GetValueInt Obj (Java only) 10-21
Parameters 10-21
GetValueShort Obj (Java only) 10-21
Parameters 10-22
GetValueUInt Obj (Java only) 10-22
Parameters 10-22
GetValueUShort Obj (Java only) 10-22
Parameters 10-23
Hold 10-23
MakeConsultCall 10-24
Reconnect 10-30
Retrieve 10-31
SendDTMF Signal 10-32
SetCallData 10-35
SingleStepConference 10-36
SingleStepTransfer 10-39
Snapshot 10-40
StartRecord 10-41
StopRecord 10-42
Transfer 10-43
CHAPTER 11  
SkillGroup Object 11-1  
Properties 11-1  
Statistics 11-2  
Methods 11-22  
DisableSkillGroupStatistics 11-23  
DumpProperties 11-24  
EnableSkillGroupStatistics 11-24  
GetElement 11-25  
GetValue Methods 11-25  
IsValid 11-25  
SetValue 11-25  
CHAPTER 12  
Helper Classes 12-1  
Arg Class 12-2  
AddRef 12-3  
Clone 12-4  
CreateInstance 12-4  
Dump Arg 12-5  
GetType 12-6  
GetValue Methods 12-7  
Release 12-9  
SetValue 12-9  
Arguments Class 12-11  
Usage Notes 12-13  
AddItem 12-14  
AddRef 12-15  
Clear 12-16  
Clone 12-17  
CreateInstance 12-17
## Contents

<table>
<thead>
<tr>
<th>Class/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DumpArgs</td>
<td>12-18</td>
</tr>
<tr>
<td>GetElement Methods</td>
<td>12-19</td>
</tr>
<tr>
<td>GetValue Methods</td>
<td>12-21</td>
</tr>
<tr>
<td>IsValid</td>
<td>12-23</td>
</tr>
<tr>
<td>NumElements</td>
<td>12-24</td>
</tr>
<tr>
<td>Release</td>
<td>12-25</td>
</tr>
<tr>
<td>RemoveItem</td>
<td>12-26</td>
</tr>
<tr>
<td>SetElement</td>
<td>12-26</td>
</tr>
<tr>
<td>SetValue</td>
<td>12-28</td>
</tr>
<tr>
<td>CILRefArg Class (C++ and Java only)</td>
<td>12-30</td>
</tr>
<tr>
<td>GetType</td>
<td>12-30</td>
</tr>
<tr>
<td>GetUniqueObjectID (Java only)</td>
<td>12-30</td>
</tr>
<tr>
<td>GetValue</td>
<td>12-31</td>
</tr>
<tr>
<td>SetValue</td>
<td>12-31</td>
</tr>
<tr>
<td>CCtiosException Class (C++ and Java only)</td>
<td>12-32</td>
</tr>
<tr>
<td>CCtiosException Constructor</td>
<td>12-33</td>
</tr>
<tr>
<td>GetCode</td>
<td>12-33</td>
</tr>
<tr>
<td>GetStatus</td>
<td>12-34</td>
</tr>
<tr>
<td>GetString</td>
<td>12-34</td>
</tr>
<tr>
<td>What</td>
<td>12-35</td>
</tr>
<tr>
<td>CWaitObject Class (C++ and Java only)</td>
<td>12-35</td>
</tr>
<tr>
<td>Methods</td>
<td>12-36</td>
</tr>
<tr>
<td>CreateWaitObject</td>
<td>12-36</td>
</tr>
<tr>
<td>DestroyWaitObject</td>
<td>12-37</td>
</tr>
<tr>
<td>DumpEventMask</td>
<td>12-37</td>
</tr>
<tr>
<td>GetMask</td>
<td>12-37</td>
</tr>
<tr>
<td>GetTriggerEvent</td>
<td>12-38</td>
</tr>
<tr>
<td>InMask</td>
<td>12-38</td>
</tr>
<tr>
<td>SetMask</td>
<td>12-39</td>
</tr>
<tr>
<td>WaitOnMultipleEvents</td>
<td>12-39</td>
</tr>
</tbody>
</table>
CHAPTER 13

SilentMonitorManager Object 13-1

Properties 13-2

Methods 13-3

Argument Parameter Rules 13-4
AcceptSilentMonitoring 13-4
GetIPPhoneInfo 13-6
GetSessionInfo 13-8
GetSMSessionList 13-10
IsMonitoredTarget 13-11
SetIPPhoneInfo 13-11
StartSilentMonitorRequest 13-13
StartSMMonitoredMode 13-15
StartSMMonitoringMode 13-16
StopSilentMonitorMode 13-16
StopSilentMonitorRequest 13-17

APPENDIX A

CTI OS Keywords and Enumerated Types A-1

Keywords A-1
Java CIL Keywords A-2
Enumerated Types A-2

APPENDIX B

CTI OS Logging E-1

Taking CTI OS Client Logs E-1
How to Set Trace Levels E-2
Java CIL Logging and Tracing E-2

INDEX
About This Guide

Purpose

This manual provides a brief overview of the Cisco CTI Object Server (CTI OS) product, introduces programmers to developing CTI enabled applications with CTI OS, and describes the syntax and usage for CTI OS methods and events.

Audience

This manual is for system integrators and programmers who want to use CTI OS to integrate CTI applications with Cisco ICM software.

Conventions

This manual uses the following conventions.
Signoff Draft 4/14/04

Format | Example
---|---
Boldface type is used for user entries, keys, buttons, and folder and submenu names. | Choose **Edit > Find** from the ICM Configure menu bar.
Italic type indicates one of the following:
- A newly introduced term
- For emphasis
- A generic syntax item that you must replace with a specific value
- A title of a publication | • **A skill group** is a collection of agents who share similar skills.
• **Do not** use the numerical naming convention that is used in the predefined templates (for example, **persvc01**).
• **IF** *(condition, true-value, false-value)*
• For more information, see the *Cisco ICM Software Database Schema Handbook*.
An arrow (**>**) indicates an item from a pull-down menu. | The Save command from the File menu is referenced as **File > Save**.

Organization

The manual is divided into the following chapters.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1, “Introduction”</td>
<td>Provides an overview of CTI and the CTI OS Client Interface.</td>
</tr>
<tr>
<td>Chapter 2, “CTI OS Client Interface Library Architecture”</td>
<td>Discusses CTI OS architecture.</td>
</tr>
<tr>
<td>Chapter 3, “CIL Coding Conventions”</td>
<td>Explains how to build an application using the CTI OS libraries.</td>
</tr>
<tr>
<td>Chapter 4, “Building Your Application”</td>
<td>Discusses how to build your custom CTI application to use the CTI OS Client Interface Library</td>
</tr>
</tbody>
</table>
**Signoff Draft 4/14/04**

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 5, “CTI OS ActiveX Softphone Controls”</td>
<td>Describes the CTI OS softphone controls and explains how to use them in a VB or COM container.</td>
</tr>
<tr>
<td>Chapter 6, “Event Interfaces and Events”</td>
<td>Describes the CTI OS event interfaces.</td>
</tr>
<tr>
<td>Chapter 7, “CtiOs Object”</td>
<td>Discusses features common to all CTI OS objects derived from CtiOsObject.</td>
</tr>
<tr>
<td>Chapter 8, “Session Object”</td>
<td>Describes the methods associated with the CTI OS Session object.</td>
</tr>
<tr>
<td>Chapter 9, “Agent Object”</td>
<td>Describes the methods associated with the CTI OS Agent object.</td>
</tr>
<tr>
<td>Chapter 10, “Call Object”</td>
<td>Describes the methods associated with the CTI OS Call object.</td>
</tr>
<tr>
<td>Chapter 11, “SkillGroup Object”</td>
<td>Describes the methods associated with the CTI OS SkillGroup object.</td>
</tr>
<tr>
<td>Chapter 12, “Helper Classes”</td>
<td>Describes the methods associated with the CTI OS Arguments classes.</td>
</tr>
<tr>
<td>Chapter 13, “SilentMonitorManager Object”</td>
<td>Describes the methods associated with the CTI OS Silent Monitor Manager object.</td>
</tr>
<tr>
<td>Appendix A, “CTI OS Keywords and Enumerated Types”</td>
<td>Discusses CTI OS keywords and enumerated types.</td>
</tr>
<tr>
<td>Appendix B, “CTI OS Logging”</td>
<td>Discusses a few issues related to CTI OS logging.</td>
</tr>
</tbody>
</table>

**Other Publications**

For additional information about Cisco Intelligent Contact Management (ICM) software and Cisco Computer Telephony Integration (CTI) products, see the Cisco web site listing ICM and CTI documentation.
Obtaining Documentation

Cisco provides several ways to obtain documentation, technical assistance, and other technical resources. These sections explain how to obtain technical information from Cisco Systems.

Cisco.com

You can access the most current Cisco documentation on the World Wide Web at this URL:
http://www.cisco.com/univercd/home/home.htm
You can access the Cisco website at this URL:
http://www.cisco.com
International Cisco websites can be accessed from this URL:

Documentation CD-ROM

Cisco documentation and additional literature are available in a Cisco Documentation CD-ROM package, which may have shipped with your product. The Documentation CD-ROM is updated regularly and may be more current than printed documentation. The CD-ROM package is available as a single unit or through an annual or quarterly subscription.

Registered Cisco.com users can order a single Documentation CD-ROM (product number DOC-CONDOCCD=) through the Cisco Ordering tool:
All users can order monthly or quarterly subscriptions through the online Subscription Store:
http://www.cisco.com/go/subscription
About This Guide

Signoff Draft 4/14/04

Ordering Documentation

You can find instructions for ordering documentation at this URL:

You can order Cisco documentation in these ways:

• Registered Cisco.com users (Cisco direct customers) can order Cisco product
documentation from the Networking Products MarketPlace:

• Nonregistered Cisco.com users can order documentation through a local
account representative by calling Cisco Systems Corporate Headquarters
(California, U.S.A.) at 408 526-7208 or, elsewhere in North America, by
calling 800 553-NETS (6387).

Documentation Feedback

You can submit comments electronically on Cisco.com. On the Cisco
Documentation home page, click Feedback at the top of the page.

You can e-mail your comments to bug-doc@cisco.com.

You can submit comments by using the response card (if present) behind the front
cover of your document or by writing to the following address:

Cisco Systems
Attn: Customer Document Ordering
170 West Tasman Drive
San Jose, CA 95134-9883

We appreciate your comments.

Obtaining Technical Assistance

Cisco provides Cisco.com, which includes the Cisco Technical Assistance Center
(TAC) website, as a starting point for all technical assistance. Customers and
partners can obtain online documentation, troubleshooting tips, and sample
configurations from the Cisco TAC website. Cisco.com registered users have complete access to the technical support resources on the Cisco TAC website, including TAC tools and utilities.

Cisco.com

Cisco.com offers a suite of interactive, networked services that let you access Cisco information, networking solutions, services, programs, and resources at any time, from anywhere in the world. Cisco.com provides a broad range of features and services to help you with these tasks:

- Streamline business processes and improve productivity
- Resolve technical issues with online support
- Download and test software packages
- Order Cisco learning materials and merchandise
- Register for online skill assessment, training, and certification programs

To obtain customized information and service, you can self-register on Cisco.com at this URL:


Technical Assistance Center

The Cisco TAC is available to all customers who need technical assistance with a Cisco product, technology, or solution. Two types of support are available: the Cisco TAC website and the Cisco TAC Escalation Center. The type of support that you choose depends on the priority of the problem and the conditions stated in service contracts, when applicable.

We categorize Cisco TAC inquiries according to urgency:

- Priority level 4 (P4)—You need information or assistance concerning Cisco product capabilities, product installation, or basic product configuration. There is little or no impact to your business operations.
About This Guide

Obtaining Technical Assistance

Signoff Draft 4/14/04

- Priority level 3 (P3)—Operational performance of the network is impaired, but most business operations remain functional. You and Cisco are willing to commit resources during normal business hours to restore service to satisfactory levels.

- Priority level 2 (P2)—Operation of an existing network is severely degraded, or significant aspects of your business operations are negatively impacted by inadequate performance of Cisco products. You and Cisco will commit full-time resources during normal business hours to resolve the situation.

- Priority level 1 (P1)—An existing network is “down,” or there is a critical impact to your business operations. You and Cisco will commit all necessary resources around the clock to resolve the situation.

Cisco TAC Website

The Cisco TAC website provides online documents and tools to help troubleshoot and resolve technical issues with Cisco products and technologies. To access the Cisco TAC website, go to this URL:

http://www.cisco.com/tac

All customers, partners, and resellers who have a valid Cisco service contract have complete access to the technical support resources on the Cisco TAC website. Some services on the Cisco TAC website require a Cisco.com login ID and password. If you have a valid service contract but do not have a login ID or password, go to this URL to register:


If you are a Cisco.com registered user, and you cannot resolve your technical issues by using the Cisco TAC website, you can open a case online at this URL:

http://www.cisco.com/tac/caseopen

If you have Internet access, we recommend that you open P3 and P4 cases online so that you can fully describe the situation and attach any necessary files.
Cisco TAC Escalation Center

The Cisco TAC Escalation Center addresses priority level 1 or priority level 2 issues. These classifications are assigned when severe network degradation significantly impacts business operations. When you contact the TAC Escalation Center with a P1 or P2 problem, a Cisco TAC engineer automatically opens a case.

To obtain a directory of toll-free Cisco TAC telephone numbers for your country, go to this URL:

Before calling, please check with your network operations center to determine the Cisco support services to which your company is entitled: for example, SMARTnet, SMARTnet Onsite, or Network Supported Accounts (NSA). When you call the center, please have available your service agreement number and your product serial number.

Obtaining Additional Publications and Information

Information about Cisco products, technologies, and network solutions is available from various online and printed sources.

- The Cisco Product Catalog describes the networking products offered by Cisco Systems, as well as ordering and customer support services. Access the Cisco Product Catalog at this URL:

- Cisco Press publishes a wide range of networking publications. Cisco suggests these titles for new and experienced users: Internetworking Terms and Acronyms Dictionary, Internetworking Technology Handbook, Internetworking Troubleshooting Guide, and the Internetworking Design Guide. For current Cisco Press titles and other information, go to Cisco Press online at this URL:
  http://www.ciscopress.com

- Packet magazine is the Cisco quarterly publication that provides the latest networking trends, technology breakthroughs, and Cisco products and solutions to help industry professionals get the most from their networking investment. Included are networking deployment and troubleshooting tips,
configuration examples, customer case studies, tutorials and training, certification information, and links to numerous in-depth online resources. You can access Packet magazine at this URL:
http://www.cisco.com/go/packet

• iQ Magazine is the Cisco bimonthly publication that delivers the latest information about Internet business strategies for executives. You can access iQ Magazine at this URL:
http://www.cisco.com/go/iqmagazine

• Internet Protocol Journal is a quarterly journal published by Cisco Systems for engineering professionals involved in designing, developing, and operating public and private internets and intranets. You can access the Internet Protocol Journal at this URL:

• Training—Cisco offers world-class networking training. Current offerings in network training are listed at this URL:
Introduction

This chapter provides an introduction to Computer Telephony Integration (CTI) and describes how CTI can enhance the value of contact center applications. This chapter also introduces the CTI OS product and discusses the advantages of using CTI OS to develop custom CTI enabled applications.

Introduction to CTI

The workflow of a modern contact center is based on two main areas: the media for communicating with the customer and the platform for servicing customer requests.

CTI is the integration of the communications media (i.e. phone, email, or web) with the customer service platform (i.e. customer databases, transaction processing systems, or CRM (customer relationship management) software packages).

Integrating communications media with the customer service platform helps agents to service customers better and faster in two ways. First, it enables the agent to leverage the information and events provided by the media to direct his workflow. Second, it increases the depth and breadth of customer information presented to the agent when the customer’s contact arrives at the workstation.
What is a CTI-Enabled Application?

A CTI-enabled application is one in which the software used by the agent to service a customer request is driven by information generated by the presentation of the customer contact.

Screen Pop

The most common CTI application is a screen pop. In a screen pop, the customer service platform is provided with customer information at the arrival of a phone call and begins processing the customer’s transaction at the same time as the communication begins between the customer and the agent. This transfer of customer information is called the call context information: a rich set of customer-specific data that travels with the call throughout the enterprise.

For example, a screen pop application for a cellular telephone company might be triggered based on the arrival of a phone call. It uses the customer ANI (automated number identification, or calling line ID) to do a database look up to retrieve the customer’s account information and displays this customer record for the agent. By the time the agent can say “Thank you for calling ABC Telephony Company,” the account record is on his screen and he is ready to service the customer’s request.

Agent State Control

Similar to a screen pop, CTI application control of agent state is a way to improve the agent’s workflow by integrating the service delivery platform with the communications media. A CTI application enabled for agent state can set the agent’s current work state according to the type of work being performed.

For example, a sales application might automatically send an agent to a wrap-up or after-call work state when the customer contact terminates. The agent could then enter wrap up data about that transaction or customer inquiry and (subject to a timer) have his state changed automatically back to available when the wrap up work has been completed.
Third-Party Call Control

The most advanced CTI integration projects seek a total integration of the customer service platform with the communications media. In third-party call control applications, the actual control over the teleset or other media is initiated via the software application, and coordinated with application screens or views.

For example, a financial services application might perform the transfer of a phone call to a speed-dial number designated by the application itself. In this kind of scenario, the agent could click one button to determine the appropriate destination for the transfer, save the application’s customer context, and transfer the call to the other agent.

Leveraging CTI Application Event Flow

The first step to developing a CTI-enabled application is to understand the events and requests that are at play within the CTI environment. Asynchronous events are messages sent to applications that indicate an event to which the application can respond (for example, CallBeginEvent). Requests are the mechanism that the application uses to request that a desired behavior happen (for example, TransferCall).

Asynchronous Events

The CTI environment is one of diverse servers and applications communicating over a network. This naturally leads to asynchronous, or unsolicited events – events that arrive based on some stimulus external to the user’s application. The main source of events in the CTI environment is the communications media.

Figure 1-1 depicts the stages of a typical inbound telephone call and its associated events:
The following events are generated, based on the state of the call:

- **BEGIN_CALL** event indicates that the call has entered the setup phase.
- **CALL_DELIVERED** event is generated when the call starts ringing.
- **CALL_ESTABLISHED** event is generated when the call is answered.
- **CALL_CLEARED** event is generated when the voice connection is terminated (e.g. call hung up).
- **END_CALL** event is generated when the logical call appearance (including call data) is complete.

In addition to the events and states shown in Figure 1-1, the following are typical call events used for CTI applications:

- **CALL_HELD** event is generated when the call transitions from the active to held state.
- **CALL_RETRIEVED** event is generated when the call is removed from hold.
- **CALL_TRANSFERRED** event indicates that the call has been transferred to another party.
- **CALL_CONFERENCED** event indicates that a new party has been added to the call.

The foregoing is only a brief sample of the events available via CTI OS. The complete set of events available for CTI developers is detailed in later chapters in this guide.

**Request-Response Paradigm**

In addition to being able to respond to asynchronous events, a CTI enabled application can make programmatic requests for services via the CTI interface. Specifically, the CTI application uses the request-response mechanism to perform agent state and third-party call control, and to set call context data.
The typical request-response flow for CTI uses the model shown in Figure 1-2:

*Figure 1-2  Sample Request-Response Message Flow.*

A request generated by the CTI-enabled application (CLIENT) is sent to the CTI service (SERVER), and a response message (CONF) is generated to indicate that the request has been received. In most cases if the request is successful, a follow-on event will be received indicating that the desired behavior has occurred. Detailed descriptions of this kind of request-response-event message flow are detailed in later chapters in this guide.

**Overview of CTI OS**

The Computer Telephony Integration Object Server (CTI OS) is Cisco’s next generation customer contact integration platform. CTI OS combines a powerful, feature-rich server and an object-oriented software development toolkit to enable rapid development and deployment of complex CTI applications. Together with
the Cisco CTI Server Interface, CTI OS and Client Interface Library (CIL) create a high performance, scalable, fault-tolerant three-tiered CTI architecture, as illustrated in Figure 1-3.

**Figure 1-3  CTI OS Three-Tiered Architecture Topology**

The CTI OS application architecture employs three tiers:

- The CIL is the first tier, providing an application-level interface to developers.
- The CTI OS Server is the second tier, providing the bulk of the event and request processing and enabling the object services of the CTI OS system.
- The Cisco CTI Server is the third tier, providing the event source and the back-end handling of telephony requests.

## Advantages of CTI OS as Interface to ICM

CTI OS brings several major advances to developing custom CTI integration solutions. The CIL provides an object-oriented and event driven application programming interface (API), while the CTI OS server does all the ‘heavy-lifting’ of the CTI integration: updating call context information, determining which buttons to enable on softphones, providing easy access to supervisor features, and automatically recovering from failover scenarios.
• **Rapid integration.** Developing CTI applications with CTI OS is significantly easier and faster than any previously available Cisco CTI integration platform. The same object oriented interface is used across programming languages, enabling rapid integrations in C++, Visual Basic, Java™, or any Microsoft COM compliant container environment. CTI OS enables developers to create a screen pop application in as little as five minutes. The only custom-development effort required is within the homegrown application to which CTI is being added.

• **Complex solutions made simple.** CTI OS enables complex server-to-server integrations and multiple agent monitoring-type applications. The CIL provides a single object-oriented interface that can be used in two modes: agent mode and monitor mode. See Chapter 2, “CTI OS Client Interface Library Architecture” for an explanation of these two modes.

• **Fault tolerant.** CTI OS is built upon the ICM NodeManager fault-tolerance platform, which automatically detects process failure and restarts the process, enabling work to continue. Upon recovery from a failure, CTI OS initiates a complete, system-wide snapshot of all agents, calls, and supervisors and propagates updates to all client-side objects.

### Key Benefits of CTI OS for CTI Application Developers

The CTI OS Client Interface Library (CIL) provides programmers with the tools required to rapidly develop high-quality CTI–enabled applications, taking advantage of the rich features of the CTI OS server. Every feature of CTI OS was designed with ease of integration in mind, to remove the traditional barriers to entry for CTI integrations.

• **Object-oriented interactions.** CTI OS provides an object-oriented CTI interface by defining objects for all call center interactions. Programmers interface directly with Session, Agent, SkillGroup, and Call objects to perform all functions. CIL objects are thin proxies for the server-side objects, where all the ‘heavy-lifting’ is done. The Session object manages all objects within the CIL. A UniqueObjectID identifies each object. Programmers can access an object by its UniqueObjectID or by iterating through the object collections.

• **Connection and session management.** The CTI OS CIL provides out-of-the-box connection and session management with the CTI OS Server, hiding all of the details of the TCP/IP sockets connection. The CIL also
provides out-of-the-box failover recovery: upon recovery from a failure, the CIL will automatically reconnect to another CTI OS (or reconnect to the same CTI OS after restart), reestablish the session, and recover all objects for that session.

- **All parameters are key-value pairs.** The CTI OS CIL provides helper classes to treat all event and request parameters as simply a set of key-value pairs. All properties on the CTI OS objects are accessible by name via a simple `Value = GetValue("key")` mechanism. Client programmers can add values of any type to the CTI OS Arguments structure, using the enumerated CTI OS keywords, or their own string keywords (for example, `AddItem("DialedNumber", "1234")`). This provides for future enhancement of the interface without requiring any changes to the method signatures.

- **Simple event subscription model.** The CTI OS CIL implements a publisher-subscriber design pattern to enable easy subscription to event interfaces. Programmers can subscribe to the appropriate event interface that suits their needs, or use the IAllInOne interface to subscribe for all events. All languages except Java contain subclassable event adapter classes. These classes enable programmers to subscribe to event interfaces; they only add minimal custom code for the events they use and no code at all for events they do not use.
CTI OS Client Interface Library Architecture

This chapter describes the architecture of the CTI OS Client Interface Library (CIL). The CIL is the programmer’s interface into the CTI OS system.

Object Server Architecture

CTI OS is a Server-based integration solution, which enables all of the objects to exist on the CTI OS server. The client-side objects, through which the developer can interact with the CTI OS CIL, can be conceptually thought of as thin proxies for server-side objects.

All objects are identified by a UniqueObjectID. The UniqueObjectID is the key which is used to map a server-side object and the client-side proxy (or proxies) for it. Requests made on a client-side object will be sent to the CTI OS Server, and the corresponding server-side object will service the request (Figure 2-1).
Client Interface Library Architecture

The Client Interface Library has a three-tiered architecture (Figure 2-2), which implements the functionality provided to developers. The CIL architecture is composed of the Connection layer, the Service Layer, and the Object Interface Layer. The CIL architecture also includes the custom application, which is developed by the customer to make use of the services provided by the Client interface Library.

Figure 2-2  Client Interface Library Three-Tiered Architecture
Connection Layer

The Connection layer provides basic communication and connection recovery facilities to the CIL. It creates the foundation, or bottom tier of the CIL’s layered architecture, and decouples the higher-level event and message architecture from the low-level communication link (TCP/IP sockets). The Connection layer sends and receives sockets messages to the CTI OS Server, where it connects to a server-side connection layer.

In addition to basic communication facilities, the connection layer provides fault tolerance to the CIL by automatically detecting and recovering from a variety of network failures. The Connection layer uses a heartbeat-by-exception mechanism, sending heartbeats only when the connection has been silent for some period of time, to detect network-level failures.

Service Layer

The Service layer sits between the connection layer and the Object Interface layer. Its main purpose is to translate between the low-level network packets sent and received by the connection layer and the high-level command and event messages used in the Object Interface layer. The Service layer implements a generic message serialization protocol which translates key-value pairs into a byte stream for network transmission and deserializes the messages back to key-value pairs on the receiving side. This generic serialization mechanism ensures forward-compatibility, since future enhancements to the message set will not require any changes at the Connection or Service layers.

A secondary purpose of the Service layer is to isolate the client from the network, such that network issues do not block the client and vice versa. This is done via a multi-threading model which allows user-program execution to continue without having to ‘block’ on network message sending or receiving. This prevents client applications from getting ‘stuck’ when a message is not immediately dispatched across the network, and allows messages to be received from the network even if the client application is temporarily blocked.
Object Interface Layer

The CTI Object Interface layer is the topmost layer on the CIL architecture. It consists of the group of objects (classes) that enable application developers to write robust applications for CTI in a short time. The framework can be extended to accommodate special requirements by subclassing one or more of the CTI OS object classes.

Custom Application

The custom application is the business application that is developed to integrate with the CTI OS Client Interface Library. The custom application makes use of the CIL in two ways. First, the CIL provides the object-based interface for interacting with CTI OS, to send requests for agent and call control. Second, the CIL provides an events subscription service, which the custom application will take advantage of to receive events from CTI OS.

For example, a custom application would use the Agent object to send a MakeCallRequest, and then receive a OnCallBeginEvent (and others) from the CIL’s events interface(s).

CIL Object Model

The Client Interface Library’s Object Interface layer provides a set of objects that create abstractions for all of the call center interactions supported. Client programs interact with the CIL objects by making requests from the objects, and querying the objects to retrieve properties. Figure 2-3 illustrates the CIL Object Model Object Interfaces.
Session Object

The Session object is the main object in the CIL. It controls the logical session between the client application and the CTIOS server. The Session object provides the interface to the lower layers of the CIL architecture (the Service and Connection layers), and also encapsulates the functions required to dispatch messages to all of the other objects in the CIL.

The Session object provides object management (creation, collection management, and deletion), and is the publisher for all CIL events. In addition, the Session object provides automatic fault tolerance and failover recovery.

Session Modes

A Session object can be set to work in one of two modes, Agent Mode or Monitor Mode, as explained in the following sections. The Session object maintains the state of the Session mode, and recovers the session mode during failover. The client application must set the session mode after it connects to the CTI OS Server; the Session mode remains active until the connection to the CTI OS Server is closed.
Chapter 2 CTI OS Client Interface Library Architecture

Agent Mode

A client connects to CTIOS Server in Agent Mode when it wants to receive events for a specific agent or supervisor. Once agent mode has been set, the CIL receives the events for the specified agent, as well as all call events for that agent’s calls. If the agent is also configured as a Supervisor in ICM, then the CIL receives events for all agents in the Supervisor’s team.

Monitor Mode

A client connects to the CTIOS Server in Monitor Mode when it wants to receive a programmer-specified set of events, such as all agent state events. For details of setting up a monitor mode connection, refer to the Session object’s SetMessageFilter feature.

For the complete interface specification of the Session object, see Chapter 8, “Session Object.”

Agent Object

The Agent object provides an interface to Agent functionality, including changing agent states and making calls. The agent object also provides access to many properties, including agent statistics. Depending on the Session Mode, a CIL application can have zero to many agent objects.

For the complete interface specification of the Agent object, see Chapter 9, “Agent Object.”

Call Object

The Call object provides an interface to Call functionality, including call control and accessing call data properties. Depending on the Session Mode, a CIL application can have any number of call objects.

For the complete interface specification of the Call object, see Chapter 10, “Call Object.”
SkillGroup Object

The SkillGroup object provides an interface to SkillGroup properties, specifically skill group statistics. Depending on the Session Mode, a CIL application can have zero to many SkillGroup objects.

For the complete interface specification of the SkillGroup object, see Chapter 11, “SkillGroup Object.”

Object Creation and Lifetime

The Session object maintains a collection for each class of objects it manages (e.g. Agents, Calls, SkillGroups, etc.).

Objects are created either by the programmer, or by the Session object as required to support the event flow received from the CTIOS Server. In Agent Mode, the programmer will create a single Agent object with which to login, whereas in Monitor Mode, Agent objects are created as required by the event flow. Call and SkillGroup objects are always created by the Session object.

An Agent, Call or SkillGroup object is created (by the Session) when the Session receives an event for an object (identified by its UniqueObjectID) that is not yet present at the CIL. This ensures that the CIL will always have the appropriate collection of proxy objects, one for each object on the CTIOS Server that it is using. When a new object is created, it is added to the Session object’s collection, and is accessible from the Session via the GetValue mechanism. See Chapter 8, “Session Object.”

Reference Counting

Object lifetime is controlled using reference counting. Reference counts determine if an object is still in use; that is, if a pointer or reference to it still exists in some collection or member variable. When all references to the object have been released, the object is deleted.

An application or object that will hold a reference to a CIL object must add to its reference count using the AddRef method. When the reference is no longer required, the application or object holding that reference must decrement the reference count using the Release() method. Reference counting is discussed further in Chapter 7, “CtiOs Object.”
Reference counting must be done explicitly in C++ applications (COM or non-COM). VisualBasic will perform automatic reference counting.

Call Object Lifetime

Call objects are created at the CIL in response to events from the CTIOS server. Usually, a Call object will be created in response to the OnCallBegin event, but in certain failover recovery scenarios a Call object could be created in response to an OnSnapshotCallConf event. Any call data available for the call is passed in the event, and is used to set up the Call object’s initial state and properties.

The Call object will remain valid at the CIL until the receipt of the OnCallEnd event. When the OnCallEnd event is received, the Session object will publish the event to any subscribers to the event interfaces. Applications and objects must release any remaining references to the Call object within their event handler for OnCallEnd to allow the Call object to be properly deleted. When the Call object’s OnEvent method returns after handling OnCallEnd, the Session will check the reference count for zero; if any references remain, the call object will be removed from the call object collection and will be deleted.

Agent Object Lifetime

In Agent Mode, the client programmer must create an Agent object (which causes its reference count to be incremented to one) and must pass it to the Session in the SetAgent method.

Note

In C++, the object must be created on the heap memory store so that it can exist beyond the scope of the method creating it. For COM/VB clients, this is handled automatically.

The Session will hold a reference to the Agent object as long as it is in use, but the client programmer must release the last reference to the object to prevent a memory leak.
In Monitor Mode objects are created at the CIL the first time the CIL receives an event for that agent (e.g. in an OnAgentStateChange event). When the Session receives an event for an unrecognized Agent, that new Agent is added to the Session’s collection of agents.

During application clean-up, the Session object will release its references to all agents in the Agent collection. To ensure proper memory clean-up, the programmer must release all reference to Agent objects.

**SkillGroup Object Lifetime**

A SkillGroup object is created at the CIL the first time an OnNewSkillGroupStatisticsEvent event occurs for that SkillGroup. It is added to the SkillGroup collection, and it is subsequently updated by follow-on OnNewSkillGroupStatisticsEvent events.

During application clean-up, the Session object releases its references to all skill groups in the SkillGroup collection. To ensure proper memory clean-up, the programmer must release all reference to SkillGroup objects.

**Where To Go From Here**

Subsequent chapters in this manual contain the following information:

- For information about CIL coding conventions, see Chapter 3, “CIL Coding Conventions.”
- For information about building an application using the CIL, see Chapter 4, “Building Your Application.”
- For a description and syntax of the CIL programming interfaces, see Chapters 8 through 13.
CIL Coding Conventions

This chapter discusses coding conventions used in the CTI OS Client Interface Library (CIL). Coding conventions are standard ways of performing common tasks. While the rest of this document discusses the programming interfaces available with the CIL, this chapter provides useful and practical explanation of how to program with the CIL – the glue that brings everything together.

One of the design goals of the CTI OS CIL is to make programming as easy and consistent as possible for client developers. As such, many design decisions about the CIL interfaces were made in order to keep things simple, clear, and consistent across various objects, methods, and programming environments.

This chapter discusses the following topics:

- Data types
- Asynchronous execution (error codes versus events)
- Generic interfaces with the Arguments structure
- Optional and reserved parameters
- Accessing properties and parameters with GetValue
- Adding parameters to requests with AddItem
- Setting properties with SetValue
- UniqueObjectIDs: how to identify objects
- Obtaining an object from its UniqueObjectID
- Using Button Enablement Masks
CTIOS CIL Data Types

The CTI OS Client Interface Library is designed to be a single interface, which can be used across multiple languages and environments (e.g. C++, COM, Visual Basic, Java). However, each programming language has its own native data types. Throughout this document, the interface parameters will be listed with the following standardized data types:

- **STRING**: This is a variable-length string variable. If a maximum length exists, it is listed with the parameter description.
- **INT**: This is a 32-bit wide integer.
- **UNSIGNED INT**: This is a 32-bit wide unsigned integer.
- **SHORT**: This is a 16-bit wide short integer.
- **UNSIGNED SHORT**: This is a 16-bit wide unsigned short integer.
- **BOOL**: This is a logical true or false variable. Different implementations will use different sized variables to represent this type. In COM, the VARIANT_BOOL is used. Tests of variables of this data type must be against VARIANT_TRUE and VARIANT_FALSE and not simply against 0 or 1.
- **ARGUMENTS**: This is a custom data structure used by CTI OS, which holds a variable-length set of key-value pairs.
- **ARG**: This is an individual element (value), which can be stored in an ARGUMENTS structure.

Table 3-1 describes the appropriate language specific types to which the documented type are associated.

<table>
<thead>
<tr>
<th>Documented Data Type</th>
<th>C++ Type</th>
<th>Visual Basic Type</th>
<th>COM Type</th>
<th>Java Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td>std::string OR const char</td>
<td>String</td>
<td>BSTR</td>
<td>String</td>
</tr>
<tr>
<td>INT</td>
<td>long OR int</td>
<td>Long</td>
<td>long OR int</td>
<td>int</td>
</tr>
<tr>
<td>UNSIGNED INT</td>
<td>unsigned int</td>
<td>None</td>
<td>unsigned int</td>
<td>long</td>
</tr>
<tr>
<td>SHORT</td>
<td>short</td>
<td>Integer</td>
<td>short</td>
<td>short</td>
</tr>
</tbody>
</table>

Table 3-1 CTI OS CIL Data Type
Asynchronous Program Execution

Synchronous execution is the most common programming approach used by most applications. In a synchronous execution mode, a method call will execute all of the code required to complete the request and provide return values as well as error codes. Client-server programming can be synchronous (the client application will make a blocking request and will continue execution when the request is completed) or asynchronous (the client application makes a request, and continues processing immediately, with the result of the request to follow at a later time).

CTI programming is unique in that requests are often serviced by third-party servers or applications, such as a PBX/ACD in the contact center. The asynchronous nature of CTI programming requires developers to note the distinction between an error code and the response to a request. In non-CTI programming, developers test the error codes (return values from method calls) to determine whether a method request succeeded or failed. However in a distributed architecture such as CTI OS, success or failure is often determined by some external server or component such as the PBX/ACD.

The CTI OS Client Interface Library API specifies error codes, which are return values for method calls. These error codes relate to the success or failure of the method call, but not the success or failure of the underlying operation. By the success of the method call, we mean that the parameters sent were of the correct format, that internal memory allocations were successful, and that the request was put on the send queue to be transmitted to the CTI OS Server. Generally, the CIL error code returned from method calls will be CIL_OK, indicating that the method call was made successfully. However, this does not indicate that the request was actually serviced by the CTI OS Server or successfully completed at the PBX/ACD.

Table 3-1  CTI OS CIL Data Type (continued)

<table>
<thead>
<tr>
<th>UNSIGNED SHORT</th>
<th>unsigned short</th>
<th>Integer</th>
<th>unsigned short</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>bool</td>
<td>Boolean</td>
<td>VARIANT_BOOL</td>
<td>Boolean</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>Arguments</td>
<td>Arguments</td>
<td>IArguments *</td>
<td>Arguments</td>
</tr>
<tr>
<td>ARG</td>
<td>Arg</td>
<td>Arg</td>
<td>IArg *</td>
<td>Arg</td>
</tr>
</tbody>
</table>

Asynchronous Program Execution

Synchronous execution is the most common programming approach used by most applications. In a synchronous execution mode, a method call will execute all of the code required to complete the request and provide return values as well as error codes. Client-server programming can be synchronous (the client application will make a blocking request and will continue execution when the request is completed) or asynchronous (the client application makes a request, and continues processing immediately, with the result of the request to follow at a later time).

CTI programming is unique in that requests are often serviced by third-party servers or applications, such as a PBX/ACD in the contact center. The asynchronous nature of CTI programming requires developers to note the distinction between an error code and the response to a request. In non-CTI programming, developers test the error codes (return values from method calls) to determine whether a method request succeeded or failed. However in a distributed architecture such as CTI OS, success or failure is often determined by some external server or component such as the PBX/ACD.

The CTI OS Client Interface Library API specifies error codes, which are return values for method calls. These error codes relate to the success or failure of the method call, but not the success or failure of the underlying operation. By the success of the method call, we mean that the parameters sent were of the correct format, that internal memory allocations were successful, and that the request was put on the send queue to be transmitted to the CTI OS Server. Generally, the CIL error code returned from method calls will be CIL_OK, indicating that the method call was made successfully. However, this does not indicate that the request was actually serviced by the CTI OS Server or successfully completed at the PBX/ACD.
To determine the success or failure of the underlying telephony operation requested, the CTI programmer must wait for an event confirming the success or failure of the request. To generalize the message flow model, most requests made at the CTI OS CIL will be answered with a confirmation message and/or an event message. See the object interface reference in Chapters 8-12 for details on each particular request. This type of response is called asynchronous – it can arrive at any time after the request is made, but typically requests are services in sub-second timeframes.

For each method request in the programmer’s interface sections of this document, the expected event sequence is described, so that programmers know which events to expect. In the event of a request failure, an eControlFailureConf message will be send to the client; the eControlFailureConf message will have a parameter called MessageType indicating which request failed, and a parameter called ErrorMessage, with a description of the failure cause.

For example: when sending a MakeCall request, the method will typically return CIL_OK, which means that the method call was successful. If the underlying make call request is successful, the CIL will receive several follow-on events, such as eBeginCallEvent and eServiceInitiatedEvent. If the request fails, the CIL will receive the eControlFailureConf message.

A common mistake: developers who have not previously programmed with asynchronous events might mistake the error code returned from a method call for the actual result of the request. The correct semantics are to interpret the error code as being indicative of result the method call, and to interpret the follow-on events to determine the actual result of the requested operation.

### CIL Error Codes

Whenever a method call is invoked by a custom application using the CIL, an error code is returned. The error codes returned only indicate success or failure of the method call, as indicated in the previous section.

The possible values of the error code returned from C++ and Java CIL methods are defined in Table 3-2.

---

**Note**

The numeric values listed in Tables 3-2 through 3-4 are subject to change. It is recommended that you use the error code numerations to check a given error code, rather than rely on a specific numeric value.
### Table 3-2 CIL Error Codes

<table>
<thead>
<tr>
<th>CIL Error Code</th>
<th>Numeric Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIL_OK</td>
<td>1</td>
<td>The method succeeded.</td>
</tr>
<tr>
<td>CIL_FAIL</td>
<td>0</td>
<td>The method failed.</td>
</tr>
<tr>
<td>E_CTIOS_METHOD_NO_IMPLEMENTED</td>
<td>-99</td>
<td>There is no implementation available for this method.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_PROPERTY</td>
<td>-100</td>
<td>One or more properties are invalid.</td>
</tr>
<tr>
<td>E_CTIOS_MODE_CONFLICT</td>
<td>-101</td>
<td>A conflict when setting session mode.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_EVENTID</td>
<td>-102</td>
<td>The Event ID is not valid.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_ARGUMENT</td>
<td>-103</td>
<td>The Argument is not valid.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_SESSION</td>
<td>-104</td>
<td>The Session is not valid.</td>
</tr>
<tr>
<td>E_CTIOS_UNEXPECTED</td>
<td>-105</td>
<td>An unexpected error has occurred.</td>
</tr>
<tr>
<td>E_CTIOS_OBJ_ALLOCATION_FAILED</td>
<td>-106</td>
<td>There is not enough memory available and an creation of CttiOsObject failed.</td>
</tr>
<tr>
<td>E_CTIOS_ARRAYREF_ALLOCATION_FAILED</td>
<td>-107</td>
<td>There is not enough memory available and an creation of an array of references to objects of type CttiOsObject failed.</td>
</tr>
<tr>
<td>E_CTIOS_ARGUMENT_ALLOCATION_FAILED</td>
<td>-108</td>
<td>There is not enough memory available and an creation of an object of type Arguments failed.</td>
</tr>
<tr>
<td>E_CTIOS_TARGET_OBJECT_NOT_FOUND</td>
<td>-109</td>
<td>There are no CTI OS Objects capable of processing an incoming event</td>
</tr>
<tr>
<td>E_CTIOS_PROP_ATTRIBUTES_ACCESS_FAILED</td>
<td>-110</td>
<td>An error occurred while accessing a property's attributes, System may be running out of memory.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_OBJECT_TYPE</td>
<td>-111</td>
<td>The object type is not one of the following predefined types CAgent, CCall, CSkillGroups, or CWaitObject.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_AGENT</td>
<td>-112</td>
<td>No valid agent.</td>
</tr>
</tbody>
</table>
### Table 3-2  CIL Error Codes (continued)

<table>
<thead>
<tr>
<th>CIL Error Code</th>
<th>Numeric Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTIOS_INVALID_CALL</td>
<td>-113</td>
<td>No valid call.</td>
</tr>
<tr>
<td>E_CTIOS_IN_FAIlOVER</td>
<td>-114</td>
<td>The session is recovering from a connection failure and had started the Fail Over procedure.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_DESKTOP_TYPE</td>
<td>-115</td>
<td>Indicates that the desktop type specified in the request for DeskSettings download is neither Agent or Supervisor.</td>
</tr>
<tr>
<td>E_CTIOS_MISSING_ARGUMENT</td>
<td>-116</td>
<td>Missing a required argument.</td>
</tr>
<tr>
<td>E_CTIOS_CALL_NOT_ON_HOLD</td>
<td>-117</td>
<td>Call is not on hold.</td>
</tr>
<tr>
<td>E_CTIOS_CALL_ALREADY_ON_HOLD</td>
<td>-118</td>
<td>Call is already on hold.</td>
</tr>
<tr>
<td>E_CTIOS_CALL_NOT_ALERTING</td>
<td>-119</td>
<td>Call is not in alert state, it can not be answered.</td>
</tr>
<tr>
<td>E_CTIOS_AGENT_NOT_LOGIN</td>
<td>-120</td>
<td>Agent is not logged in.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_METHOD_PARAMETER</td>
<td>-121</td>
<td>The input parameter is invalid.</td>
</tr>
</tbody>
</table>

C++ methods can return the additional error codes listed in Table 3-3.

### Table 3-3  C++ Only CIL Error Codes

<table>
<thead>
<tr>
<th>C++ CIL Error Code</th>
<th>Numeric Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTIOS_UNKNOWN</td>
<td>-122</td>
<td>The cause of this error is unknown.</td>
</tr>
<tr>
<td>E_CTIOS_OUT_OF_MEMORY</td>
<td>-123</td>
<td>Failed to allocate new memory.</td>
</tr>
<tr>
<td>E_CTIOS_PORT_UNAVAILABLE</td>
<td>-124</td>
<td>The specified port is not available for use.</td>
</tr>
</tbody>
</table>
### Table 3-3  C++ Only CIL Error Codes (continued)

<table>
<thead>
<tr>
<th>C++ CIL Error Code</th>
<th>Numeric Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTIOS_SM_SESSION_TERMINATED_ABNORMALLY</td>
<td>-125</td>
<td>The Silent Monitor session was terminated abnormally.</td>
</tr>
<tr>
<td>E_CTIOS_SM_REJECTED_ALREADY_IN_SESSION</td>
<td>-126</td>
<td>The request was rejected because there is an active silent monitor session in progress.</td>
</tr>
<tr>
<td>E_CTIOS_SM_PACKET_SNIFER_NOT_INSTALLED</td>
<td>-127</td>
<td>The packet sniffer is not present in the system; verify installation.</td>
</tr>
<tr>
<td>E_CTIOS_PACKET_SNIFER_FAILED</td>
<td>-128</td>
<td>An error occurred in the packet sniffer.</td>
</tr>
<tr>
<td>E_CTIOS_SOCKET_CALL_FAILED</td>
<td>-129</td>
<td>A CTI OS socket call failed.</td>
</tr>
<tr>
<td>E_CTIOS_MEDIA_TERMINATION_NOT_INSTALLED</td>
<td>-130</td>
<td>EVVBU Media Termination component in the system, verify installation.</td>
</tr>
<tr>
<td>E_CTIOS_MT_UNKNOWN_CODEC</td>
<td>-131</td>
<td>Specified CODEC is not supported.</td>
</tr>
<tr>
<td>E_CTIOS_MEDIA_TERMINATION_FAILED</td>
<td>-132</td>
<td>An error occurred in the Media Termination Packet Decoder.</td>
</tr>
<tr>
<td>E_CTIOS_SNIFER_NO_PACKETS_RECEIVED</td>
<td>-133</td>
<td>The Sniffer has not received any IP packets.</td>
</tr>
<tr>
<td>E_CTIOS_SNIFER_FAILED_TO_OPEN_DEVICE</td>
<td>-134</td>
<td>The Sniffer failed to open the networking device.</td>
</tr>
<tr>
<td>E_CTIOS_SNIFER_FAILED_TO_SET_FILTER</td>
<td>-135</td>
<td>The Sniffer failed when setting the packet filter.</td>
</tr>
<tr>
<td>E_CTIOS_ERROR_IN_PACKET_FILTER</td>
<td>-136</td>
<td>The packet filter expression is incorrect.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_MONITORED_IP_ADDRESS</td>
<td>-137</td>
<td>The IP Address specified for the monitored device (IP Phone) is not valid.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_SNIFER_OBJECT</td>
<td>-138</td>
<td>Invalid Sniffer object.</td>
</tr>
</tbody>
</table>
### Table 3-3  C++ Only CIL Error Codes (continued)

<table>
<thead>
<tr>
<th>C++ CIL Error Code</th>
<th>Numeric Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTIOS_INVALID_DECODER_OBJECT</td>
<td>-139</td>
<td>Invalid Decoder object.</td>
</tr>
<tr>
<td>E_CTIOS_NO_SM_SESSION_IN_PROGRESS</td>
<td>-140</td>
<td>There are no Silent Monitor Sessions in progress.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_SILENT_MONITOR_SESSION</td>
<td>-141</td>
<td>The specified Silent Monitor session does not exist.</td>
</tr>
<tr>
<td>E_CTIOS_FAILED_REMOVING_SILENT_MONITOR_SESSION</td>
<td>-142</td>
<td>Silent Monitor Session was not removed from the collection.</td>
</tr>
<tr>
<td>E_CTIOS_IP_PHONE_INFORMATION_NOT_AVAILABLE</td>
<td>-143</td>
<td>There is no information available about the IP Phone.</td>
</tr>
<tr>
<td>E_CTIOS_PEER_NOT_ENABLED_FOR_SILENT_MONITOR</td>
<td>-144</td>
<td>The peer application is not enabled for Silent Monitor</td>
</tr>
<tr>
<td>E_CTIOS_NOT_ENABLED_FOR_SILENT_MONITOR</td>
<td>-145</td>
<td>This application is not enabled for Silent Monitor.</td>
</tr>
<tr>
<td>E_CTIOS_COM_OBJ_ALLOCATION_FAILED</td>
<td>-200</td>
<td>CoCreateInstance failed to create a COM object wrapper for a CIL Object (Session, Agent, Call, Skill, etc.).</td>
</tr>
<tr>
<td>E_CTIOS_COM_CORRUPTED_REGISTRY</td>
<td>-201</td>
<td>A COM component failed to access data from the registry.</td>
</tr>
<tr>
<td>E_CTIOS_COM.DialPad_FAIL_TO_LOAD</td>
<td>-202</td>
<td>The Dial Pad common dialog was not created and CoCreateInstance failed.</td>
</tr>
<tr>
<td>E_CTIOS_COM_CONV_COMPTR_TO_CPPPTR_FAILED</td>
<td>-203</td>
<td>Failed converting COM pointer to C++ pointer.</td>
</tr>
<tr>
<td>E_CTIOS_COM.NOT_INITIALIZED</td>
<td>-204</td>
<td>The MS COM library is not initialized. Invoke CoInitialize(...)</td>
</tr>
</tbody>
</table>

Java methods can return the additional error codes listed in Table 3-4.
Note

If a method that is supposed to trigger an event returns an error code, you need to check this return value before continuing to wait for events. Depending on the error code, the event you were waiting for may not be triggered.

## COM Error Codes

For applications using the CTI OS CIL for COM, the Microsoft COM layer adds a level of error detection and provides additional error codes, called HRESULTs. For COM method calls in C++, the HRESULT is returned from the method call, and indicates success or failure of the method call. The CIL error code is also returned, but as an [out, retval] parameter. For example:

### Table 3-4  Java Only CIL Error Codes

<table>
<thead>
<tr>
<th>Java CIL Error Code</th>
<th>Numeric Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTIOS_SESSION_DISCONNECT_PENDING</td>
<td>-300</td>
<td>A disconnect is already pending.</td>
</tr>
<tr>
<td>E_CTIOS_SESSION_NOT_CONNECTED</td>
<td>-301</td>
<td>The session is not connected.</td>
</tr>
<tr>
<td>E_CTIOS_AGENT_ALREADY_IN_SESSION</td>
<td>-900</td>
<td>An object for this agent already exists in the session.</td>
</tr>
<tr>
<td>E_CTIOS_SET_AGENT_SESSION_DISCONNECT_REQUIRED</td>
<td>-901</td>
<td>Session must be disconnected before operation.</td>
</tr>
<tr>
<td>E_CTIOS_SERVICE_SEND_MESSAGE_FAILED</td>
<td>-902</td>
<td>Could not send message. Session may not be connected.</td>
</tr>
<tr>
<td>E_CTIOS_CALL_ALREADY_CURRENT_IN_SESSION</td>
<td>-903</td>
<td>An object for this call is already set as current in the session.</td>
</tr>
<tr>
<td>E_CTIOS_LOGIN_INCONSISTENT_ARGUMENTS</td>
<td>-904</td>
<td>The AgentID and/or PeripheralID provided to a Login call do not match the properties set on the Agent object when SetAgent() was called.</td>
</tr>
</tbody>
</table>
// COM Example in C++
int errorCode = 0;
HRESULT hr = pCall->Answer(&errorCode);
if (errorCode==CIL_FAILED)
    printf("An error has occurred while answering the call.");

In Visual Basic, HRESULT values are hidden under the covers. When an error occurs, a Visual Basic exception is thrown, which can be caught using the On Error: construct. The CIL error code is returned as the result of the method call:

' VB example:
On Error GoTo Error_handler
Dim errorCode as Long

errorCode = pCall.Answer
If ErrorCode = CIL_FAILED
    Debug.print "An error has occurred."

The complete set of HRESULT values is defined by Microsoft in the header file winerror.h. The most common HRESULT values that might be seen by CTI OS developers are listed in Table 3-5:

Table 3-5 COM Error Codes

<table>
<thead>
<tr>
<th>COM Error Code</th>
<th>Numeric Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>0x00000000</td>
<td>The method succeeded.</td>
</tr>
<tr>
<td>S_FALSE</td>
<td>0x00000001</td>
<td>The method succeeded, but something unusual happened.</td>
</tr>
<tr>
<td>E_FAILED</td>
<td>0x80000008</td>
<td>The method failed.</td>
</tr>
<tr>
<td>REG_DB_E_CLASSNOTREG</td>
<td>0x80040143</td>
<td>The class was not found in the registry. You will need to run regsvr32.exe on the DLL file to register it.</td>
</tr>
</tbody>
</table>

**Generic Interfaces**

One of the main design goals of CTI OS was to enable future enhancements to the CTI OS feature set without breaking existing interfaces. To accomplish this, a parameter for almost every method and event will be an Arguments array containing the actual parameters needed. Therefore, parameters may be added or
deleted in future versions without affecting the signature of the method or event. This provides the benefit to developers that code developed to work with one version of the CTI OS developer’s toolkit will work with future versions without requiring any code changes on the client’s side (except to take advantage of new features). For example, CTI OS will automatically send a new parameter in the Arguments array for an event, without requiring an interface or library code change. The dilemma of creating a generic interface is solved by using generic mechanisms to send parameters with events and request, and to access properties.

Arguments

The CTI OS developer’s toolkit makes extensive use of a new data structure (class) called Arguments. Arguments is a structure of key-value pairs that supports a variable number of parameters and accepts any user-defined parameter names. For any given event, the arguments structure allows the CTI OS Server to send the CIL any new parameters without requiring client side changes. Similarly, for any request, the programmer can send any new parameters, without any changes to the underlying layers.

Example of using Arguments in a Visual Basic MakeCall request:

```vba
Dim args As New Arguments
args.AddItem "DialedNumber", dialthis.Text
If Not 0 = Len(callvar1.Text) Then
    ' set callvar1
    args.AddItem "CallVariable1", callvar1.Text
End If

' send makecall request
m_Agent.MakeCall args, errorcode
```

Java example:

```java
Arguments args = new Arguments();
args.SetValue(CtiOs_IkeywordIDs.CTIOS_DIALEDNUMBER, "12345");
args.SetValue(CtiOs_IkeywordIDs.CTIOS_CALLVARIABLE1, "MyData");
int iRet = m_Agent.MakeCall(args);
```

The Arguments structure can store and retrieve all native C/C++/Visual Basic and Java types, as well as nested Arguments structures.
CTI OS makes extensive use of generic data abstraction. The CTI OS CIL objects, as well as the Arguments structure, store all data by key-value pair. Properties and data values in CTI OS are accessible through a generic mechanism called GetValue. For a list of the different GetValue methods, see Chapter 7, “CtiOs Object” or Chapter 12, “Helper Classes.” The GetValue mechanism provides for the retrieval of any data element based on its name. This enables the future enhancement of the data set provided for event parameters and object properties without requiring any interface changes to support new parameters or properties. GetValue supports use of string keywords, as shown in the following examples:

// C++
string sAgentID;
args.GetValueString("AgentID", &sAgentID);

'Visual Basic
Dim sAgentID As String
sAgentID = args.GetValueString "AgentID"

//Java
String sID =
args.GetValueString(CtiOs_IkeywordIDs.CTIOS_AGENTID);
Integer IPeriph =
args.GetValueIntObj(CtiOs_IkeywordIDs.CTIOS_PERIPHERALID);

if (IPeriph == null)
// Error accessing Peripheral ID! Handle Error here
else
    iPeriph = IPeriph.intValue();

CTI OS defines a set of well-known keywords for event parameters and properties. The well-known keywords are of type string and are listed throughout this document with the methods and events for which they are valid. The complete set of valid keywords are listed in the C++ header file, ctioskeywords.h, and are provided in the COM (Visual Basic) type library as well. Java CIL keywords are listed in the Javadoc in the description of the CtiOs_IKeywordIDs interface.
Setting Request Parameters with AddItem

Similar to accessing parameters and properties with the GetValue methods, the CTI OS CIL provides an extensible mechanism to add parameters to requests. The AddItem mechanism, available in the CTI OS Arguments class, enables adding parameters of any known type to the Arguments structure as a key-value pair. For each request (method) in the CIL Object Interface, the required and optional parameters are listed in the chapter on that particular object. Calling AddItem with an unrecognized keyword is allowed but it will be ignored by the CTI OS Server.

AddItem, similar to GetValue, supports string keywords and enumerated names:

```
// C++
string sAgentID = "22866";
args.AddItem("AgentID", sAgentID);
args.AddItem(CTIOS_AGENTID, sAgentID); // alternative
args.AddItem(ekwAgentID, sAgentID); // alternative

'Visual Basic
Dim sAgentID As String
sAgentID = "22866"
args.AddItem "AgentID", sAgentID
```

Setting Object Properties with SetValue

The CIL also provides an extensible mechanism to set properties on CTI OS Client Interface Objects. The SetValue mechanism, available on the CIL Interface Objects (as well as the CTI OS Arguments class), enables setting properties of any known type to the object as a key-value pair.

SetValue, similar to GetValue and AddItem, supports string keywords and enumerated names:

```
// C++
Agent a;
a.SetValue("AgentID", "22866");
a.SetValue(CTIOS_AGENTID, "22866"); // alternative
a.SetValue(ekwAgentID, "22866"); // alternative

'Visual Basic
Dim a As Agent
a.SetValue "AgentID", "22866"
```
//Java. Note use of the CTIOS_AGENTID version of keywords.
String sAgentID = "22866";
Args.SetValue("AgentID", sAgentID);
Args.SetValue(CtiOs_IkeywordIDs.CTIOS_AGENTID, sAgentID);  // alternative
Args.SetValue(ekwAgentID, sAgentID);

The complete syntax and usage of the GetValue, AddItem, and SetValue methods is detailed in Chapter 7, “CtiOs Object.” The Arguments structure is detailed in Chapter 12, “Helper Classes.”

**UniqueObjectID**

The CTI OS Server creates and manages the CTI OS objects, representing all interactions for the contact center. The CTI OS Server and CIL use the UniqueObjectID field to match up a CTI OS object on the CIL with the corresponding object on the Server.

The UniqueObjectID is a variable-length string which can uniquely identify the object within the current context of the CTI OS Server and the ICM Enterprise. The UniqueObjectID is composed of an object type (e.g. call, agent, skillgroup, etc.), and two or more additional identifying fields. Table 3-6 explains the composition of the UniqueObjectID.

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Sample UniqueObjectID</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call Object</td>
<td>call.5000.202.23901</td>
<td>The call object is uniquely identified by its PeripheralID (5000, generated by ICM), ConnectionCallID (202, generated by the PBX/ACD), and its ConnectionDeviceID (23901, generated by the PBX/ACD).</td>
</tr>
<tr>
<td>Agent Object</td>
<td>agent.5000.22866</td>
<td>The agent object is uniquely identified by its PeripheralID (5000, generated by ICM), and its agent ID.</td>
</tr>
</tbody>
</table>
The CTI OS UniqueObjectID is not the same as the ICM globally unique 64 bit key used in the ICM historical databases (herein called the ICMEnterpriseUniqueID), which exists only for calls. The ICMEnterpriseUniqueID stays with the call even when the call is transferred between call center sites, whereas the UniqueObjectID for a call is specific to its site (by PeripheralID, ConnectionCallID, and ConnectionDeviceID).

The ICMEnterpriseUniqueID in CTI OS takes the form of a variable-length string with the form

“icm.routercallkeyday.routercallkeycallid”

where routercallkeyday is the field Day in the ICM Route_Call_Detail and Termination_Call_Detail tables, and routercallkeycallid is the field RouterCallKey in the ICM Route_Call_Detail and Termination_Call_Detail tables.
The CTI OS server enables certain types of monitor mode applications that track the pre-call notification event (eTranslationRouteEvent or eAgentPrecallEvent) and seek to match the call data with the arrival of an eCallBeginEvent.

To do so, the application will receive the pre-call notification (for calls routed by ICM (either pre-route, post-route, or translation route), and create a record (object) using the ICMEnterpriseUniqueID field as the unique key. Later, when the call arrives at the ACD, and is queued or targeted (by the ACD) for a specific agent, the application can match the saved record (object) with the incoming call by the ICMEnterpriseUniqueID field. The following events will contain the ICMEnterpriseUniqueID that can be used to associate a call with the saved call information:

- eCallBeginEvent
- eCallDataUpdateEvent
- eSnapshotCallConf
- eCallEndEvent

### Obtaining Objects from UniqueObjectIDs

Client applications written to take advantage of the CIL can use the UniqueObjectID to obtain a pointer (in C++ or COM for C++) or a reference (in Visual Basic or Java) to the underlying object.

The CIL Session object provides access to the object collections via several methods, including GetObjectFromObjectID. GetObjectFromObjectID takes as a parameter the string UniqueObjectID of the desired object, and returns a pointer to the object. Since this mechanism is generic, and does not contain specific information about the object type retrieved, the pointer (or reference) returned is a pointer or reference to the base class: a CCtiosObject* in C++, an Object in Visual Basic, or an IDispatch* in COM for C++.

**Note**

The GetObjectFromObjectID method will perform an AddRef() on the pointer before it is returned to the programmer.

**C++ example:**

```cpp
string sUniqueObjectID = "call.5000.101.23901";
Ccall * pCall = NULL;
```
Using Button Enablement Masks

The CTI OS Server provides a rich object-level interface to the CTI interactions of the contact center. One of the features the CTI OS Server provides is to evaluate all of the telephony events, and map them to the features permitted by the vendor-specific implementation. The CTI OS Server provides a peripheral-independent mechanism for clients to determine which requests are valid at any given time by using a bitmask to indicate which requests are permitted.

For example, the only time when it is valid to answer a call is when the ENABLE_ANSWER bit in the enablement mask is set to the on position. The following C++ example depicts this case:

```cpp
void EventSink::OnCallDeliveredEvent(Arguments& args)
{
  unsigned int unBitMask = 0;
  if (args.IsValid("EnablementMask"))
  {
    args.GetValueInt("EnablementMask", & unBitMask)
    //do bitwise comparison
    If(unBitMask & ENABLE_ANSWER)
      m_AnswerButton.Enable();
  }
}
```

Visual Basic example:

```vbnet
Public Sub m_Session_OnCallDeliveredEvent(ByVal pArguments as CTIOSClientLib.Arguments)
  Dim nBitMask as Integer
  If pArguments.IsValid "EnablementMask" Then
```
Using Button Enablement Masks

```plaintext
nBitMask = pArguments.GetValueInt "EnablementMask"
End If

' do bitwise comparison
  If nBitMask And ENABLE_ANSWER Then
    m_AnswerButton.Enable
  End If
End Sub

Java example:

void OnCallDeliveredEvent(Arguments args)
{
  Long LMask =
  args.GetValueUIntObj(CtiOs_IKeywordIDs.CTIOS_ENABLEMENTMASK);
  if (null!=LMask)
  {
    long lMask = LMask.longValue();
    if ((lMask & CtiOs_Enums.ButtonEnablement.ENABLE_ANSWER) == 0)
      m_AnswerButton.setEnabled(false);
    else
      m_AnswerButton.setEnabled(true);
  }
}

The advantage of using this approach is that all of the peripheral-specific details of enabling and disabling buttons is determined in a central location – at the CTI OS Server. This allows future new features to be enabled, and software bugs to be corrected in a central location, which is a great benefit for deploying future releases.

For any given event, the CTI OS Server calculates the appropriate button enablement bitmask, and sends it to the CIL with the event parameters. The button enablement bit masks are discussed in detail in Chapter 6, “Event Interfaces and Events.” You can use these masks to write a custom softphone-type application without writing any custom code to enable and disable buttons. This is also the approach used internally for the CTI OS ActiveX softphone controls.
Building Your Application

This chapter discusses how to build your custom CTI application to use the CTI OS Client Interface Library. Specifically, this chapter will help translate the choice of programming language and environment into a set of steps you will need to take to be able to reference CTI OS CIL components in your application and to be able to compile (and, if necessary link) your application.

This chapter is organized in sections according to the programming language and interface you will be using:

- **ActiveX Controls.** This section covers using the CTI OS ActiveX controls in a COM container such as Visual Basic.
- **COM CIL in Visual Basic.** This section covers the steps required to reference the CIL’s COM components in a Microsoft Visual Basic application.
- **COM CIL in C++.** This section covers the steps required to use the CIL’s COM components in a Microsoft Visual C++ application.
- **C++ CIL using static libraries.** This section covers the steps required to reference the CIL’s C++ classes in your application, and how to link the C++ static library files into a Microsoft Visual C++ application.
- **Java CIL libraries.** This section covers considerations for installing and using the Java CIL libraries.
Creating an integration between your application and CTI OS via the CIL is fairly straightforward. The first step is to articulate the desired behavior, and to create a complete design specification for the integration.

Planning and Designing Your Integration

Good design depends upon understanding how CTI will fit into your application and work flow. Your requirements analysis and design process should address the following points, as they relate to your specific application:

- **Start with the call flow.** What kind of call processing is done before calls are targeted for a specific skill? Determine how CTI data can be collected from the caller before the call arrives at an agent.

- **Study the agent’s workflow.** What are the points where CTI will be able to make the workflow easier and faster? Build a business case for the CTI integration.

- **Evaluate what will CTI do for your application.** A good approach is to make a list in order of priority (e.g. screen pop, then call control) and then design and implement features in that order.

- **Design how CTI should work within your application.** What are the interaction points? Get specific as to which screen will do the interacting, and which data values should be sent between your application and the CTI OS platform.

- **Determine when the application should connect to the CTI OS Server.** Some applications will be server-type integrations that will connect at startup, specify a monitor-mode event filter, and stay connected permanently. Agent-mode applications will connect up when a specific agent begins his or her work shift.

- **Clean up when you’re done.** When and how does the application stop? Some applications will stay up and running permanently, while others will have a defined runtime, such as the agent’s workday or shift. For server-type applications without a specified stopping point, create an object lifetime model and procedure for recovering no-longer-used resources. For
applications with a specific stopping point, determine what kind of clean up needs to be done when the application closes (e.g. disconnect from server, release resources).

What Language and Interface to Use

The CTI OS Client Interface Library API comes in programming languages, each with benefits and costs. The choice of interface is important to direct you through this developer’s guide, since this guide addresses the CIL API for the C++ and COM programming environments.

The main decision point in choosing which API to use will depend on your workstation operating system, your existing applications, and the language skills of your developers.

- **ActiveX Controls.** The CTI OS ActiveX controls are the appropriate choice for creating a rapid “drag-and-drop” integration of CTI and third-party call control with an existing desktop application. The CTI OS ActiveX controls are appropriate choice for developing a CTI integration with any fully ActiveX-compliant container, such as Microsoft Visual Basic, Microsoft Visual C++, or any other container that fully supports ActiveX features (e.g. Powerbuilder, Delphi, and many third-party CRM packages). The ActiveX controls will be the easiest to implement in graphical environments, and will help achieve the fastest integrations by providing a complete user interface. All CTI OS ActiveX components are distributed via dynamic link library files (.dll), which only have to be registered once to work on any Microsoft Windows platform. These components are not appropriate for non-Windows environments.

- **COM.** The CTI OS Client Interface Library for COM (Microsoft’s Component Object Model) is the appropriate choice for developing a CTI integration with any COM-compliant container, such as Microsoft Visual Basic, Microsoft Visual C++, or any other container that fully supports COM features, such as Microsoft Internet Explorer or Visual Basic for Applications scripting languages. The COM CIL will be the easiest to implement in scripting environments, and will help achieve the fastest integrations requiring a custom or non-graphical user interface. All CTI OS components are distributed via dynamic link library files (.dll), which only have to be registered once to work on any Microsoft Windows platform. These components are not appropriate for non-Windows environments.
Testing CTI Applications

Testing is often characterized as the most time-consuming part of any application development process.

Developing a Test Plan

Testing CTI applications requires a detailed test plan, specific to the business requirements set forth in the requirements gathering phase of the project. The test plan should list behaviors (test cases) and set forth requirements to prove that each test case is successfully accomplished. If a test case fails, it should be investigated and corrected (if appropriate) before proceeding to the next phase of testing.

It is recommended that you perform (at minimum) the following test phases:

- **Unit Testing.** In a unit test, you ensure that all of the new code units can execute properly. Each component will operate correctly based on the input, and produce the correct output. An example of a unit test would be to ‘stub-in’ or hardcode the expected screen-pop data, and ensure that all of the screens come up properly based on this data.

- **Integration Testing.** In an integration test, you ensure that all of the new components work together properly. The physical connections and data passing between the layers and servers involved in the system are tested. An example of an integration test would be testing your client application with the CTI OS server, to ensure that data can be passed correctly through the components.
• **System Testing.** In a system test, you ensure that the correct application behavior is exhibited. An example of a system test would be to make a phone call to a VRU, collect the appropriate caller information, transfer the call to an agent, and ensure that the screen pop arrives correctly.

• **User Acceptance Testing.** In a user acceptance test, you ensure that your application has met all of the business requirements set forth by your analysis and design process. An example of a user acceptance test would be to try out your new application with real agents, and ensure that it satisfies their requirements.

**Test Environment**

The CTI OS Software Development Toolkit (SDK) CD media includes a CTIServerSimulator that can be used for application development and demo purposes. It has the capability to roughly simulate a Lucent PBX/ACD or a Cisco IPCC environment. Documentation on how to configure and use the simulator can be found on the CTI OS CD in the directory Tools\Simulator.

---

*Note*

This simulator is *only* appropriate for preliminary testing of client applications. Because it does not fully replicate the behavior of the actual switch environment, the simulator should not be used for any type of QA testing. To ensure proper design conformance and ensure the correctness of the application, the CTI application *must* be tested with the actual telephony environment in which it will run. This enables the event flow and third-party control components, which are driven by the switch- and implementation-specific call flow, to be properly and thoroughly tested.

**Using the Samples**

The CTI OS Software Development Toolkit (SDK) is distributed with a rich set of sample applications on the CD media. These samples demonstrate several working applications that use the CTI OS Client Interface Library API. The samples are organized by programming language and demonstrate the syntax and usage of the API, as well as ‘real-world’ uses of the CIL API to build applications.
For many developers, these samples will form the foundation of your custom application. The samples are available for you to customize and distribute as part of your finished product.

Table 4-1 lists the sample programs on the CTI OS Toolkit CD.

Table 4-1 CTI OS Toolkit Sample Programs

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplephone</td>
<td>CTIOSToolkit\Samples\Com\VB</td>
<td>A simple softphone application build in VB by dragging and dropping the CTIOS ActiveX controls (no VB code).</td>
</tr>
<tr>
<td>BlendedAgentPhone</td>
<td>CTIOSToolkit\Samples\Com\VB</td>
<td>A softphone application that demonstrates Outbound Option (formerly Blended Agent) functionality.</td>
</tr>
<tr>
<td>Vbphone</td>
<td>CTIOSToolkit\Samples\Com\VB</td>
<td>A Visual Basic program interfacing to CTI OS via the COM interface. Sending Requests and handling events is demonstrated. No ActiveX controls are used for this sample.</td>
</tr>
<tr>
<td>Comphone</td>
<td>CTIOSToolkit\Samples\Com\C++</td>
<td>A C++ program building a softphone application interfacing to CTIOS via the COM interface (ctiosclient.dll). Demonstrates sending requests and event handling.</td>
</tr>
<tr>
<td>AllAgents</td>
<td>CTIOSToolkit\Samples\MonitorSamples\VB</td>
<td>A Visual Basic program demonstrating a monitor mode application. This program lists all agents in a grid along with current state updates and even allows the manipulation of agent states (logout, ready, not ready)</td>
</tr>
<tr>
<td>AllCalls</td>
<td>CTIOSToolkit\Samples\MonitorSamples\VB</td>
<td>Similar to AllAgents but lists calls instead of agents.</td>
</tr>
<tr>
<td>AllAgentsMonitor</td>
<td>CTIOSToolkit\Samples\MonitorSamples\VB</td>
<td>Similar to AllAgents; in addition, demonstrates Silent Monitor functionality.</td>
</tr>
</tbody>
</table>
Chapter 4  Building Your Application

Using The CTI OS ActiveX Controls

This section discusses the steps involved in building CTI OS Applications with Microsoft Visual Basic using the CTI OS ActiveX controls.

Building a Simple Softphone with ActiveX Controls

To use the CTIOS ActiveX controls, the ActiveX controls need to be copied on the target system and registered with Windows. This is accomplished by the CTIOS toolkit install, as well as the CTIOS Agent and Supervisor installs. See the section entitled “Deployment” for details.

Once Visual Basic is launched, you can use the ActiveX controls by selecting them via the Components dialog (CTRL-T from the keyboard in VB or Project->Components via the menu).

Note

Note: If the CTIOS ActiveX controls are not listed as shown in Figure 4-1, the files are either not copied on the target system or the controls were not properly registered.
Once the CTIOS ActiveX controls have been selected in the components dialog, they should be visible in the Visual Basic ToolBox. If the ToolBox is not visible, it can be activated via the menu (View->ToolBox). The ActiveX components can now be dragged and dropped onto the VB form. For a softphone application, it is useful to start with the CallAppearanceCtl (see Figure 4-2).
Figure 4-2  Microsoft Visual Basic Screen with the CTIOS ActiveX controls.

On the very left the “ToolBox” is visible showing some of the CTI OS ActiveX icons. On the form, the CallAppearanceCtl has been dragged and dropped.

For a complete description of the ActiveX controls see Chapter 5, “CTI OS ActiveX Softphone Controls.” Figure 4-3 shows a Simple Softphone application, which is also included as a sample on the CTIOS CD.
Once all ActiveX controls are placed on the phone, you can create an executable in Visual Basic via File->Make <filename>.exe.

Adding a Hook for Screenpops

This simple phone application did not require any Visual Basic coding at all. A user may choose to add some custom VB code to add a hook for screenpops. For example, a user may want to retrieve CallVariables, which are passed along with certain call events.
CTIOS SessionResolver

A CTIOS Client application connects to CTIOS with a Session object (see Chapter 8, “Session Object”). Depending on the application, a client can use one or more Session objects. For most softphone applications, however, it is useful to employ only a single Session object.

If one chooses to write a plain Visual Basic program not using ActiveX controls, a Session object can be created and used directly (see samples).

However, in the case of a Visual Basic application built with the ActiveX controls, all ActiveX controls must use the same session object. The ActiveX controls accomplish this by retrieving a pointer to the same session object via the SessionResolver. The Visual Basic program hosting the ActiveX can obtain the same session object by using the SessionResolver.GetSession method to retrieve a session named "".

Sample VB code to Retrieve CallVariable1

The following sample VB code will retrieve the common session and just listen for a CallEstablishedEvent occurring in that session. If a CallEstablishedEvent occurs, it will retrieve CallVariable 1 and put it in the Windows Clipboard (from where it can be retrieved via CTRL-v or be used by other applications).

This code uses the COM CIL Interfaces and therefore, needs the following references: CTI OS Client, CTI OS Arguments and CTI OS Session Resolver. The references are shown in Figure 4-4 (in Visual Basic, select Project->[References]).
Figure 4-4  CTI OS References Needed for Visual Basic COM Programming

VB sample for a simple CTIOS phone
' needs references to CTIOSCLIENTLib CTIOSSESSIONRESOLVERLib and CTIOSARGUMENTSLib
'
' dim CTIOS session interface
' the session interface handles connect, setagent and others
Dim WithEvents m_session As CTIOSCLIENTLib.Session
'
' the sessionresolver is needed to retrieve the session pointer
Dim m_sessionresolver As CTIOSSESSIONRESOLVERLib.SessionResolver

Private Sub Form_Load()
' instantiate the sessionresolver
Set m_sessionresolver = New
CTIOSSESSIONRESOLVERLib.SessionResolver
'
' ActiveX controls use the session named "" - blank
' since the ActiveX controls do the connection and login,

Using the COM CIL in Visual Basic

Building a custom CTI application in Visual Basic (or adding CTI functionality to your existing Visual Basic application) is very simple, but requires basic knowledge of referencing and using COM components in Visual Basic. All of the CIL components for COM are distributed as COM Dynamic Link Libraries (COM DLL).

In order to be accessible to COM containers, including Visual Basic, COM components must be registered with Windows. The components that you will require for programming in Visual Basic are:

```vbnet
' all we do is listen for events
Set m_session = m_sessionresolver.GetSession(""
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Call m_sessionresolver.RemoveSession(""
End Sub

Private Sub m_session_OnCallEstablished (ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    GetCallVariable1 pIArguments
End Sub

Function GetCallVariable1(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    Dim m_uid As String
    m_uid = pIArguments.GetStringValue("Uniqueobjectid")
    Dim m_call As CTIOSCLIENTLib.Call
    Set m_call = m_session.GetObjectFromObjectID(m_uid)
        ' retrieve callvar1
    Dim m_callvar1 As String
    m_callvar1 = m_call.GetStringValue("Callvariable1")
    ' copy call variable1 to the clipboard
    Clipboard.SetText m_callvar1
End Function
```
CTI OS Client library (CTIOSClient.dll). This is the main CIL library for COM. The objects available in this library are described fully in the chapters that follow, “Session Object,” “Agent Object,” “Call Object,” and “SkillGroup Object.”

CTI OS Arguments Library (arguments.dll). The Arguments helper class is used extensively in CTI OS, and is described fully in the chapter “Helper Classes.”

CTI OS Session Resolver (ctiosessionresolver.dll). The Session Resolver is used to share a single CTI OS Session across multiple compilation units (e.g. DLL files). It is only required in applications that use ActiveX controls or mix ActiveX controls with custom code.

Referencing COM Components in Visual Basic

To make use of these objects in your CTI application, Visual Basic uses the concept of referencing COM components. To add or remove references within a Visual Basic application, go to the Project / References menu. The following Project References window will be displayed:
In the Project References menu, select the required COM DLLs by checking the appropriate check boxes. The required library names are:

- CTIOSClient 1.0 Type Library.
- CTIOSArguments 1.0 Type Library.

Click on the OK button to exit the Project Reference screen, and save your project.

**Registering for Events in Visual Basic**

To start using the CTI OS Client Interface Library in your application, you will need to declare the Session object and register for events.

In your code window, you will also need to declare (Dim) the Session object, and register for COM events from it. The following sample code demonstrates the syntax required to declare the Session object and register for COM events in Visual Basic:

```vbnet
Dim WithEvents session As CTIOSCLIENTLib.session
```

In addition, you will need to tell Visual Basic to create the Session object when your application loads, and to destroy the Session object when your application closes:

```vbnet
Private Sub Form_Load()
    Set session = New session
End Sub

Private Sub Form_Terminate()
    Set session = Nothing
End Sub
```

Your Visual Basic application now references the CTI OS Client Library and is registered to receive COM events.

**Next Steps**

- For detailed information on the CTI OS client start up and shut down sequence, see the section “Disconnecting from CTI OS Server”.
- For detailed information on the CTI OS Client Interface Library objects, see Chapters 8 through 12.
For a sample application written in Visual Basic, see the Vbphone sample application on the CTI OS CD.

Using the COM CIL in C++

Building a custom CTI application in Visual C++ with COM requires knowledge of creating and using COM components in Microsoft Visual C++. Client applications of this type tend to be more complex to build, but more powerful and faster in execution, than scripting clients (i.e. Visual Basic). All of the CIL components for COM are distributed as COM Dynamic Link Libraries (COM DLL).

In order to be accessible to COM containers, including Microsoft Visual C++, COM components must be registered with Windows. The components that you will require for programming in Microsoft Visual C++ are:

- **CTI OS Client library** (CTIOSClient.dll). This is the main CIL library for COM. The objects available in this library are described fully in Chapters 8 through 11.
- **CTI OS Arguments Library** (arguments.dll). The Arguments helper class is used extensively in CTI OS, and is described fully in Chapter 12, “Helper Classes.”
- **CTIOS Session Resolver** (ctiossessionresolver.dll). This object allows multiple applications or controls to use a single CTIOS Session object. It is required when building an application that will include the CTIOS ActiveX controls.

Adding COM Support to your Application

To make use of these objects in your CTI application, your application must support COM. To add COM support to your application, you must use one of the following:

- Microsoft Foundation Classes (MFC). The following header file are required for MFC applications to use COM: afxwin.h, afxext.h, afxdisp.h, and afxdctl.h. If you build an application using the Microsoft Visual C++ application wizard, these files are included for you automatically.
Microsoft’s ActiveX Template Library (ATL). To use ATL, include the standard COM header file: \textit{atbase.h}.

\textbf{Important Note About COM Method Syntax}

In this manual, the syntax used to describe method calls in COM shows standard COM data types such as BSTR, VARIANT and SAFEARRAY. Be aware that depending on the development environment, tools, and the manner how the COM CIL is included in your project application, these data types can be encapsulated by wrapper classes proper to the environment.

For example, in a Microsoft Visual C++ project a \texttt{VARIANT} type can be either a \texttt{CComVariant} or \_\texttt{variant} \_\texttt{t}, and a \texttt{BSTR} type can be either a \texttt{CComBSTR} or \_\texttt{bstr} \_\texttt{t}.

For more information, see the documentation for your development environment.

\textbf{Note}

The sample application for COM C++ is built using the MFC class support. Either MFC or COM is acceptable for using the CIL for COM in a C++ application.

\textbf{Using the CIL Dynamic Link Libraries}

Next, you must \textit{import} the COM Dynamic Link Libraries into your C++ application. The following code sample (which you might put into your \texttt{StdAfx.h} file) depicts how to use a COM Dynamic Link Library in C++:

\begin{verbatim}
#import "..\..\Distribution\COM\ctiosessionresolver.dll" using namespace CTIOSSESSIONRESOLVERLib;
#import "..\..\Distribution\COM\ctiosclient.dll" using namespace CTIOSCLIENTLib;
\end{verbatim}

\textbf{Note}

You must register three DLLs, but you do not need to import the \texttt{arguments.dll} into your project since it is imported by the \texttt{ctiosclient.dll} type library.
Creating an Instance of a COM Object

COM objects in C++ are created via the COM runtime library. To create a COM object at run time, your program will need to use the `CreateInstance()` method call.

```cpp
// Create SessionResolver and Session object
hRes = m_pSessionResolver.CreateInstance(OLESTR("CTI OSSessionResolver.SessionResolver"));

if (m_pSessionResolver)
{
    m_pSession = m_pSessionResolver->GetSession(_bstr_t(""));
}
```

Once the Session object is created, you can use it to make requests, and subscribe for events.

Subscribing and Unsubscribing to COM Events in C++

In this model, client applications subscribe for events by registering an instance of an event sink in the client with the event source. The COM Session object publishes several event interfaces (event sources), and clients can subscribe to any or all of them.

To receive COM events, you must first create an event sink class, which should derive from a COM event sink class. The Comphone sample application uses the MFC class `CCmdTarget`.

```cpp
class CEventSink : public CCmdTarget
{
    //...
};
```

This class must implement the method signatures for the events it expects to receive. When an event is fired from the event source, the corresponding method in your event sink class will be invoked, and you can perform your custom event handling code at that time.

To subscribe for an event, the client must call the `AtlAdvise()` method, specifying a pointer to the interface of the event source.

```cpp
// Add event sink as event listener for the _IallEvents interface
HRESULT hRes =
```
AtlAdvise( m_pSession, m_EventSink.GetIDispatch(FALSE),
__uuidof(_IAllEvents), &m_dwEventSinkAdvise);

When the program run is complete, the client must unsubscribe from the event source, using the AtlUnadvise() method:

// Unsubscribe from the Session object for the _IAllEvents interface

HRESULT hRes =
AtlUnadvise( m_pSession, __uuidof(_IAllEvents), m_dwEventSinkAdvise );

**Next Steps**

- For detailed information on the CTI OS client start up and shut down sequence, see the section “Disconnecting from CTI OS Server”.
- For detailed information on the CTI OS Client Interface Library objects, see Chapters 8 through 12.
- For a complete sample application that uses the CIL COM interface written in C++, see the Comphone sample application on the CTI OS CD.

**Using the C++ CIL and static libraries**

The CTI OS Client Interface Library for C++ is the most powerful, object-oriented CTI interface for C++ developers. It provides the same interface methods and events as the COM interface for C++, but will be more straightforward for C++ developers who are not experienced COM programmers, and will provide faster code execution.

The CIL interface for C++ is a set of C++ header files (.h), and static libraries compiled for the Win32 platform (Windows NT, Windows 2000). The header files required to access the class definitions are located on the CTI OS SDK media in the CTIOSToolkit\Include\ directory, and the static libraries are located in the CTIOSToolkit\Lib\ directory.

**Header Files and Libraries**

The header files you will most likely require are all included in the main CIL header file, CIL.h, which you would want to include in your application.
#include <Cil.h>

To link your application code with the CIL for C++, you will require the following C++ static libraries:

- **ConnectionLib.lib.** This library contains the connection-layer services for CIL.
- **ServiceLib.lib.** This library contains the service-layer services for CIL.
- **SessionLib.lib.** This library contains the object-interface services for CIL.
- **UtilLib.lib.** This library contains helper classes for CIL.
- **ArgumentsLib.lib.** This library contains the Arguments data structure for CIL.
- **SilentMonitorLib.** This library contains all the services required to establish and control silent monitor sessions.

**Note**

The preceding are the Release versions of the libraries. The Debug equivalent libraries use the same library name with the appended ‘d’; e.g., for ArgumentsLib, the Debug library is ArgumentsLibd.lib.

In addition to the aforementioned CTI OS CIL libraries, your application will require the standard Microsoft sockets library, Wsock32.lib, and the standard multimedia library winmm.lib.

### Project Settings for Compiling and Linking

Setting up your Visual C++ application requires you to configure some program settings. The Program Setting in Visual C++ are accessed under the Project / Setting menu:
Within the Project Settings dialog, select the C/C++ tab. Under Category, select Code Generation. For a Debug Mode program, the setting for “Use run-time library” should be “Debug Multithreaded DLL”. For a Release Mode program, the setting should be “Multithreaded DLL.”

Next, under the Category for “Proprocessor,” you will need to set the “Additional include directories.” You will need to provide the compiler with either the absolute or relative path to find the header files (.h) required for your application. This path should point to the CTIOSToolkit\Include directory, where the CIL header files are installed.
Next, you need to set the link settings for your project, under the Link tab. You must list all the static libraries for your program to link with under the section “Object/library modules.” The libraries required for CIL (in addition to the default libraries) for the debug settings are ConnectionLibd.lib, ServiceLibd.lib, SessionLibd.lib, ArgumentsLibd.lib, UtilLibd.lib, ws2_32.lib, and winmm.lib. For the Release version, the required libraries are ConnectionLib.lib, ServiceLib.lib, SessionLib.lib, ArgumentsLib.lib, UtilLib.lib, ws2_32.lib, and winmm.lib.
Finally, on the Link tab, set the Category to “Input.” You will need to set the “Additional library path:” to the location of the CTIOSToolkitLib directory:
The foregoing are all the Project Settings required for CTI OS. Click OK, and save your project settings.

**Subscribing for Events in C++**

Events interfaces are provided in C++ using the publisher-subscriber model. To subscribe for events, you must create a callback class (event sink), or implement the event interface in your main class. The callback class can be derived from the Adapter classes defined in CIL.h, such as AllInOneEventsAdapter.h.

To register for an event, you use the appropriate AddEventListener method on the Session object:

```cpp
// Initialize the event sink
m_pEventSink = new CEventSink(&m_ctiSession, &m_ctiAgent, this);

// Add event sink as an event listener
m_ctiSession.AddAllInOneEventListener((IAllInOne *) m_pEventSink);
```

To remove an event listener (upon program termination), use the appropriate RemoveEventListener on the Session object:

```cpp
// Tell session object to remove our event sink
m_ctiSession.RemoveSessionEventListener((IAllInOne *) m_pEventSink);
```

**STLport**

The Cisco CTI OS Toolkit uses an independent version of the C++ Standard Template Library (STL) known as *STLport*. A copy of *STLport* is provided with the Toolkit and is intended to be used by programmers using the toolkit in C++.

*STLport* is a multiplatform ANSI C++ Standard Library implementation. It is free, open-source product, featuring the following:

- Advanced techniques and optimizations for maximum efficiency
- Exception safety and thread safety
- Important extensions - hash tables, singly-linked list, rope

There are absolutely no restrictions on *STLport* use in commercial projects, and no royalties are involved. There is no licensing fee.
Why CTI OS Uses the STLport

The CTI OS development team moved to the STLport implementation primarily because of difficulties encountered when using the implementation provided by Microsoft with Visual Studio. Unfortunately, when using the Microsoft implementation, accessing STL objects created in a DLL may cause access violations or serious program errors including the appearance of data corruption. This is because the templates use static data members and each executable image (DLL or EXE) will contain its own copy of the static data members for a given class. The static data members between images are not kept in sync and can result in access violations or unstable behavior.

While it is possible to export STL objects from a DLL, it is not possible to export collection objects that have nested classes, such as maps. These types of collections are used extensively in the internal implementation of the CIL. The STLport implementation avoids this problem altogether.

For more information about Microsoft STL problems with DLLs, refer to MSDN Knowledge Base Article: Q172396 “PRB: Access Violation When Accessing STL Object in DLL”.

For more information about exporting STL components and limitations, refer to MSDN Knowledge Base Article: Q168958 “HOWTO: Exporting STL Components Inside & Outside of a Class”.

For more information about STLport, visit: http://www.stlport.org.

For more information about SGI STL visit: http://www.sgi.com/tech/stl.

Next Steps

- For detailed information on the CTI OS client start up and shut down sequence, see the section “Disconnecting from CTI OS Server”.
- For detailed information on the CTI OS Client Interface Library objects, see Chapters 6 through 11.
- For a complete sample application that uses the CIL interface with C++ static libraries, see the C++phone sample application on the CTI OS CD.
Using the Java CIL Libraries

The Java CIL provides a powerful cross-platform library for developing Java CTI applications. It is built using a similar architecture to the C++ CIL and the interface is also similar to C++ with some differences. As a result, a developer porting a C++ CIL application to Java or working between a Java and C++ should find it fairly easy to switch between the two.

The Java CIL consists of two packages contained in a single JAR file called JavaCIL.jar. The packages are com.cisco.cti.ctios.util and com.cisco.cti.ctios.cil. The Java CIL can be installed on Windows using the CTI OS Client Install or it can be copied directly from the CTIOS_JavaCIL directory on the CTIOS media under Installs\CTIOSClient. The Java CIL also includes JavaDoc with the distribution. No install is provided for Linux. Users will need to mount the CDROM and copy the CTIOS_JavaCIL directory from the media. The Java CIL version can be checked by using the CheckVersion.bat program in Windows or the checkversion shell script on Linux. Both of these can be found in the same directory as the JAR file.

Sun JRE installers are also included on the media as a convenience for developers who need to obtain the correct version of the JRE.

The Java CIL ships with a Java version of the CILTest tool as well as a GUI TestPhone application which provides most of the functionality found on the CTIOS Agent and Supervisor Desktops. The distribution also includes samples that are Java versions of some of the C++/COM/VB sample applications. See the section entitled “Using the Samples” for more information.

Next Steps

- Refer to Chapter 6 and Appendix A for differences between the C++ and Java event publishing.
- Refer to Chapters 7 through 12 for differences in method calls and syntax for those classes between C++ and Java.
- Refer to Appendix B for differences between C++ and Java tracing.
Connecting to the CTI OS Server

To connect a desktop application to the CTI OS server, you must:

1. Create a session instance, described below.
2. Set the event listener and subscribe to events, described below.
3. Set connection parameters, described below.
4. Call the Connect() method, described on page 4-29.
5. Set the connection mode, described on page 4-31.

This section also describes how to deal with connection failures, on page 4-29.

How to Create the Session Instance

To connect to the CTIOS Server, you must first create an instance of the CtiOsSession object.

The following line shows this in Java:

```java
CtiOsSession rSession = new CtiOsSession();
```

Session Object Lifetime (C++ only)

In C++, a Session object must be created on the heap memory store so that it can exist beyond the scope of the method creating it. (In COM, VB, and Java, this is handled automatically.)

For example:

```c++
CCtiOsSession * m_pSession = NULL;
m_pSession = new CCtiOsSession();
```

The client application should hold a reference to the Session object as long as it is in use, but the client programmer must release the last reference to the object to prevent a memory leak when the object is not longer needed.

During application cleanup, the Session object must only be disposed by invoking the CCtiOsSession::Release() method. This will ensure proper memory cleanup.

For example:

```c++
m_pSession->Release();
```
How to Set the Event Listener and Subscribe to Events

Before making any method calls with the Session instance, you must set the session as an event listener for the desktop application and subscribe to events. The following lines show this in Java:

```java
rSession.AddListener(this, CtiOs_Enums.SubscriberList.eAllInOneList);
```

In this example, the session is adding the containing class, the desktop application, as the listener, and using the eAllInOneList field in the CtiOs_Enums.SubscriberList class to subscribe to all events.

How to Set Connection Parameters for the Session

To set connection parameters:

1. **Step 1** Create an instance of the Arguments class.
2. **Step 2** Set values for the CTIOS servers, ports, and the heartbeat value.

**Note** When setting values, use the String key fields in the CtiOs_IKeywordIDs interface, as shown in the example below.

The following example demonstrates this task in Java:

```java
/* 1. Create Arguments object.*/
Arguments rArgs = new Arguments();

/* 2. Set Connection values.*/
rArgs.SetValue(CTIOS_CTIOSA, "CTIOServerA");
rArgs.SetValue(CTIOS_PORTA, 42408);
rArgs.SetValue(CTIOS_CTIOSB, "CTIOServerB");
rArgs.SetValue(CTIOS_PORTB, 42408);
rArgs.SetValue(CTIOS_HEARTBEAT, 100);
```

**Note** The Arguments.setValue() methods return a boolean value to indicate whether the method succeeded (true) or not (false).
How to Connect the Session to the CTI OS Server

After you successfully create the Session instance, you must connect it to the CTI OS Server using the Session.Connect() method, using the Arguments instance you constructed when setting connection parameters, as described in the previous section.

The following line shows this in Java:

```java
int returnCode = session.Connect(rArgs);
```

For more information on the possible values and meanings of the int value returned by the Connect() method in the Java CIL, see page 4-30.

When successful, the Connect() method generates the OnConnection() event. Code within the OnConnection() event should set the connection mode, as described in the next section.

Dealing with Connection Failures

This section contains the following information:

- Connection Failure Events, page 4-29
- Connection Attempt Error Codes in Java CIL, page 4-30
- Configuring the Agent to Automatically Log In after Failover, page 4-31
- Stopping the Failover Procedure, page 4-31

Also see How to Deal with Failover In Monitor Mode, page 4-37.

Connection Failure Events

If the Connect() method does not succeed, one of the following events is generated:

- OnConnectionRejected() event indicates that an unsupported version mismatch has been found.
OnCTIOSFailure() indicates that the CTI OS Server requested in the Connect() method is down. If an OnConnectionFailure() event is generated, the application is in Failover and the CIL continues to attempt to connect until the connection succeeds or until the application calls Disconnect(). The Arguments parameter for the event includes the following keywords:

- FailureCode
- SystemEventID
- SystemEventArgs
- ErrorMessage

For more information on the contents of the OnConnectionFailure() event, see the description in Chapter 6.

**Connection Attempt Error Codes in Java CIL**

The following field values may be returned by the Connect() method. See the documentation for the CtiOsEnums.CilError interface in the Java CIL JavaDoc for information on these fields.

- **CIL_OK** - There are no obvious connection errors. The application should wait for an event indicating whether or not the Connect() method succeeded.

- **CIL_FAIL** - The initial attempt to connect to host has failed. The Java CIL will fire the OnConnectionFailure() event and go into failover mode. The Java CIL will continue to attempt to connect, alternating between hosts CTIOS_CTIOSA and CTIOS_CTIOSB, until the connection succeeds, at which point the Java CIL will fire the OnConnection() event.

- **E_CTIOS_INVALID_ARGUMENT** - A null Arguments parameter was passed to the Connect() method. The connection failed. No events are fired.

- **E_CTIOS_MISSING_ARGUMENT** - The Arguments parameter did not contain values for both CTIOS_CTIOSA and CTIOS_CTIOSB. At least one of these values must be provided. The connection failed. No events are fired.

- **E_CTIOS_IN_FAILOVER** - A previous connection attempt failed and the Java CIL is currently in failover and attempting to establish a connection. This continues until a connection is established, at which point the Java CIL will fire an OnConnection() event indicating that the previous Connect() method has succeeded. To attempt to connect again with different parameters, the application must first use the Disconnect() method.
Connecting to the CTI OS Server

- **E_CTIOS_MODE_CONFLICT** - The Session is not disconnected (i.e. a previous Connect() method is in progress, or the Session is already connected). The application must call the Disconnect() method before attempting to establish another connection. The Java CIL may fire an OnConnection() event for the to previous call to the Connect() method if the connection was in progress, but will not fire one corresponding to this method call.

- **E_CTIOS_SESSION_NOT_CONNECTED** - There was an unanticipated error. The connection failed. No events are fired.

### Configuring the Agent to Automatically Log In after Failover

If you are using CTI OS in an IPCC Enterprise environment, you can configure the agent to automatically log in again in the event of a failover.

To configure the agent to log back in automatically, add the CTIOS_AUTOLOGIN keyword with the value “1” to the Arguments instance used to configure the agent:

```java
rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AUTOLOGIN, "1");
```

For more information on logging in an agent, see page 4-38.

### Stopping the Failover Procedure

To stop the failover procedure, call the Disconnect(args) method, with the Arguments instance containing the CTIOS_FORCEDDISCONNECT keyword as a parameter.

### How to Set the Connection Mode

After the session is created, you must specify the connection mode for the session. You must use one of two modes:

- Agent mode
- Monitor mode
Chapter 4  Building Your Application

Connecting to the CTI OS Server

Setting the Connection Mode in the OnConnection() Event Handler

To ensure that you only try to set the connection mode on valid connections, you should place the code to set the connection mode within the OnConnection() event handler. The OnConnection() event is generated by a successful Connect() method.

⚠️ Caution

The application should contain logic within the OnConnection() event handler to ensure it attempts to set the connection mode only during the initial connection, and not in an OnConnection() event due to failover.

When to Use Agent Mode

You use Agent mode for connections when the client application must log in and control a specific agent. When in Agent mode, the connection also receives call events for calls on that agent's instrument, as well as system events.

How to Select Agent Mode

To select Agent mode for the connection, in the OnConnection() event:

1. Set properties for the agent.

    📝 Note
    The properties required for the agent depend on the type of ACD you are using. The following example demonstrates the required properties for IPCC users.

2. Set the agent for the Session object to that Agent object.

    📝 Note
    In the **Java CIL only**: If the SetAgent() method is called on a session in which the current agent is different than the agent parameter in the SetAgent() method, the Java CIL automatically calls the Disconnect() method on the current session instance, generating an OnCloseConnection() event, then attempts to reconnect, generating an OnConnection() event. Then the new agent is set as the current agent.
The following example, which assumes the Session object has been created and connected to the CTI OS Server, demonstrates this task in Java:

```java
void OnConnection(Arguments rArgs) {
  /* 1. Create and agent and set the required properties. */
  Agent agent = new Agent();
  agent.SetValue(CtiOs_IKeywordIDs.CTIOS_AGENTID, "275");
  agent.SetValue(CtiOs_IKeywordIDs.CTIOS_PERIPHERALID, "5002");

  /* 2. Set the session's agent */
  int returnValue = session.SetAgent(agent);
}
```

When successful, the SetAgent() method generates the following events:

- OnQueryAgentStateConf()
- OnSetAgentModeConf()
- OnSnapshotDeviceConf(), if the agent is already logged in
- OnSnapshotCallConf(), if there is a call and the agent is already logged in
- OnCTIOSFailureEvent()

**When to Use Monitor Mode**

You use Monitor mode to receive all events that CTI OS Server publishes, or a specified subset of those events. You may receive events for calls, multiple agents, or statistics. The session receives specific events based on the event filter used when setting the session to Monitor mode.

**Caution**

When a Monitor mode session is initialized, the CTI OS Server performs a CPU intensive sequence of operations to provide the application with a snapshot of the state of the system. A large number of monitor-mode applications connecting to CTIOS server at the same time, such as in a fail-over scenario, may cause significant performance degradation on CTI OS Server. You should therefore minimize the number of Monitor mode applications connecting to CTI OS Server.
Monitor Mode Filters

Overview

To set a connection to Monitor mode, you must create a filter that specifies which events to monitor over that connection. The filter is a String; that String is the value for the CtiOs_IKeywordIDs.CTIOS_FILTER key in an Arguments instance. That Arguments instance is the argument for the SetMessageFilter() method.

Filter String Syntax

The filter String you create to specify events to monitor must adhere to a specific syntax to accurately instruct the CTI OS Server to send the correct events.

The general syntax for the filter String is as follows:

"key1=value1, value2, value3;key2=value4, value5, value6"

Note

The filter String may also contain an asterisk (*), which is used as a wildcard to indicate any possible value.

The filter String must contain at least one key, and there must be at least one value for that key. However, a key may take multiple values, and the filter String may contain multiple keys.

Multiple values for a single key must be separated by commas (,). Multiple keys must be separated by semicolons (;).

Note

Multiple keys in a single filter combine using a logical AND. That is, the filter is instructing CTI OS to send to this connection only events that meet all the criteria included in the filter.

For example, a filter String could be as follows:

S_MESSAGEID + "=" + CtiOsEnums.EventID.eAgentStateEvent + ";" + S_AGENTID + "=5128";
Chapter 4  Building Your Application  Connecting to the CTI OS Server

This example works as follows:

- The first key-value pair, `S_MESSAGEID + "=" + CtiOs_Enums.EventID.eAgentStateEvent`, serves to request events with a message ID equal to `eAgentStateEvent`; that is, it requests agent state events.
- The second key-value pair, `S_AGENTID + "=5128"`, specifies that the request is for the agent with the ID 5128.
- The result of the filter then is that the connection will receive agent state events for agent 5128.

**Filter Keys**

Filter keys can be any known key value used by CTI OS. These keys have corresponding fields in the CtiOs_IKeywords interface.

**Note**

When constructing the filter String, use the fields that begin with “S_”, as these are the String values for the key.

For example, in Java:

```java
String sFilter = S_AGENTID + "=5128,5129,5130";
```

In this example, `S_AGENTID` is the String representation of the key indicating an Agent ID.

**Filtering for Events for Monitored Calls or Monitored Agents**

If a client filter mode application wants to filter for events for monitored calls, the application should do the following:

- Create the filter
- Check events to verify that the CTIOS_MONITORED parameter is present and is TRUE
- Ignore events if the CTIOS_MONITORED parameter is missing or FALSE
How to Select Monitor Mode

To select Monitor mode for the connection:

---

**Step 1** Specify the filter String. See the previous section for filter details.

**Step 2** Create an Arguments instance and add an item with CtiOs_IKeywordIDs.CTIOS_FILTER as the keyword and the filter String as the value.

**Step 3** Use the CtiOsSession.SetMessageFilterArgs(args) method to select Monitor mode and to set the event filter.

---

**Note** You should always include the OnCtiOsFailure() event in the message filter, so that the application can detect when a system component is comes on- or offline.

---

**Caution** A Monitor Mode application that monitors any Call-related events must also monitor the OnCallEnd() event, as described on page 4-51.

The following example, which assumes the Session object has been created, demonstrates this task in Java:

```java
/* 1. Constructing message filter string */
String filter = "messageid=" + eAgentStateEvent + "," + eAgentInfoEvent + "," + eCTIOSFailureEvent;

/* 2. Create the Arguments object*/
Arguments rArgs = new Arguments();

/* 3. Add the filter to the Arguments instance.*/
rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_FILTER, filter);

/* 3. Set the message filter.*/
int returnValue = session.SetMessageFilter(rArgs);
```

When successful, the SetMessageFilter() method generates the following events:

- With IPCC only, OnQueryAgentStateConf() for each team and each agent logged in
Connecting to the CTI OS Server

- OnSnapshotDeviceConf() for each device
- OnSnapshotCallConf()
- OnMonitorModeEstablished()

How to Deal with Failover In Monitor Mode

The CTI OS CIL does not support failover for Monitor mode. Agents in monitor mode cannot recover their state after a failover. Furthermore, after a failover, the CTI OS CIL may leak Call objects.

To deal with failover in Monitor mode:

---

**Step 1** When the application detects a failover, for example, in a CTIOSFailure() event indicating a connection failure or an offline component, wait until the CIL has failed over and everything is back online and the CIL is connected to CTI OS.

The Monitor mode application is responsible for determining when all required servers are online. You can do this by monitoring OnCtiosFailure() events and keeping track of system status changes as they occur.

**Step 2** Use the Disconnect() method to disconnect the session from CTI OS.

**Step 3** Follow the steps starting at the beginning of the section “Enabling Silent Monitor in your Application” to:

a. Create a session instance.

b. Set the event listener.

c. Set connection parameters.

d. Call the Connect() method.

e. Set the connection mode in the OnConnection() event handler.
Logging In and Logging Out an Agent

How to Log In an Agent

When the connection to the CTIOS Server is established and the mode set, you log in the agent.

Note
Before attempting to log in an agent, you would typically request global configuration data, in order to correctly handle a duplicate log in attempt. For more information, see the section “How to Get Registry Configuration Values to Your Desktop Application.”

To log in the agent, in the SetAgentModeEvent() event:

Step 1
Create an instance of the Arguments class.

Step 2
Set log in values for the agent in the Arguments instance.

Note
The properties required for the agent depend on the type of ACD you are using. The following example demonstrates the required properties for IPCC.

Step 3
Log in the agent.

The following example, which assumes the Agent object has been created, demonstrates this task in Java:

```java
public void SetAgentMode(Arguments rArgs) {
    /* 1. Create Arguments object*/
    Arguments rArgs = new Arguments();

    /* 2. Set log in values.*/
    rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AGENTID, "275");
    rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_PERIPHERALID, "5002");
    rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AGENTINSTRUMENT, "5002");
    rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AGENTPASSWORD, "********");
```
rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AUTOLOGIN, "1");

/* 3. Log in the agent. */
int returnValue = agent>Login(rArgs);
}

**Note**  
It is the client application’s responsibility to keep track of whether the log in attempt is the first attempt, or during failover, and branch accordingly in the SetAgentMode() event to avoid calling the Login() method during failover.

The Login() method generates the following events:

- QueryAgentStateConf()
- AgentStateEvent(), if the agent is unknown or is logged out.

**Note**  
The client application receiving these events must check both the ENABLE_LOGOUT and ENABLE_LOGOUT_WITH_REASON bitmasks. For more information, see What to do in the OnButtonEnablementChange() Event, page 4-47.

When not successful, the Login() method generates the eControlFailureConf() event.

### How to Handle Duplicate Log In Attempts

#### Overview

A duplicate log in attempt is when an agent who is already logged in tries to log in a second time using the same ID. Desktop applications must account for such a possible situation and have a plan for dealing with it.

You can handle duplicate log in attempts in three ways:

- Allow the Duplicate Log In with No Warning
- Allow the Duplicate Log In with a Warning
- Do not allow a duplicate log in.

You control how duplicate log in attempts are handled in two ways:
By configuring how duplicate log in attempts are to be handled on a global basis by creating custom values in the CTI OS Server Registry. By using custom values in the CTI OS Server registry to control how duplicate log in attempts are handled, and downloading these settings to your desktop application as described on page 4-45, you can enable flexibility without having to modify your desktop application code.

By implementing code in your desktop application to detect then to handle the duplicate log in attempt error according to the custom values in the CTI OS Server Registry. You can write code to handle duplicate log in attempts in each of the three ways listed above. Then when you must change how such attempts are handled, you would simply change the registry settings; you would then not need to change the desktop application code.

How to Create Values in the CTI OS Server Registry to Control Duplicate Log In Attempts

You can create keys in the CTI OS Server Registry that will instruct desktop applications to handle duplicate log in attempts in a specific way.

**Warning**

Two keys exist by default in the registry that are used by the CTI OS CIL: WarnIfAlreadyLoggedIn and RejectIfAlreadyLoggedIn. You must not use these keys in your desktop application. You must instead create other keys as described in this section.

You should create two custom values:

- custom_WarnIfAgentLoggedIn
- custom_RejectIfAgentLoggedIn

The custom keys you create can be set to 0 (False) or 1 (True).

The following table lists the settings to use to control how duplicate log in attempts are to be handled:
Goal | custom_WarnIfAgentLoggedIn | custom_RejectIfAgentLoggedIn
---|---|---
To warn the agent of the duplicate log in attempt, but to allow the agent to proceed | 1 | 0
To allow the agent to proceed with the duplicate log in attempt with no warning | 0 | 0
To not allow the agent to proceed with a duplicate log in attempt | 0 or 1 | 1

To create keys to control duplicate log in attempts:

**Step 1** Open the registry and navigate to: `HKEY_LOCAL_MACHINE\Software\Cisco Systems, Inc\CTIOS\CTI Instance Name\CTIOS1\EnterpriseDesktopSettings\AllDesktops\Login\ConnectionProfiles\Name\Profile Name`.

**Step 2** Right click in the registry window and select New->DWord Value. The new value appears in the window.

**Step 3** Change the value name to `custom_WarnIfAgentLoggedIn`.

**Step 4** Double-click the value to open the Edit DWORD Value dialog box.

**Step 5** Enter 1 in the Value data field to set the value to true, or 0 to set it to false.

**Step 6** Repeat steps 2 through 5 for the value `custom_RejectIfAgentLoggedIn`.

### How to Get Registry Configuration Values to Your Desktop Application

To get CTI OS registry configuration values to your desktop application, in order to handle duplicate log in attempts correctly, you must request global configuration settings, then extract the custom settings from the event. You would typically do this task before attempting to log in an agent, in the OnConnection() event.
Step 1 Create an instance of the Arguments class.

Step 2 In the Arguments instance, set the value for the CTIOS_DESKTOPTYPE key to either:

- CtiOsEnums.DesktopType.eAgentDesktop
- CtiOsEnums.DesktopType.eSupervisorDesktop

**Note** Although the Arguments object must have one of these fields as a value for the CTIOS_DESKTOPTYPE key, this version of CTI OS does not utilize the desktop type parameter when sending global configuration data to a desktop application. Regardless of which field you use in defining the Arguments object, CTI OS returns all global configuration data with the OnGlobalSettingsDownloadConf() event. The desktop type indicators, through currently required, are reserved for future use.

Step 3 Request desktop settings for the session using the RequestDesktopSettings() method. This results in a OnGlobalSettingsDownloadConf() event.

The following example demonstrates steps 1 through 3 in Java:

```java
/* 1. Create Arguments object*/
Arguments rArgs = new Arguments();

/* 2. Set the desktop type.*/
rArgs.SetValue("CTIOS_DESKTOPTYPE",
CtiOsEnums.DesktopType.eAgentDesktop);

/* 3. Request desktop settings. This should cause CTI OS to send the
OnGlobalSettingsDownloadConf event.*/
int returnValue = session.RequestDesktopSettings(rArgs);
```

Step 4 Then, in the OnGlobalSettingsDownloadConf() event, get the Arguments instance for Login configuration from the event Arguments parameter. Use the S_LOGIN key from the CtiOs_IKeywordIDs interface.

Step 5 Get the Arguments instance for the correct switch from the Login Arguments instance. The example below uses the “IPCC/SoftACD” login configuration information, the key for which is established by the CTI OS Server installation.

Step 6 Get the Integer instances for the custom values you established for the key in the CTI OS Server registry.
Step 7  For convenience, get the int values for those Integers to test with, as described in the section How to Handle Duplicate Log In Attempts, page 4.39.

The following example demonstrates steps 4 through 7 in Java:

```java
void OnGlobalSettingsDownloadConf(Arguments rArgs) {
    /* 4. Get the Arguments instance for the Login configuration information from the event Arguments parameter. */
    Arguments logInArgs = rArgs.getValueArray(CTIOS_LOGIN);
    /* 5. Get the Arguments instance for the Connection Profile from the Login Arguments instance. */
    Arguments connectionProfilesArgs = logInArgs.GetValueArray(CTIOS_CONNECTIONPROFILES);
    /* 6. Get the Arguments instance for the specific switch from the Connection Profiles instance */
    Arguments IPCCLogInArgs = connectionProfilesArgs.GetValueArray("IPCC/SoftACD");
    /* 7. Get the Integer instances for the custom values you entered in the CTI OS Server registry. */
    Integer warningIntObj = IPCCLogInArgs.GetValueIntObj("custom_WarnIfAgentLoggedIn");
    Integer rejectIntObj = IPCCLogInArgs.GetValueIntObj("custom_RejectIfAgentLoggedIn");
    /* 8. Get the int values for those object to test later. */
    custom_WarnIfAgentLoggedIn = warningIntObj.intValue();
    custom_RejectIfAgentLoggedIn = rejectIntObj.intValue();
}
```

How to Detect the Duplicate Log In Attempt in the Desktop Application

You detect the duplicate log in attempt in the OnQueryAgentStateConf() event, which is sent after the application calls SetAgent():
Step 1
Get the agent state value from the Arguments instance passed to the event.

Step 2
Test the agent state value in the CtiOs_Enums.AgentState interface, as follows.

\[(\text{state} \neq \text{eLogout}) \land (\text{state} \neq \text{eUnknown})\]

Step 3
If the test is true, handle the duplicate log in attempt as described in the next section.

The following example demonstrates this task in Java:

```java
public void eQueryAgentStateConf(Arguments rArgs) {
    /* 1. Get the agent state value*/
    Short agentState = rArgs.getValueShortObj(CTIOS_AGENTSTATE);

    /*Test the agent state*/
    if (agentState.intValue() != eLogout
        && agentState.intValue() != eUnknown) {
        /*If the agent is logged in, handle duplicate log in attempt.*/
    }
}
```

How to Handle Duplicate Log In Attempts in the Desktop Application

If you detect from the OnQueryAgentStateConf() event that the agent is already logged in as described in the previous section, do the following:

- If your customer_WarnIfAgentLoggedIn = 1 and custom_RejectIfAgentLoggedIn = 0, notify the user that the agent is already logged in and proceed with Login() depending on the user’s response.
- If your custom_RejectIfAgentLoggedIn = 1, notify the user that the agent is already logged in and Disconnect.

How to Log Out an Agent

To log out an agent:

Step 1
Create an instance of the Arguments class.

Step 2
Set log out values for the agent in the Arguments instance.
Note  
IPCC requires a reason code to log out. Other switches may have different requirements.

Step 3  
Log out the agent.

The following example demonstrates this task in Java:

```java
/* 1. Create Arguments object*/
Arguments rArgs = new Arguments();

/* 2. Set log out values.*/
rArgs.SetValue(CTIOS_EVENTREASONCODE, 1);

/* 3. Log out the agent.*/
int returnValue = agent.Logout(rArgs);
```

Working with Calls

Handling Multiple Calls

It is critical that you design an Agent Mode desktop application to be able to store all the calls on the specific device, in order to do the following:

- Apply incoming events to the correct call
- Select the correct call on which to make method calls (i.e. telephony requests)

It is not necessary to maintain a set of Call Objects in order to do this. Instead, the application could store the string UniqueObjectID of each call (keyword CTIOS_UNIQUEOBJECTID). CTIOS_UNIQUEOBJECTID is always included in the args parameter for each call event. The actual Call Object can be obtained with the Session object's GetObjectFromObjectID() method in order to make a method call.
What is the Current Call?

The CIL maintains a concept of a *Current Call*, which is the call for which the last OnButtonEnablementChange() event was fired. Knowing which call is the Current Call is useful when there are multiple components which set and act on the Current Call, such as telephony ActiveX Controls.

The CTI OS ActiveX controls included in the CTI OS Toolkit use the concept of the Current Call. The CallAppearance grid control sets the Current Call when the user clicks on a particular call in the grid. Then when the user clicks the Answer control, this control must get the Current Call in order to call the Answer() method on the correct call.

The Current Call is set according to the following rules:

- When there is only 1 call on a device, the CIL sets it to the Current Call.
- When there are multiple calls on a device and an application wants to act on a call that is not the Current Call, it sets a different call to the Current Call with the SetCurrentCall() method.
- When the call which is the Current Call ends, leaving multiple calls on the device, the application must set another call to be the Current Call.
- Whenever the Current Call is set to a different call, OnCurrentCallChanged() event is fired as well as an OnButtonEnablementChange() event.

How to Get a Call Object

You can get the Call object from the session using the GetObjectFromObjectID() method.

The following code fragment, which assumes that existing Call Unique Identifiers are stored in an array called UIDArray, shows how to get a specific Call object in Java:

```java
String sThisUID = UIDArray[Index];
Call ThisCall = (Call) m_Session.GetObjectFromObjectID(sThisUID);
```
How to Set the Current Call for the Session

To set the current call you use the SetCurrentCall() method for the Session. The following code fragment, which assumes you retrieved the Call object as described in the previous section, shows how to set the current call.

The following line shows this in Java:

```java
m_Session.SetCurrentCall(ThisCall);
```

When Does the Application Receive the OnButtonEnablementChange() Event?

An application receives an OnButtonEnablementChange() event in the following situations:

- When the Current Call is changed.
- When the call which is the Current Call receives an event which includes a CTIOS_ENABLEMENTMASK argument. Usually the included enablement mask is changed from what it was set to, but occasionally it is the same. This mask is used to indicate which functions are allowed for this Call in its current state.

For example, when a Call receives an OnCallDelivered() event with a Connection State of LCS_ALERTING, its enablement mask is changed to set the Answer bit. When this Call is answered, and it receives the OnCallEstablished() event, the mask no longer sets the Answer bit, but instead enables the Hold, Release, TransferInit and ConferenceInit bits.

What to do in the OnButtonEnablementChange() Event

To see if a button should be enabled, simply do a bitwise "AND" with the appropriate value listed in the Table included under the OnButtonEnablementChange event in Chapter 6.
The following examples shows this in Java:

```java
Integer IMask = rArgs.GetValueIntObj(CTIOS_ENABLEMENTMASK);
if (null != IMask) {
    int iMask = IMask.intValue();
    if (iMask & ENABLE_ANSWER) {
        //Enable the AnswerCall button
    }
    else {
        //Disable the AnswerCall button
    }
} // else do nothing
```

### Checking Not Ready Bitmasks in OnButtonEnablementChange() Event

A client application receiving the OnButtonEnablementChange() event must check both the ENABLE_NOTREADY and ENABLE_NOTREADY_WITH_REASON bitmasks in the event.

---

**Caution**

Failure to check both the ENABLE_NOTREADY and ENABLE_NOTREADY_WITH_REASON bitmasks may lead to problems properly displaying a NotReady control to the agent.
The following example shows this in Java:

```java
void OnButtonEnablementChange(Arguments rArguments) {
    m_appFrame.LogEvent("OnButtonEnablementChange", rArguments);

    // Get mask from message
    Long LMask = rArguments.GetValueUIntObj(CTIOS_ENABLEMENTMASK);
    if (null==LMask)
        return;

    final long bitMask = LMask.longValue();

    /* Transfer modification of the GUI objects to the
    EventDispatchThread or we could have a thread sync issue. We're
    currently on the CtiOsSession's event thread. */
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            /* Enable a button if it's bit is
            turned on. Disable it if not. */

            m_appFrame.m_btnAnswer.setEnabled (((bitMask & ENABLE_ANSWER) > 0));
            m_appFrame.m_btnConference.setEnabled (((bitMask & ENABLE_CONFERENCE_COMPLETE) > 0));
            m_appFrame.m_btnCCConference.setEnabled (((bitMask & ENABLE_CONFERENCE_INIT) > 0));
            m_appFrame.m_btnHold.setEnabled (((bitMask & ENABLE_HOLD) > 0));
            m_appFrame.m_btnLogin.setEnabled (((bitMask & ENABLE_LOGIN)> 0));
            m_appFrame.m_btnLogout.setEnabled (((bitMask & (ENABLE_LOGOUT |
                CtiOs_Enums.ButtonEnablement.ENABLE_LOGOUT_WITH_REASON)) > 0));
            m_appFrame.m_btnMakeCall.setEnabled (((bitMask & ENABLE_MAKECALL) > 0));
            m_appFrame.m_btnNotReady.setEnabled(((bitMask & (ENABLE_NOTREADY |
                ENABLE_NOTREADY_WITH_REASON)) > 0));
            m_appFrame.m_btnRelease.setEnabled(((bitMask & ENABLE_RELEASE)> 0));
            m_appFrame.m_btnRetrieve.setEnabled (((bitMask & ENABLE_RETRIEVE) > 0));
            m_appFrame.m_btnSSTransfer.setEnabled (((bitMask & ENABLE_SINGLE_STEP_TRANSFER)> 0));
            m_appFrame.m_btnSSConference.setEnabled (((bitMask & ENABLE_SINGLE_STEP_CONFERENCE) > 0));
            m_appFrame.m_btnTransfer.setEnabled (((bitMask & ENABLE_TRANSFER_COMPLETE)> 0));
            m_appFrame.m_btnCCTransfer.setEnabled (((bitMask & ENABLE_TRANSFER_INIT) > 0));
```
OnButtonEnablementChange() Event in Supervisor Desktop Applications

When a supervisor desktop application processes an OnButtonEnablementChange() event, the application should check for the CTIOS_MONITORED parameter and ignore this parameter if it is present and is TRUE. In a supervisor desktop application, the OnButtonEnablementChange() event can reflect button enablement for either a monitored team member or the supervisor.

Coding Considerations for CIL Event Handling

The CTI OS CIL fires events to the application in a single thread. It is recommended that the amount of time spent in a particular CIL event handler be kept to a minimum so as to ensure timely delivery of subsequent CIL events. If a screenpop based on a call event (e.g., CallDelivered, CallDataUpdate event) takes longer than a few seconds (e.g., remote database lookup), it is recommended to delegate this operation to a separate thread or separate process so as not to block CTIOS event handling.
If an application calls a COM CIL API method from a COM CIL event callback routine it must ensure that the method call is made on the same thread as the CIL event callback. This rule applies to the following methods:

- `CComSession::SetCurrentCall`
- `CComSession::SetAgent`

This rule must be followed in order to guarantee that events are fired from the COM CIL to the application in the proper sequence.

When handling events in the browser using JavaScript, event processing time should be kept to a minimum since all other JavaScript execution (e.g., handling of button clicks) may be blocked during handling of the event.

### Monitoring the OnCallEnd() Event

A Monitor Mode application that monitors any Call-related events must also monitor the OnCallEnd() event.

**Warning**

The Call object in the CTI OS CIL is only deleted when the OnCallEnd() event is received; therefore if the OnCallEnd() event is not monitored, Call objects will accumulate and cause a memory leak.

### Working with Agent Statistics

#### Overview

This section describes how to work with agent statistics and contains the following subsections:

- “How to Set Up an Agent Application to Receive Agent Statistics” on page 4-52
- “How to Set Up a Monitor-Mode Application to Receive Agent Statistics” on page 4-53
- “Accessing Agent Statistics” on page 4-57
How to Set Up an Agent Application to Receive Agent Statistics

To set up an Agent application to receive agent statistics:

Step 1 Create an instance of the Session class, as described on page 4-27.
Step 2 Subscribe for events for the session, as described on page 4-28.

Note You must register to receive agent and session events; therefore, in the AddEventListener() method you must use as parameters the field CtiOsEnums.SubscriberList.eAgentList and CtiOsEnums.SubscriberList.eSessionList. Or you can use the CtiOsEnums.SubscriberList.eAllInOneList.

Step 3 Set connection parameters, as described on page 4-28.
Step 4 Connect the desktop application to the CTI OS Server, as described on page 4-29.
Step 5 In the OnConnection() event handler, set the Agent for the session, as described on page 4-32.
Step 6 Log in the agent, as described on page 4-38.
Step 7 Enable agents statistics using the EnableAgentStatistics() method.

Note Although the EnableAgentStatistics() method requires an Arguments parameter, there are no parameters to set for agent statistics; you can therefore send an empty Arguments instance as a parameter.

Caution The agent must be logged in before you can use the EnableAgentStatistics() method.

Step 8 To disable agents statistics, use the DisableAgentStatistics() method.
The following example demonstrates this task in Java:

```java
/* 1. Create session.*/
CtiOsSession rSession = new CtiOsSession();

/* 2. Add event listener.*/
rSession.AddEventListener(this,
   CtiOsEnums.SubscriberList.eAgentList);

/* 3. Set Connection values.*/
Arguments rArgs = new Arguments();
rArgs.SetValue(CTIOS_CTIOSA, "CTIOSServerA");
rArgs.SetValue(CTIOS_PORTA, 42408);
rArgs.SetValue(CTIOS_CTIOSB, "CTIOSServerB");
rArgs.SetValue(CTIOS_PORTB, 42408);
rArgs.SetValue(CTIOS_HEARTBEAT, 100);

/*4. Connect to server.*/
returnCode = rSession.Connect(rArgs);

public void OnConnection(Arguments rArgs) {

   /*5. Set agent for the session. */
   returnCode = rSession.SetAgent(agent);

   /* 6. Log in the agent.*/
   Arguments rArgs = new Arguments();
rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AGENTID, "275");
rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_PERIPHERALID, "5002");
rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AGENTINSTRUMENT, "5002");
rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AGENTPASSWORD, "********");
   returnCode = agent.Login(rArgs);

   /* 7. Enable Agent statistics. */
   if (returnCode == CIL_OK) {
      agent.EnableAgentStatistics(new Arguments());
   }
}
```

How to Set Up a Monitor-Mode Application to Receive Agent Statistics

To set up a Monitor-mode application to receive agent statistics, follow the instructions below.
Note: The agent to monitor must be logged in Agent mode before a Monitor-mode application can receive statistics for that agent.

**Step 1** Create an instance of the Session class, as described on page 4-27.

**Step 2** Subscribe for events for the session, as described on page 4-28.

Note: You must register to receive agent events; therefore, in the AddEventListener() method you must use as a parameter the field CtiOsEnums.SubscriberList.eAgentList.

**Step 3** Set connection parameters, as described on page 4-28.

**Step 4** Connect the desktop application to the CTI OS Server, as described on page 4-29.

**Step 5** Set a String variable to store the ID of the agent for which you want statistics.

Note: The application must be aware of the Agent ID and the agent’s Peripheral ID for any agent to monitor; the application cannot dynamically get these values from CTI OS Server.

**Step 6** Set the message filter as described on page 4-35.

a. Create String for the filter using the keyword CTIOS_MESSAGEID as the name, and "*:agentID" as the value.

Note: The "*:" indicates all events for that agent.

b. Create an instance of the Arguments class.

c. Set the value in the filter for the CTIOS_FILTER keyword to the String created in Step a.

d. Use the SetMessageFilter() method in the Session class to set the filter for the session, using the Arguments instance you created in Step b as a parameter.

**Step 7** Wait for any event for the agent, to ensure that the Agent instance exists for the Session.
Chapter 4  Building Your Application

Working with Agent Statistics

Caution  The application must wait for the first event for this agent before continuing, to ensure that the Agent instance is part of the current session.

Note  This example uses a Wait object to wait.

Step 8  Get the Agent instance from the Session using GetObjectFromObjectID() method.

Step 9  Enable agents statistics using the EnableAgentStatistics() method.

Note  Although the EnableAgentStatistics() method requires an Arguments parameter, there are no parameters to set for agent statistics; you can therefore you an empty Arguments instance as a parameter.

Caution  The agent must be logged in before you can use the EnableAgentStatistics() method.

Step 10  To disable agents statistics, use the DisableAgentStatistics() method.
The following example demonstrates this task in Java:

```java
/* 1. Create session.*/
CtiOsSession rSession = new CtiOsSession();

/* 2. Add event listener.*/
rSession.AddEventListener(this,
    CtiOs_Enums.SubscriberList.eAgentList);

/* 3. Set Connection values.*/
Arguments rArgs = new Arguments();
rArgs.SetValue(CTIOS_CTIOSA, "CTIOSServerA");
rArgs.SetValue(CTIOS_PORTA, 42408);
rArgs.SetValue(CTIOS_CTIOSB, "CTIOSServerB");
rArgs.SetValue(CTIOS_PORTB, 42408);
rArgs.SetValue(CTIOS_HEARTBEAT, 100);

/* 4. Connect to server.*/
int returnCode = rSession.Connect(rArgs);

/* 5. Set String to AgentID*/
String UIDAgent = "agent.5000.5013";

/* 6. Set the message filter. */
String filter = "MessageId=*;AgentId=5013;"
    rArgs = new Arguments();
rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_FILTER, filter);
returnCode = rSession.SetMessageFilter(rArgs);

/* 7. Wait for agent events.*/
rArgs = new Arguments();

    // Create a wait object in the session
    WaitObject rWaitObj = rSession.CreateWaitObject(rArgs);

    // Load the events into the Args for the wait object
    rArgs.SetValue("Event1", eAgentStateEvent);
    rArgs.SetValue("Event2", eQueryAgentStateConf);
    rArgs.SetValue("Event3", eControlFailureConf);
    rArgs.SetValue("Event4", eCTIOSFailureEvent);

    // Set the mask for the WaitObject
    rWaitObj.SetMask(rArgs);

    // Wait for up to 9 seconds, and then give up
    if (rWaitObj.WaitOnMultipleEvents(9000) != EVENT_SIGNALED)
    {
        // Handle error ...
    }
```
return;
}

// Find out what triggered the wait
int iEventID = rWaitObj.GetTriggerEvent();
if (iEventID == eControlFailureConf|| iEventID == eCTIOSFailureEvent)
{
    // Handle error ...
    return;
}

## Accessing Agent Statistics

### Overview

Once the applications are set up to receive agent statistics, as described in the preceding section, you can access agent statistics in two ways:

- By implementing the eOnNewAgentStatisticsEvent() (in Java) or the OnAgentStatistics() event (in C++, COM, or VB)

⚠️ **Caution**

Note carefully that the name of the event through which to access agent statistics is different in Java than it is in other languages supported by CTI OS.

- Through the Agent instance itself

The rest of this section describes these methods for accessing agent statistics.

### Registering to the eOnNewAgentStatisticsEvent() (JAVA)

To register to receive agent statistics, you must include the eOnNewAgentStatisticsEvent() in the message filter.

For example, in Java, the message filter to receive agent statistics is:

```java
String filter = S_MESSAGEID + "=" + CtiOs_Enums.EventID.eNewAgentStatisticsEvent;
```

For more information on message filters, see page 4-34.
Registering to the OnAgentStatistics() (C++, COM, and VB)

To register to receive agent statistics, you must include the OnAgentStatistics() event in the message filter.

For more information on message filters, see page 4-34.

How to Get Agent Statistics through the Agent Instance

Once you have used the EnableAgentStatistics() method for the agent, agent statistics are available through that Agent instance.

To get the agent statistics

---

Step 1
Get the Arguments instance containing statistics from the Agent instance using the GetValueArray() method.

Step 2
Parse the Arguments instance as needed to get specific statistics.

The following example demonstrates this task in Java:

/* 1. Get Arguments instance.*/
Arguments rArgs = agent.GetValueArray(CTIOS_STATISTICS);

/* 2. Parse as necessary. For example:*/
int availTimeSession = rArgs.GetValueIntObj(CTIOS_AVAILTIMESESSION);

Changing Which Agent Statistics are Sent

You can change which agent statistics are sent to applications by modifying the registry on the CTI OS Server.

For information on how to change which agent statistics are sent to applications by default, see the CTI OS System Manager’s Guide.

Agent Statistics Computed by the Sample CTI OS Desktop

The sample CTI OS Desktop are computes many agent statistics from data received from CTI Server. You may choose to develop applications that compute these same statistics. Therefore, these computed statistics (in italics) and the data and formulas used to derive them are listed below:
Working with Skill Group Statistics

Overview

This section describes how to receive and work with skill group statistics in a server-to-server integration environment and contains the following subsections:

- “How to Set Up a Monitor-Mode Application to Receive Skill Group Statistics” on page 4-60
- “Accessing Skill Group Statistics” on page 4-62
- “Changing Which Skill Group Statistics are Sent” on page 4-63
• “Skill Group Statistics Computed by the Sample CTI OS Desktop” on page 4-63

How to Set Up a Monitor-Mode Application to Receive Skill Group Statistics

To set up a Monitor-mode application to receive skill group statistics:

---

**Step 1**  Create an instance of the Session class, as described on page 4-27.

**Step 2**  Subscribe for events for the session, as described on page 4-28.

**Note**  You must register to receive session and skill group events; therefore, in the AddEventListener() method you must use as a parameter the field `CtiOsEnums.SubscriberList.eAllInOneList`, or you must call the method twice using the fields `CtiOsEnums.SubscriberList.eSessionList` and `CtiOsEnums.SubscriberList.eSkillGroupList`.

---

**Step 3**  Set connection parameters, as described on page 4-28.

**Step 4**  Connect the desktop application to the CTI OS Server, as described on page 4-29.

**Step 5**  Set the message filter as described on page 4-35.

a. Create String for the filter using the keyword `S_FILTERTARGET` as the name and the keyword `CTIOS_SKILLGROUPSTATISTICS` as the value.

b. Create an instance of the Arguments class.

c. Set the value in the filter for the `CTIOS_FILTER` keyword to the String created in Step a.

d. Use the `SetMessageFilter()` method in the Session class to set the filter for the session, using the Arguments instance you created in Step b as a parameter.

**Step 6**  Enable individual statistics as needed.

a. Create an instance of the Arguments class.
b. Set values in the Arguments instance. You must provide the skill group number and the peripheral number for each skill group for which you want to receive statistics. Use the SetValue(keyword, int) method signature.

   For example: use SetValue(CTIOS_SKILLGROUPNUMBER, sgNumber) where sgNumber is an integer for the skill group for which you want to receive statistics, and SetValue(CTIOS_PERIPHERALID, peripheralNumber) where sgNumber is an integer for the skill group for which you want to receive statistics.

   Caution The application must be aware of the Skill Group ID, and the skill group’s Peripheral ID, for any skill group to monitor; the application cannot dynamically get these values from CTI OS Server.

c. Use the Arguments instance as a parameter for the session’s EnableSkillGroupStatistics() method.

d. Repeat steps b and c for each skill group for which you want to receive events.

Step 7 When the desktop application no longer requires the statistics for a certain skill group, the application can disable those statistics.

   a. Create an instance of the Arguments class.

   b. Set values in the Arguments instance. You must provide the skill group number and the peripheral number for each skill group for which you want to receive statistics. Use the SetValue(keyword, int) method signature.

   For example: use SetValue(CTIOS_SKILLGROUPNUMBER, sgNumber) where sgNumber is an integer for the skill group for which you want to receive statistics, and SetValue(CTIOS_PERIPHERALID, sgNumber) where sgNumber is an integer for the skill group for which you want to stop receiving statistics.

   c. Use the Arguments instance as a parameter for the session’s DisableSkillGroupStatistics() method.
The following example demonstrates this task in Java:

```java
/* 1. Create session. */
CtiOsSession rSession = new CtiOsSession();

/* 2. Add event listener. */
rSession.AddListener(this,
    CtiOs.Enums.SubscriberList.eSessionList);
rSession.AddListener(this,
    CtiOs.Enums.SubscriberList.eSkillGroupList);

/* 3. Set Connection values. */
Arguments rArgs = new Arguments();
rArgs.SetValue(CTIOS_CTIOSA, "CTIOSServerA");
rArgs.SetValue(CTIOS_PORTA, 42408);
rArgs.SetValue(CTIOS_CTIOSB, "CTIOSServerB");
rArgs.SetValue(CTIOS_PORTB, 42408);
rArgs.SetValue(CTIOS_HEARTBEAT, 100);

/* 4. Connect to server. */
int returnCode = session.Connect(rArgs);

/* 5. Set the message filter. */
String filter = S_FILTERTARGET + "=" + CTIOS_SKILLGROUPSTATISTICS;
rArgs = new Arguments();
rArgs.SetValue(CtiOs.IKeywordIDs.CTIOS_FILTER, filter);
returnCode = session.SetMessageFilter(rArgs);

/* 6. Enable statistics. */
rArgs = new Arguments();
rArgs.SetValue(CTIOS_SKILLGROUPID, sgID);
rArgs.SetValue(CTIOS_PERIPHERALID, peripheralID);
rSession.EnableSkillGroupStatistics(rArgs);
```

### Accessing Skill Group Statistics

#### Overview

Once the application is set up to receive skill group statistics, as described in the preceding section, you access skill group statistics through an event handler. The name of the event depends on the language of the application:

- In Java, `onNewSkillGroupStatisticsEvent`
- In C++, COM, or VB, `OnSkillGroupStatisticsUpdated`
Caution  
Note carefully that the name of the event through which to access skill group statistics is different in Java than it is in other languages supported by CTI OS.

Registering to the eOnNewSkillGroupStatisticsEvent() (JAVA)

To register to receive skill group statistics, you must include the eOnNewSkillGroupStatisticsEvent() in the message filter.

For example, in Java, the message filter to receive skill group statistics is:

```java
String filter = S_MESSAGEID + "=" + CtiOs_Enums.EventID.eNewSkillGroupStatisticsEvent;
```

For more information on message filters, see page 4-34.

Registering to the OnSkillGroupStatisticsUpdated() (C++, COM, and VB)

To register to receive skill group statistics, you must include the OnSkillGroupStatisticsUpdated() event in the message filter.

For more information on message filters, see page 4-34.

Changing Which Skill Group Statistics are Sent

You can change which skill group statistics are sent to desktop applications by modifying the registry on the CTI OS Server.

For information on how to change which skill group statistics are sent to desktop applications, see the CTI OS System Manager’s Guide.

Skill Group Statistics Computed by the Sample CTI OS Desktop

The sample CTI OS Desktop are computes many skill group statistics from data received from CTI Server. You may choose to develop applications that compute these same statistics. Therefore, these computed statistics (in *italics*) and the data and formulas used to derive them are listed below:
Working with Skill Group Statistics

- \( \text{AvgCallsQTimeNow} = \frac{\text{CallsQTimeNow}}{\text{CallsQNow}} \)
- \( \text{AvgAgentOutCallsTalkTimeToHalf} = \frac{\text{AgentOutCallsTalkTimeToHalf}}{\text{AgentOutCallsToHalf}} \)
- \( \text{AvgAgentOutCallsTimeToHalf} = \frac{\text{AgentOutCallsTimeToHalf}}{\text{AgentOutCallsToHalf}} \)
- \( \text{AvgAgentOutCallsHeldTimeToHalf} = \frac{\text{AgentOutCallsHeldTimeToHalf}}{\text{AgentOutCallsHeldToHalf}} \)
- \( \text{AvgHandledCallsTalkTimeToHalf} = \frac{\text{HandledCallsTalkTimeToHalf}}{\text{HandledCallsToHalf}} \)
- \( \text{AvgHandledCallsAfterCallTimeToHalf} = \frac{\text{HandledCallsAfterCallTimeToHalf}}{\text{HandledCallsToHalf}} \)
- \( \text{AvgHandledCallsTimeToHalf} = \frac{\text{HandledCallsTimeToHalf}}{\text{HandledCallsToHalf}} \)
- \( \text{AvgIncomingCallsHeldTimeToHalf} = \frac{\text{IncomingCallsHeldTimeToHalf}}{\text{IncomingCallsHeldToHalf}} \)
- \( \text{AvgInternalCallsRcvdTimeToHalf} = \frac{\text{InternalCallsRcvdTimeToHalf}}{\text{InternalCallsRcvdToHalf}} \)
- \( \text{AvgInternalCallsHeldTimeToHalf} = \frac{\text{InternalCallsHeldTimeToHalf}}{\text{InternalCallsHeldToHalf}} \)
- \( \text{AvgCallsQTimeHalf} = \frac{\text{CallsQTimeHalf}}{\text{CallsQHalf}} \)
- \( \text{AvgAgentOutCallsTalkTimeToday} = \frac{\text{AgentOutCallsTalkTimeToday}}{\text{AgentOutCallsToday}} \)
- \( \text{AvgAgentOutCallsTimeToday} = \frac{\text{AgentOutCallsTimeToday}}{\text{AgentOutCallsToday}} \)
- \( \text{AvgAgentOutCallsHeldTimeToday} = \frac{\text{AgentOutCallsHeldTimeToday}}{\text{AgentOutCallsHeldToday}} \)
- \( \text{AvgHandledCallsTalkTimeToday} = \frac{\text{HandledCallsTalkTimeToday}}{\text{HandledCallsToday}} \)
- \( \text{AvgHandledCallsAfterCallTimeToday} = \frac{\text{HandledCallsAfterCallTimeToday}}{\text{HandledCallsToday}} \)
- \( \text{AvgHandledCallsTimeToday} = \frac{\text{HandledCallsTimeToday}}{\text{HandledCallsToday}} \)
Enabling Silent Monitor in your Application

The silent monitor manager object is responsible for establishing and maintaining the state of a silent monitor session.

The first thing a client application should do is to create a silent monitor object instance. The application should then set this object instance as the current manager in the session object. The CIL provides the interface to this functionality.

A client application can work in one of two possible modes:

- **Monitoring mode.** The client receives audio from a remote monitored target (device/agent).
- **Monitored mode.** The client sends audio to a remote monitoring client.

Creating a Silent Monitor Object

The first step towards setting up a silent monitor session is creating a SilentMonitorManager using the Session object CreateSilentMonitorManager method. Then, set the new manager object to be the current silent monitor manager using the Session object SetCurrentSilentMonitor method.

The following VB code sample demonstrates how to create a SilentMonitorManager object and make it the current manager in the Session object:

```vbnet
Dim  errorcode   As Long
Dim m_nSMSessionKey As Integer
Dim m_SMManager  As CTIOSCLIENTLib.SilentMonitorManager
Dim m_Args As New Arguments

'Assemble arguments to create the silent monitor manager
```
Enabling Silent Monitor in your Application

m_Args.AddItem "HeartbeatInterval", "5"
m_Args.AddItem "HeartbeatTimeout", 3

'Create the silent monitor manager
Set m_SMManager = m_session.CreateSilentMonitorManager(m_Args)

'Make the object the current manager
errorcode = m_session.SetCurrentSilentMonitorManager(m_SMManager)

Setting the Session Mode

After you set this new object as the current object, set the manager’s work mode to Monitoring for the monitoring client and Monitored for the monitored client. The following sections provide code examples. See also Chapter 13, “SilentMonitorManager Object” for syntax of the StartSMMonitoringMode and SMMonitoredMode methods.

Monitoring Mode

In this mode the client receives audio, confirmation and session status events for a specific silent monitor session. This mode is intended for use by client applications developed for Supervisor desktop functionality. The StartSMMonitoringMode method on the SilentMonitorManager object selects this mode.

Following is a code sample for specifying the mode for the client application.

Dim m_Args As New Arguments

'Assemble arguments to set the work mode
'Port number where audio will be listened for
m_Args.AddItem "MediaTerminationPort", 8000

'Set the working mode to monitoring
m_SMManager.StartSMMonitoringMode(m_Args)

Monitored Mode

In this mode, the client sends audio and status reports on silent monitor session and receives requests for start and stop silent monitor session. This mode is intended for client applications developed for Agent desktop functionality. The StartSMMonitoredMode method on the SilentMonitorManager object selects this mode.
Following is a code sample for specifying the mode for the client application:

```vbscript
Dim m_Args As New Arguments

' Assemble arguments to set the work mode
' Extension number of the Hard IP Phone to monitor
m_Args.AddItem "MonitoredDeviceID", "1234"
' Port where audio will be sent
m_Args.AddItem "MonitoringIPPort", 99999
' Set the working mode to monitored
m_SMManager.StartSMMonitoredMode(m_Args)
```

### Initiating and Ending a Silent Monitor Session

Initiating a silent monitor session starts with the client, in monitoring mode, calling the StartSilentMonitorRequest method. This indicates that the CTI OS server should send an OnSilentMonitorStartRequestedEvent to a remote client in monitored mode. The remote client, upon receiving the OnSilentMonitorStartRequestedEvent, chooses whether or not to accept the request. The remote client acknowledges its approval or rejection by sending a status report back to the monitoring client. The monitoring client will receive the acceptance or rejection via the OnSilentMonitorStatusReportEvent. When the session is accepted by the remote client it, will immediately start forwarding voice to the monitoring client. The silent monitoring session can only be terminated by the monitoring client by calling the StopSilentMonitorRequest method, CTI OS server will issue the OnSilentMonitorStopRequestedEvent to the remote client. The monitored client will stop sending audio immediately when OnSilentMonitorStopRequestedEvent is received by its SilentMonitorManager object.

Following are code samples for initiating and ending a silent monitor session:

#### Monitoring Client Code Sample

```vbscript
Private Sub btnStartSM_OnClick()
Dim m_Args As New Arguments

' Assemble arguments for start request
' Agent to monitor
m_Args.AddItem "AgentID", "23840"
m_Args.AddItem "PeripheralID", "5000"

' Request silent monitor session to start
```
Enabling Silent Monitor in your Application

```vbscript
m_SMManager.StartSilentMonitorRequest(m_Args, m_nSMSessionKey)
End Sub

Private Sub m_session_OnSilentMonitorStatusReportEvent(ByVal IArguments As IArguments)
    Dim strAgent As String
    Dim nMode As Integer
    nMode = IArguments.GetValueInt("StatusCode")
    If nMode = eSMStatusMonitorStarted Then strAgent = IArguments.GetValueString("MonitoredUniqueObjectID")
        MsgBox "Silent Monitor Status" & strAgent
    Else
        MsgBox "Silent Monitor Status" & nMode
    End If
End Sub

Private Sub tmrScreening_Timer()
    'After listening the conversation for 30 sec, drop monitoring session
    'Assemble arguments for stop request
    'Agent to monitor
    m_Args.AddItem "SMSessionKey", m_nSMSessionKey
    'Request silent monitor session to stop
    m_SMManager.StopSilentMonitorRequest(m_Args, m_nSMSessionKey)
End Sub

Monitored Client Code Sample

```vbscript
Private Sub m_session_OnSilentMonitorStartRequestedEvent(ByVal IArguments As IArguments)
    Dim strRequestInfo As String
    strRequestInfo = IArguments.DumpArgs
    MsgBox "Request to Start Silent Monitor Received", strRequestInfo
End Sub

Private Sub m_session_OnSilentMonitorStopRequestedEvent(ByVal IArguments As IArguments, bDoDefaultProcessing)
    Dim strRequestInfo As String
    ...
Shutting Down Silent Monitor Manager

Shutting down the Silent monitor object requires that the monitoring client call the StopSilentMonitorMode method when it is done monitoring an agent, and that the monitored client call the StopSilentMonitorMode method during cleanup. Each client must then remove the silent monitor manager from the Session object by calling SetMonitorCurrentSilentMonitor with a NULL pointer. Finally each client must destroy the silent monitor object using Session’s DestroySilentMonitorManager method.

Following is a code sample for initiating and ending a silent monitor session:

```vbscript
strRequestInfo = pIArguments.DumpArgs
MsgBox "Request to Stop Silent Monitor Received", strRequestInfo
End Sub

'Stop Silent Monitor ModeRequest
m_SMManager.StopSilentMonitorMode
'Remove silent monitor manager object from session
errorcode = m_session_SetCurrentSilentMonitor(Nothing)
'Destroy silent monitor manager object
errorcode = m_session.DestroySilentMonitorManager()
```

Deployment of Custom CTI OS Applications

This section discusses the deployment of CTI OS applications in the various programming languages and interfaces.

Applications Using the ActiveX Controls

ActiveX controls need all the components for COM deployment plus the components listed in Table 4-2.

<table>
<thead>
<tr>
<th>DLL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agentselectctl</td>
<td>AgentSelect ActiveX control</td>
</tr>
<tr>
<td>agentstatectl.dll</td>
<td>Agentstate ActiveX control</td>
</tr>
</tbody>
</table>
Table 4-2  ActiveX Control DLLs (continued)

<table>
<thead>
<tr>
<th>DLL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlternateCtl.dll</td>
<td>Alternate ActiveX control</td>
</tr>
<tr>
<td>answerctl.dll</td>
<td>Answer/Release ActiveX control</td>
</tr>
<tr>
<td>arguments.dll</td>
<td>Arguments COM class</td>
</tr>
<tr>
<td>badlinectl.dll</td>
<td>Badline ActiveX control</td>
</tr>
<tr>
<td>buttoncontrol.dll</td>
<td>Basic Button ActiveX control</td>
</tr>
<tr>
<td>ccnsmt.dll</td>
<td>Cisco EVVBU Media Termination ActiveX control</td>
</tr>
<tr>
<td>chatctl.dll</td>
<td>Chat ActiveX control</td>
</tr>
<tr>
<td>conferencectl.dll</td>
<td>Conference ActiveX control</td>
</tr>
<tr>
<td>cticommandlgs.dll</td>
<td>Common Dialogs utility COM object</td>
</tr>
<tr>
<td>CTIOSAgentStatistics.dll</td>
<td>AgentStatistics ActiveX control</td>
</tr>
<tr>
<td>ctioscallappearance.dll</td>
<td>CallAppearance ActiveX control</td>
</tr>
<tr>
<td>ctiosclient.dll</td>
<td>COM cil interfaces</td>
</tr>
<tr>
<td>ctiossessionresolver.dll</td>
<td>COM sessionresolver</td>
</tr>
<tr>
<td>CTIOSSkillGroupStatistics.dll</td>
<td>SkillgroupStatistics ActiveX control</td>
</tr>
<tr>
<td>ctiosstatusbar.dll</td>
<td>StatusBar ActiveX control</td>
</tr>
<tr>
<td>EmergencyAssistCtl.dll</td>
<td>EmergencyAssist ActiveX control</td>
</tr>
<tr>
<td>gridcontrol.dll</td>
<td>GridControl ActiveX control</td>
</tr>
<tr>
<td>holdctl.dll</td>
<td>Hold/Retrieve ActiveX control</td>
</tr>
<tr>
<td>IntlResourceLoader.dll</td>
<td>Internationalization COM object</td>
</tr>
<tr>
<td>makecallctl.dll</td>
<td>MakeCall ActiveX control</td>
</tr>
<tr>
<td>ReconnectCtl.dll</td>
<td>Reconnect ActiveX control</td>
</tr>
<tr>
<td>recordctl.dll</td>
<td>Record ActiveX control</td>
</tr>
<tr>
<td>SilentMonitorCtl.dll</td>
<td>Standalone Silent Monitor ActiveX control</td>
</tr>
<tr>
<td>SubclassForm.dll</td>
<td>COM utility control</td>
</tr>
<tr>
<td>SupervisorOnlyCtl.dll</td>
<td>Supervisor ActiveX control</td>
</tr>
<tr>
<td>transferctl.dll</td>
<td>Transfer ActiveX control</td>
</tr>
</tbody>
</table>
ActiveX controls need to be copied and registered using the regsvr32 Windows utility. Some ActiveX controls are dependent on others. For example, all Button type controls (e.g. AgentStatectl.dll) depend on (buttoncontrol.dll) and all Grid type controls (e.g. CtiosCallappearance.dll) depend on Gridcontrol.dll. The following table below means that for a dll listed in the left column to work properly, all dll’s listed in the right column (“dependencies”) need to be available (copied and registered).

Table 4-3 lists the dependencies of CTI OS ActiveX controls.

**Table 4-3 Dependencies of CTIOS ActiveX Controls**

<table>
<thead>
<tr>
<th>DLL File</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agentselectctl</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll</td>
</tr>
<tr>
<td>agentstatectl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommondlgs.dll</td>
</tr>
<tr>
<td>AlternateCtl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll</td>
</tr>
<tr>
<td>answerctl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll</td>
</tr>
<tr>
<td>arguments.dll</td>
<td>ATL.dll</td>
</tr>
<tr>
<td>badlinectl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll</td>
</tr>
<tr>
<td>buttoncontrol.dll</td>
<td>ATL.dll</td>
</tr>
<tr>
<td>ccnsmt.dll</td>
<td>Traceserver.dll, LIBG723.dll</td>
</tr>
<tr>
<td>chatctl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll</td>
</tr>
<tr>
<td>conferencectl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommondlgs.dll</td>
</tr>
<tr>
<td>cticommondlgs.dll</td>
<td>ATL.Dll, ctiosclient.dll, arguments.dll</td>
</tr>
<tr>
<td>CTIOSAgentStatistics.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll, Gridcontrol.dll</td>
</tr>
<tr>
<td>ctioscallappearance.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommondlgs.dll</td>
</tr>
</tbody>
</table>
Cisco ICM Software CTI OS Developer's Guide Release 6.0(0)

Table 4-3 Dependencies of CTIOS ActiveX Controls (continued)

<table>
<thead>
<tr>
<th>DLL File</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctiosclient.dll</td>
<td>ATL.dll, arguments.dll, ctiosractext.exe.</td>
</tr>
<tr>
<td></td>
<td>If the client application will use silent</td>
</tr>
<tr>
<td></td>
<td>monitoring in monitoring mode, ccnsmt.dll is</td>
</tr>
<tr>
<td></td>
<td>also a dependency.</td>
</tr>
<tr>
<td></td>
<td>If the client application will use silent</td>
</tr>
<tr>
<td></td>
<td>monitoring in monitored mode, wpcap.dll is</td>
</tr>
<tr>
<td></td>
<td>also a dependency.</td>
</tr>
<tr>
<td>ctiossessionresolver.dll</td>
<td>Atl.dll, ctiosclient.dll, arguments.dll</td>
</tr>
<tr>
<td>CTIOSSkillGroupStatistics.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll,</td>
</tr>
<tr>
<td></td>
<td>Gridcontrol.dll</td>
</tr>
<tr>
<td>ctiosstatusbar.dll</td>
<td>Atl.dll, ctiosclient.dll, arguments.dll,</td>
</tr>
<tr>
<td></td>
<td>cticommondlgls.dll</td>
</tr>
<tr>
<td>EmergencyAssistCtl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll,</td>
</tr>
<tr>
<td></td>
<td>buttoncontrol.dll</td>
</tr>
<tr>
<td>gridcontrol.dll</td>
<td>ATL.dll</td>
</tr>
<tr>
<td>holdctl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll,</td>
</tr>
<tr>
<td></td>
<td>buttoncontrol.dll</td>
</tr>
<tr>
<td>IntlResourceLoader.dll</td>
<td>ATL.dll</td>
</tr>
<tr>
<td>makecallctl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll,</td>
</tr>
<tr>
<td></td>
<td>buttoncontrol.dll, cticommondlgls.dll</td>
</tr>
<tr>
<td>ReconnectCtl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll,</td>
</tr>
<tr>
<td></td>
<td>buttoncontrol.dll</td>
</tr>
<tr>
<td>recordctl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll,</td>
</tr>
<tr>
<td></td>
<td>buttoncontrol.dll</td>
</tr>
<tr>
<td>SilentMonitorCtl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll,</td>
</tr>
<tr>
<td></td>
<td>ccnsmt.dll</td>
</tr>
<tr>
<td>SubclassForm.dll</td>
<td>ATL.dll</td>
</tr>
<tr>
<td>SupervisorOnlyCtl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll,</td>
</tr>
<tr>
<td></td>
<td>buttoncontrol.dll</td>
</tr>
<tr>
<td>transferctl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll,</td>
</tr>
<tr>
<td></td>
<td>buttoncontrol.dll, cticommondlgls.dll</td>
</tr>
</tbody>
</table>
Applications Using COM (But Not ActiveX Controls)

Custom applications using COM from VB or C++ or any other Com supported development platform, need the following COM Dynamic Link Libraries.

- CTIOSClient.dll
- Arguments.dll
- CtiosSessionresolver.dll (only if used – see previous discussion)
- ATL.DLL (only if not already available on target system)
- If the client application will use silent monitoring in monitoring mode, ccnsmt.dll is needed. If the client application will use silent monitoring in monitored mode, wpcap.dll is also a dependency.

The Dll files need to be copied and registered on the target system. Registration is done by using the Windows utility regsvr32.exe providing the dll name (i.e., regsvr32 ctiosclient.dll).

ATL.DLL is a Microsoft Dynamic Link Library implementing the Active Template Library used by CTI OS. It will usually be available on most Windows client systems in a windows system directory (e.g. \winnt\system32 on Windows 2000). Since CTI OS depends on this DLL, it needs to be copied and registered if it is not already available at the target system.

Deployment of C++ Applications

Custom C++ applications link to the static CTI OS libraries. With your custom application, you should also distribute ctiostracetext.exe. For the tracing component to work, you need to register it on the system where your application will run. To register the trace tool run ctiostracetext /RegServer. Besides ctiostracetex.exe, there is no need to ship additional components.
CTI OS ActiveX Softphone Controls

The CTI OS Developer's Toolkit includes a set of ActiveX controls to enable rapid application development. ActiveX controls are typically UI components (there are also ActiveX controls which are invisible at run time) that enable easy drag-and-drop creation of custom CTI applications in a variety of container applications. Container applications include: Microsoft Visual Basic, Microsoft Internet Explorer, Microsoft Visual C++, Borland Delphi, Sybase Powerbuilder and other applications supporting the OC96 ActiveX standard.

The CTI OS Agent Desktop and CTI OS Supervisor Desktop for IPCC Enterprise applications are both Visual Basic applications based on the CTI OS ActiveX controls. See also the Cisco ICM Software CTI OS Agent Desktop User Guide as well as the Cisco ICM Software CTI OS Supervisor Desktop for IPCC Enterprise User Guide for further reference on features of the CTI OS ActiveX controls.

Table 5-1 lists the ActiveX controls included with CTI OS. As seen in the table, CTI OS Controls can be grouped into Agent Related Controls, Call Related Buttons, Statistics Controls, and Supervisor Controls.

Table 5-1  CTI OS ActiveX Controls

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentRelated Controls</td>
<td></td>
</tr>
<tr>
<td>AgentStateCtl</td>
<td>Provides UI for login, logout, ready, not ready and wrapup requests, also enables the use to specify reason codes for logout and Not_Ready (if supported and configured).</td>
</tr>
</tbody>
</table>
ChatCtl Provides UI to send text messages to a supervisor or (if allowed) to other agents.

EmergencyAssistCtl Provides UI to place Emergency and Supervisor Assist calls. These calls allow agents to conveniently contact a supervisor if they need help. The corresponding ICM scripts must be configured in order for this control to work.

Call Related Controls

AlternateCtl Provides UI for alternate requests. If an agent has Call A active and Call B on hold, alternate will put call A on hold and make Call B active. Useful during consult calls.

AnswerCtl Provides UI to answer a call. Only a call with state “LCS_Alerting” (Ringing) can be answered.

BadLineCtl Provides a UI to report a Bad Line, e.g. bad voice quality or equipment problems.

CallAppearanceCtl A grid control displaying call information, including call status and context data.

ConferenceCtl Provides UI to place a conference call in single step or consultative mode.

HoldCtl Provides UI to put calls on hold and retrieve them.

MakeCallCtl Provides UI to enter a telephone number and place a make call request.

ReconnectCtl Provides a UI for reconnect requests. If an agent has Call A active and Call B on hold, reconnect will hang up call A and make Call B active. Useful during consult calls to return to the original call.

StatusBarCtl Visually displays information about the logged on agent (id, instrument, extension, current state).

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChatCtl</td>
<td>Provides UI to send text messages to a supervisor or (if allowed) to other agents.</td>
</tr>
<tr>
<td>EmergencyAssistCtl</td>
<td>Provides UI to place Emergency and Supervisor Assist calls. These calls allow agents to conveniently contact a supervisor if they need help. The corresponding ICM scripts must be configured in order for this control to work.</td>
</tr>
<tr>
<td>AlternateCtl</td>
<td>Provides UI for alternate requests. If an agent has Call A active and Call B on hold, alternate will put call A on hold and make Call B active. Useful during consult calls.</td>
</tr>
<tr>
<td>AnswerCtl</td>
<td>Provides UI to answer a call. Only a call with state “LCS_Alerting” (Ringing) can be answered.</td>
</tr>
<tr>
<td>BadLineCtl</td>
<td>Provides a UI to report a Bad Line, e.g. bad voice quality or equipment problems.</td>
</tr>
<tr>
<td>CallAppearanceCtl</td>
<td>A grid control displaying call information, including call status and context data.</td>
</tr>
<tr>
<td>ConferenceCtl</td>
<td>Provides UI to place a conference call in single step or consultative mode.</td>
</tr>
<tr>
<td>HoldCtl</td>
<td>Provides UI to put calls on hold and retrieve them.</td>
</tr>
<tr>
<td>MakeCallCtl</td>
<td>Provides UI to enter a telephone number and place a make call request.</td>
</tr>
<tr>
<td>ReconnectCtl</td>
<td>Provides a UI for reconnect requests. If an agent has Call A active and Call B on hold, reconnect will hang up call A and make Call B active. Useful during consult calls to return to the original call.</td>
</tr>
<tr>
<td>StatusBarCtl</td>
<td>Visually displays information about the logged on agent (id, instrument, extension, current state).</td>
</tr>
</tbody>
</table>
### Table 5-1 CTI OS ActiveX Controls (continued)

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecordCtl</td>
<td>Provides UI for Call Recording requests (start/stop recording), the requests will be forwarded to CTI Server, so they can be handled by a configured call recording service.</td>
</tr>
<tr>
<td>TransferCtl</td>
<td>Provides UI to transfer a call in single step or consultative mode.</td>
</tr>
</tbody>
</table>

#### Statistics Controls

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentStatisticsCtl</td>
<td>A grid control displaying real-time agent statistics. Columns are configurable at CTI OS server (see the Cisco ICM Software CTI OS System Manager’s Guide).</td>
</tr>
<tr>
<td>SkillgroupStatisticsCtl</td>
<td>A grid control displaying real-time skill group statistics. Columns are configurable at CTI OS server (see the Cisco ICM Software CTI OS System Manager’s Guide).</td>
</tr>
</tbody>
</table>

#### Supervisor Controls

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentSelectCtl</td>
<td>Supervisor specific; displays all agent team members of a supervisor (configured by ICM), including agent name, agentid, agentstate, timeinstate and skillgroups.</td>
</tr>
<tr>
<td>SupervisorOnlyCtl</td>
<td>Provides UI for Supervisor functions including Barge-In, Intercept, logout monitored agent and make monitored agent ready.</td>
</tr>
<tr>
<td>SilentMonitorCtl.dll</td>
<td>Standalone control that provides the capability of creating a monitoring application that connects to CTI OS, but does not need to login as a supervisor.</td>
</tr>
</tbody>
</table>

### Property Pages

While most settings in CTI OS are downloaded from CTI OS server to the client, ActiveX controls additionally offer property pages. The activation of the property pages is container dependent (e.g. in Visual Basic, you can “right-click” on an
ActiveX control and select Properties from the pop-up menu). In CTI OS the most common properties selectable via property pages are ButtonType (e.g., The Holdctl can be a hold or retrieve button), as well as fonts and colors.

Button Controls and Grid Controls

Most of the CTI OS ActiveX controls are either Button Type Controls or Grid Type Control, with the following exceptions:

- Statusbarcontrol
- ChatCtl
- Some of the Utility controls (CtiCommonDlgs, SubClassFormCtl).

As such they share common principles.

Button Controls

Button Controls include the AgentStateCtl, AlternateCtl, AnswerCtl, BadLineCtl, ConferenceCtl, EmergencyAssistCtl, HoldCtl, MakeCallCtl, ReconnectCtl, SupervisorOnlyCtl, RecordCtl, and TransferCtl. They provide an UI to perform a certain CTI task (like logging in or answering a call). All of the Button Controls are based on the CTI OS ButtonCtl (see Utility controls) and share the same characteristics. All CTI OS buttons will enable and disable themselves automatically based on the current state of the system. For example, if an agent is not logged in, the only button available to click is the Login Button (see AgentStateCtl), or if a call has not been answered and is selected in the CallAppearanceCtl, the Answer Button will be enabled (see AnswerCtl and CallAppearanceCtl). All buttons can be configured via their property pages to show custom text captions, custom icons and custom tooltip captions.

Grid Controls

Grid controls include the AgentSelectCtl, CallAppearanceCtl, AgentStatisticsCtl and SkillGroupStatisticsCtl. The Grid Controls are used to display data, select calls (see CallAppearanceCtl) or Agents (AgentSelectCtl), or in some cases
enable you to enter data (e.g. Callvariables in the CallAppearanceCtl). The following grid properties can be configured by CTI OS server (see the *Cisco ICM Software CTI OS System Manager’s Guide*):

- Columns to display
- Column header
- Column width
- Column alignment

**CTI OS ActiveX Control Descriptions**

This section describes the CTI OS ActiveX softphone controls listed in Table 5-1.

**AgentStateCtl**

The agentstate control is based on the CTI OS button control and can be one of several button types. To select the button type, bring up the property page (container dependent, for example right click in VB) and select the desired agentstate functionality from the following:

- **Login Button.** Pressing the login button will bring up the Login dialog box (Figure 5-1) to allow the agent to select a connection profile (see the *Cisco ICM Software CTI OS System Manager’s Guide*), agent id and instrument or other switch specific fields.
The fields displayed can be configured. The dialog shows a login dialog configured for IPCC. An agent logging in can select a connection profile for the **Connect To:** drop down box, enter, agent id, password and instrument and click on OK to send a Login request.

- **Logout Button.** Clicking the logout button will send a request to CTI OS server, to log out the currently logged in agent. For some switches, including IPCC, the agent needs to be in the not ready state in order for this button to be enabled. If Reason Codes are supported on the switch and configured on ICM, a reason code dialog will pop up as shown in Figure 5-2.
This dialog lets you select a reason code to be sent along with the logout request. Reason codes can be configured at CTI OS server.

- **Ready Button.** Clicking the ready button will send a request to CTI OS server, to put the agent in ready state (ready to accept calls).

- **Not Ready Button.** Clicking the not ready button will send a request to CTI OS server, to put the agent in not ready state (ICM will not route calls to an agent in the not ready state). If Reason Codes are supported on the switch and configured on ICM, a reason code dialog will pop up as shown in Figure 5-3.
Figure 5-3  Reason Code Dialog for Not Ready

This dialog lets you select a reason code to be sent along with the not_ready request. Reason codes can be configured at the CTI OS Server.

- **Work Ready Button.** Clicking this button will send a request to CTI OS server to put the agent in the work ready or wrapup state. The behavior of this button depends on the wrapup mode support of the switch. On IPCC, the behavior is controlled by the ICM AgentDeskSettings (see the Cisco ICM Software IPCC Administrator’s Guide).

- **Work Not Ready Button.** Clicking this button will send a request to CTI OS server to put the agent in the work not ready or wrapup state. The behavior of this button depends on the wrapup mode support of the switch. On IPCC, the behavior is controlled by the ICM AgentDeskSettings (see the Cisco ICM Software IPCC Administrator’s Guide).
AgentSelectCtl

The agent select control is used for supervising agents and becomes active if the currently logged in agent is a supervisor. When a supervisor has logged on, this grid based control will display all agent team members of a supervisor (configured by ICM), including agent name, AgentID, AgentState, TimeInState and SkillGroups. The TimeInState column will be reset in real-time as the agents change state. If an agent remains in a state for more than 10 minutes, the TimeInState figure will be displayed in red.

Figure 5-4 Agent Select Grid Populated with Sample Data

<table>
<thead>
<tr>
<th>Name</th>
<th>AgentID</th>
<th>State</th>
<th>Time In State</th>
<th>SkillGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>lead 5111</td>
<td>5111</td>
<td>Unknown</td>
<td>0.50</td>
<td>2</td>
</tr>
<tr>
<td>lead 5112</td>
<td>5112</td>
<td>Unknown</td>
<td>0.50</td>
<td>2</td>
</tr>
<tr>
<td>lead 5101</td>
<td>5101</td>
<td>Talking</td>
<td>0.51</td>
<td>234</td>
</tr>
<tr>
<td>lead 5102</td>
<td>5102</td>
<td>NotReady</td>
<td>0.51</td>
<td>2</td>
</tr>
</tbody>
</table>

The agent select control handles the following events:

- **OnNewTeamMember.** Informs the supervisor of a new team member or a team member change. This will cause a row in the agentselect grid to be updated (add/remove agent).

- **OnMonitoredAgentStateChange.** Informs the supervisor of an agent state change. The new agentstate will be displayed in the State column and the TimeInState Column will be set to zero.

- **OnAgentInfo Event.**

  A supervisor can select a “currently monitored agent” by clicking on an agent displayed in the grid. This causes a set monitored agent method call on the agent object. Any supervisory action (for example logout monitored agent – see SupervisorOnlyCtl) will be performed on the “currently monitored” agent.
AgentStatisticsCtl

The AgentStatistics control is a grid based control displaying ICM agent real time statistics. The columns displayed are configurable at CTI OS server (see the Cisco ICM Software CTI OS System Manager’s Guide). Also, the update interval can be adjusted. It defaults to 10 seconds.

Figure 5-5  Agent Statistics Grid

<table>
<thead>
<tr>
<th>CellsHandledToday</th>
<th>TimeLoggedInToday</th>
<th>TimeTalkingToday</th>
<th>TimeHoldingToday</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5:17.95</td>
<td>0:00.00</td>
<td>0:00.00</td>
<td></td>
</tr>
</tbody>
</table>

AlternateCtl

The AlternateCtl is a Button type control allowing the agent to send an alternate call request. Alternate is a compound action of placing an active call on hold and then retrieving a previously held call or answering an alerting (ringing) call on the same device. Alternate is a useful feature during a consult call.

AnswerCtl

The Answer Control is a button that provides UI for sending answer and release call requests. The behavior (answer or release) can be set via the ButtonType set from the property page as explained under AgentState controls.

Answer Icon:
Release Icon:

**BadLineCtl**

The Bad Line Control is a button that provides UI for reporting a Bad Line. This request will generate a database entry in ICM and is an indicator for voice/equipment problems.

**CallAppearanceCtl**

The CallAppearance Control is a grid based control displaying call information, including call status and call context data (i.e., CallVariable1 through CallVariable10 and ECC variables).
Each incoming or outgoing call is displayed in one row in the grid. When a call first arrives, it will usually show a status of “Ringing”, until it is answered. A call can be answered by double clicking the call in the grid as well clicking on the Answer Button. Some columns in the CallAppearance grid can be edited if so configured (for example, the Columns displaying Callvariables) by clicking on the cell to be edited.

The grid can display multiple calls (see Figure 5-6). If the grid is displaying multiple calls, a user can select a call by clicking anywhere on the row where the call is displayed. This will highlight the whole row displaying this call (e.g. in Figure 5-6 the call with id 16777886 is currently selected). Any button controls (e.g., Answer, Release, Hold,) will enable or disable themselves based on the state the newly selected call is in.

The CallAppearance grid handles most call related events. It will display a call as soon as it receives an eCallBeginEvent. It will update the CallStatus and CallContext (CallVariables and ECC variables) on eCallDataUpdate and other call events (eServiceInitiated, eCallEstablished,). It will erase the call from the grid when it receives an eCallEnd event.

The CallAppearance grid can be in one of two modes. In “normal” mode it will show any calls for the agent/supervisor logged in; in “monitored” mode (only for supervisor), the CallAppearance grid will display all calls for a currently monitored agent (see Agent Select grid). A supervisor can then select a “monitored call” by clicking on a row in the grid to perform supervisory functions like barge-in or intercept (see SupervisorOnly control).
ChatCtl

The Chat Control provides a UI to formulate and send text messages to a supervisor or (if allowed) other agents. The chat privileges are configurable at CTI OS server (see the Cisco ICM Software CTI OS System Manager’s Guide).

![Chat Control](CTI-OS.Chat.png)

You can specify an AgentID in the field labeled **Send to AgentID** and then enter a message in the **Edit Outgoing Message** box. Clicking the **Send** Button will send the message. Incoming messages will be displayed in the **Message Display**. Clicking on the **Clear** button allows an agent to clear the display.

The ChatCtl does not implement a button directly, but may be linked to a button through Visual Basic, so that clicking the button will pop up the ChatCtl.

ConferenceCtl

The conference control is used to create a conference call. This can be done in either single step or consultative mode.
Icon for ConferenceInit:

![Icon for ConferenceInit]

Icon for Conference Complete:

![Icon for Conference Complete]

Depending on call status, pressing the Conference button once will bring up the dialog shown in Figure 5-8 (see also MakeCall dialog):

**Figure 5-8  The Conference Init Dialog**

![CTI Dialing Pad](Figure5-8.png)

- Mute Tones:
  - All
  - DTMF
  - Dial
  - Ring In
  - Originating
This dialog is similar to the Make Call dialog. It allows you to initiate a consultative Conference (Conf Init) or to place a Single Step Conference call.

Enter the number you wish to dial by either typing it into the text box labeled **Number to Dial** or by clicking the numbers on the displayed keypad. Once the number is entered you can click on **Conf Init** to place a consultative conference call or **Single Step** to initiate a single step conference. This will close this dialog. If you choose to place a consultative call, the conference button will change to **Conference Complete**. You must press this button to complete the conference after talking to the consult agent.

The conference dialog also has a **Mute Tones** section that allows you to suppress audio output of selected or all tones.

The **More** button brings up an additional section of the dialog displaying all CallVariables along with any values set in the original call. The agent may change or add values to send along with the consult call by double clicking on the appropriate line in the Value column (see Figure 5-9).

**Figure 5-9 Expanded Dialog**

---

**EmergencyAssistCtl**

The EmergencyAssistCtl is a button that provides a UI to place emergency or supervisor assist calls to a supervisor. On the ICM side this functionality is implemented with a script (see the *Cisco ICM Software CTI OS System Manager's Guide*). The main difference between the emergency call and supervisor assist
requests is the script to be run. An agent may click this control whether he has a call or not. If the agent has an active customer call, clicking this button will place a consult call to the supervisor. The “Conference Complete” as well as the “Transfer Complete” will be enabled to allow the agent to either conference the supervisor into the call or to transfer the call to the supervisor. If configured, clicking this button can also cause a single step conference. The behavior (emergency call or supervisor assist) can be set via the ButtonType property set from the Property Page, as described under AgentState controls.

Emergency Icon:

Supervisor Assist Icon:

HoldCtl

The HoldCtl is a button that provides a UI for sending hold and retrieve call requests. The behavior (hold or retrieve) can be set via the ButtonType property set from the Property Page, as described under AgentState controls.

Hold Icon:

Retrieve Icon:
MakeCallCtl

The MakeCallCtl is used to place calls and to generate DTMF tones. When this button is clicked it will bring up the dialing pad dialog box to enter data and place a makecall request (Figure 5-10).

Figure 5-10 Dial Dialog

Enter the number you wish to dial by either typing it into the textbox labeled Number to Dial or by clicking the numbers on the displayed keypad. Once the number is entered you can click on Make Call to send the MakeCall request.

This dialog also has a Mute Tones section that allows you to suppress audio output of selected or all tones.
You may enter values for CallVariable1 through CallVariable10 and ECC Call Variables via the Dial Dialog. Clicking the More button on the dialog extends it to display a grid listing all possible Call Variables. A value may be entered for each of these variables by double clicking on the appropriate line in the Value column (see Figure 5-11).

**Figure 5-11 Expanded Dialog**

![Expanded Dialog](image)

If the agent is on a call while pressing the Make Call button, the dialpad will be displayed without the MakeCall feature. The agent can then use the dialpad to play DTMF tones.

**ReconnectCtl**

The ReconnectCtl is a Button control allowing the agent to send a Reconnect Call request. Reconnect is a useful feature during a consult call. If an agent has Call A held and Call B active, reconnect will hang up Call B and make Call A active. In a consult call scenario, reconnect will hang up the consult call and return to the original call.
SkillgroupStatisticsCtl

The SkillGroupStatistics control is a grid based control displaying ICM real time SkillGroup statistics.

The columns displayed are configurable at CTI OS server (see the Cisco ICM Software CTI OS System Manager’s Guide). The update interval can be configured but defaults to 10 seconds.

If an agent belongs to multiple SkillGroups, each row will display statistics for one SkillGroup. For a supervisor this control will display all skillgroups in his team.

Figure 5-12 SkillgroupStatisticsCtl Displaying Sample Data for Three Skillgroups

<table>
<thead>
<tr>
<th>SkillGroupNumber</th>
<th>AgentLbLoggedOn</th>
<th>AgentAval</th>
<th>AgentNReady</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

StatusBarCtl

The CTI OS statusbar control displays information about the logged on agent as well as CTI OS specific details (Figure 5-13).

Figure 5-13 StatusBar Control Displaying Sample Data

The statusbar is separated into several panes. The panes are defined as follows:

- Pane 1: displays current extension and instrument
- Pane 2: displays Agent ID
- Pane 3: Message Waiting Indicator. If media termination is used and Voicemail is active, this pane will display “Voicemail” to indicate Voicemail was left.
- Pane 4: displays Agent State
- Pane 5: displays the CTI OS server currently connected to
- Pane 6: displays overall status (online, offline)

**SupervisorOnlyCtl**

The SupervisorOnly Control provides buttons for Supervisor functions including Barge-In, Intercept, Logout Monitored Agent and make Monitored Agent Ready. The behavior of the button can be set in the General tab of the Property Page.

**Logout Monitored Agent**: Logs out the currently monitored agent (set for example via the AgentselectCtl). If the currently monitored agent has a call active, the request will be queued and the agent will be logged out as soon as the call ends.

**Set monitored agent ready**: Forces the currently monitored agent from the “not ready” state into the ready state.

**Barge-In**: Lets the supervisor participate in the currently monitored call. The currently monitored call is selected via the CallAppearanceCtl (in monitor mode). Barge-in is really a conference on behalf of the monitored agent.
**Intercept:** Intercept can only be applied on a previously barged in call. The monitored agent will be dropped out of the call and the supervisor is left with the customer in a call.

Together with the AgentSelectCtl and the CallAppearanceCtl (in monitor mode) the SupervisorOnlyCtl is used in the CTI OS Supervisor Desktop application to build the Agent Real Time Status window, as shown in Figure 5-14.

*Figure 5-14 Supervisor Softphone Agent-RealTime Status Window*

This window shows the AgentSelectCtl and the CallappearanceCtl in monitor mode on the right side and four instances of the SupervisorOnlyCtl on the left side. From top to bottom they are: “Make Monitored Agent Ready” (disabled, since Agent 5101 is talking), “Logout monitored Agent”, Barge-in and Intercept.

**Start Silent Monitor:** Initiates a silent monitor session with the currently monitored agent
Stop Silent Monitor: Terminates the currently ongoing silent monitored session

RecordCtl

The RecordCtl is a button that provides UI for Call Recording requests (start/stop recording), the requests will be forwarded to CTI Server, so they can be handled by a configured call recording service. To record a call a current call has to be selected (e.g. via the CallAppearanceCtl). Once the record button is clicked, it will turn into record stop button.

Icon for Record Start:

Icon for Record Stop:

TransferCtl

The TransferCtl is a button that provides UI to transfer a call in single step or consultative mode. The mechanism is the same as explained for the conference control.
Icon for TransferInit:

![Icon for TransferInit]

Icon for Transfer Complete:

![Icon for Transfer Complete]

Depending on call status, pressing the Transfer button once will bring up the dialog shown in Figure 5-15 (see also MakeCall dialog):

**Figure 5-15  Dial Dialog**

![CTI Dialing Pad](image-url)
This dialog is similar to the Make Call dialog. It allows you to initiate a consultative Transfer (Transfer Init) or to place a Single Step Transfer call.

Enter the number you wish to dial by either typing it into the text box labeled **Number to Dial** or by clicking the numbers on the displayed keypad. Once the number is entered you can click on **Conf Init** to place a consultative transfer call or **Single Step** to initiate a single step transfer. This will close this dialog. If you choose to place a consultative call, the transfer button will change to **Transfer Complete**. You must press this button to complete the transfer after talking to the consult agent.

The transfer dialog also has a **Mute Tones** section that allows you to suppress audio output of selected or all tones.

The **More** button brings up an additional section of the dialog displaying all CallVariables along with any values set in the original call. The agent may change or add values to send along with the consult call by double clicking on the appropriate line in the Value column (see Figure 5-9).

**Figure 5-16 Expanded Dialog**

---

### The Silent Monitor StandAlone Com Object

The Silent Monitor StandAlone ComObject provides an interface for easy integration with the CTI OS Silent Monitor functionality. The ComObject can be used in Visual Basic as well as other host containers. This section demonstrates the use of this control in Visual Basic.
The Silent Monitor StandAlone Com Object is supported for use on IPCC only.

The Standalone ComObject wraps calls to the CTI OS Session as well as SilentMonitor manager. It provides the following four methods (displayed in IDL format; IDL is the language used to define COM interfaces).

```csharp
interface ISilentMonitor : IDispatch
{
    [id(1), helpstring("method Connect to CTIOS")]
    HRESULT Connect ([in] IArguments * args, [out] int* returnvalue);
    [id(2), helpstring("method Disconnect to CTIOS")]
    HRESULT Disconnect ([in] IArguments * args, [out] int* returnvalue);
    [id(3), helpstring("method StartMonitoring to CTIOS")]
    HRESULT StartMonitoring ([in] IArguments * args, [out] int* returnvalue);
    [id(4), helpstring("method StopMonitoring to CTIOS")]
    HRESULT StopMonitoring ([in] IArguments * args, [out] int* returnvalue);
};
```

**Connect**

The Connect method establishes a Monitor Mode Session with the specified CTI OS Server. The syntax and parameters are the same as for the CTI OS session object Connect method (see “Returns” in Chapter 8, “Session Object”).

**Disconnect**

The Disconnect method disconnects an established session. This method has no required parameters. See “CreateSilentMonitorManager” in Chapter 8, “Session Object” for syntax and optional parameters.

**StartMonitoring**

The StartMonitoring method starts a Silent Monitor Session. The StartMonitoring Arguments array contains the following parameters,
The Silent Monitor StandAlone Com Object

Note

If a pointer to the agent object is available (e.g., a m_MonitoredAgent), The PeripheralID can be retrieved via m_MonitoredAgent.GetValueInt ("PeripheralID")

Table 5-2  StartMonitoring Arguments Array Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentID</td>
<td>AgentID of the agent to be monitored.</td>
</tr>
<tr>
<td>Peripheralnumber</td>
<td>PeripheralID of the Peripheral to which the Agent is logged in.</td>
</tr>
</tbody>
</table>

StopMonitoring

The StopMonitoring method stops a Silent Monitor Session. The StopMonitoring Arguments array contains the same parameters as the StartMonitoring method (Table 5-2).

SilentMonitor Com Object Events

The ComObject will fire the following events via a COM connection point event interface (again in IDL):

```plaintext
dispinterface _ISilentMonitorCtlEvents
{
    properties:
    methods:
        [id(1)] void OnConnection([in] IArguments *pIArguments);
        [id(2)] void OnConnectionFailure([in] IArguments *pIArguments);
        [id(5)] void OnMonitorModeEstablished([in] IArguments *pIArguments);
        [id(39)] void OnConnectionClosed([in] IArguments *pIArguments);
        [id(41)] void OnControlFailureConf([in] IArguments *pIArguments);
        [id(304)] void OnCtiOsFailure([in] IArguments *pIArguments);
        [id(502)] void OnCallRTPStartedEvent([in] IArguments *pIArguments);
}```
The Silent Monitor StandAlone Com Object

Following is a brief description of each event. These events are described in detail in the Session Object and Silent Monitor Object sections of Chapter 6, “Event Interfaces and Events.”

**Table 5-3 SilentMonitor Com Object Events**

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnConnection</td>
<td>Indicates that the connect method was successful in establishing a connection.</td>
</tr>
<tr>
<td>OnConnectionFailure</td>
<td>Indicates that an active connection has failed. Can also indicate a bad parameter in the Connect method</td>
</tr>
<tr>
<td>OnMonitorModeEstablished</td>
<td>Signals a successful call to SetMsgFilter. The call to Setmsgfilter is hidden by the Standalone control.</td>
</tr>
<tr>
<td>OnConnectionClosed</td>
<td>Disconnect was called and the connection is now closed.</td>
</tr>
<tr>
<td>OnControlFailureConf</td>
<td>A ControlFailureConf was received and can be handled.</td>
</tr>
<tr>
<td>OnCtiOsFailure</td>
<td>A CtiOsFailure event was received. This could be Silent Monitor specific error code.</td>
</tr>
<tr>
<td>OnCallRTPStartedEvent, OnCallRTPStoppedEvent</td>
<td>RTP events have been received signaling the start and stop of the RTP streams.</td>
</tr>
</tbody>
</table>
Deployment

The StandAlone Com Object is a COM dll that needs to be registered on the client system via the Regsvr32 Silentmonitorctl.dll. In addition, ccnsmt.dll and the two standard CTI OS COM dlls (CTIOSClient.dll and Arguments.dll) are also required.

Sample Usage in Visual Basic

The following sample code assumes a VB form with 4 buttons (Connect, Disconnect, StartMonitoring and StopMonitoring. If the parameters are based on edit fields, the source code below is all that is needed to silent monitor via CTI OS. It is important to note, that this control does not require supervisor privileges or even any login. The only event handler shown below (OnSessionDisconnected) is the one for a timed out Silent Monitor session. Other event handlers.

```vbnet
Dim WithEvents SilentMonitorCtl As SILENTMONITORCTRLLib.SilentMonitor
Dim m_Args As New Arguments
Const CIL_OK = 1

Private Sub btnConnect_Click()
    m_Args.clear
    m_Args.AddItem "CtiosA", "localhost"
    m_Args.AddItem "portA", "42028"
    Dim nRetVal As Long
    SilentMonitorCtl.Connect m_Args, nRetVal
    If nRetVal <> CIL_OK Then
```

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnSilentMonitorStatusReportEvent</td>
<td>This event is used to report status from a monitored client to the monitoring application.</td>
</tr>
<tr>
<td>OnStartSilentMonitorConf, OnStopSilentMonitorConf</td>
<td>These confs acknowledge that CTI OS handled the StartMonitoring and StopMonitoring request, respectively.</td>
</tr>
<tr>
<td>OnSilentMonitorSessionDisconnected</td>
<td>Indicates that the Silent Monitor session has timed out on the monitoring side.</td>
</tr>
</tbody>
</table>
Chapter 5      CTI OS ActiveX Softphone Controls

The Silent Monitor StandAlone Com Object

Private Sub btnStartMonitoring_Click()
    m_Args.clear
    m_Args.AddItem "AgentId", "1000"
    m_Args.AddItem "PeripheralID", "5004"
    Dim nRetVal As Long
    SilentMonitorCtl.StartMonitoring m_Args, nRetVal
    If nRetVal <> CIL_OK Then
        MsgBox "StartMonitoring returned error " + Str(nRetVal)
    End If
End Sub

Private Sub btnStopMonitoring_Click()
    m_Args.clear
    m_Args.AddItem "AgentId", "1000"
    m_Args.AddItem "PeripheralID", "5004"
    Dim nRetVal As Long
    SilentMonitorCtl.StopMonitoring m_Args, nRetVal
    If nRetVal <> CIL_OK Then
        MsgBox "StopMonitoring returned error " + Str(nRetVal)
    End If
End Sub

Private Sub SilentMonitorCtl_OnSessionDisconnected(ByVal pIArguments As SILENTMONITORCTLLib.IArguments)
    MsgBox "SilentMonitorSession Disconnected Event"
End Sub

AllagentsMonitor Sample

The Standalone control was also integrated with the Allagents Sample, which is called AllAgentsmonitor. This sample demonstrates how to connect to CTI OS, select an agent and silent monitor him or her. Essentially it creates some UI plus an Agent Selection tool around the code given in the previous section. No specific
agent team configuration is required (all agents are displayed and can be monitored). Also, no login is required. The user can connect to CTI OS server and can start/stop monitoring after selecting an agent.

The following screenshot shows an Agent selected. The Silent Monitor button is enabled.
The following screenshot shows Silent monitoring in progress. The Stop Monitoring button is enabled.
Event Interfaces and Events

This chapter describes the CTI OS Client Interface Library’s event publications mechanism. Programs written to take advantage of CTI interfaces are generally event driven, meaning that the application will have a code module executed when an external event arrives. The CIL interface provides a rich set of event interfaces and events for use by client programmers.

Events are generated asynchronously, either by the telephony equipment (e.g. phone, PBX, ACD, etc.) or by the CTI environment (ICM, CTI Server, or CTI OS Server). Each event passes an Arguments structure of key-value pairs that contains all of the event parameters. These parameters are discussed in greater detail in this chapter.

Event Publication Model

The CIL event interfaces discussed in this section and the following sections apply only to the C++, COM, and VB interfaces. See the section “Events in Java CIL” at the end of this chapter for a discussion of Java CIL counterpart events and event handling in the Java CIL.

The Client Interface Library provides a publisher-subscriber model for notifying clients of events. Client applications using the CIL can subscribe to one or more of the CIL event interfaces. For detailed information and examples for how to subscribe and unsubscribe for events, see Chapter 4, “Building Your Application.”
The published CIL event interfaces are organized around the different classes of CTI objects that the CIL provides. The event interfaces described in this chapter are:

- **ISessionEvents.** This interface publishes the events that relate to actions on the Session object.
- **IAgentEvents.** This interface publishes the events that relate to actions on Agent objects.
- **ICallEvents.** This interface publishes the events that relate to actions on Call objects.
- **ISkillGroupEvents.** This interface publishes the events that relate to actions on SkillGroup objects.
- **IButtonEnablementEvents.** This interface publishes the events that relate to changes in the enable-disable status of softphone buttons.
- **IMonitoredAgentEventsInterface.** This interface fires Agent events to a supervisor for his team members.
- **IMonitoredCallEventsInterface.** This interface fires Call events to a supervisor for one of his agent team members.
- **ISilentMonitorEvents.** This interface fires events to subscribers of the Silent Monitor interface.

The remainder of this chapter provides the detailed description of each event interface available from the CIL.

---

**Note**

The data type listed for each keyword is the standardized data type discussed in the section “CTIOS CIL Data Types” in Chapter 3, “CIL Coding Conventions.” See Table 3-1 for the appropriate language specific types for these keywords.

---

**ISessionEvents Interface**

The Session object fires events on the ISessionEvents interface. The following events are published to subscribers of the ISessionEvents interface.
OnCallTranslationRouteEvent

The TranslationRouteEvent is a pre-call indication. The event indicates the pending arrival of a call, and provides early access to the call context information. From a call flow perspective, this event can be used to begin an application or database lookup for the call context data before the call actually arrives at the agent’s teleset.

The contact is uniquely identified by the ICMEnterpriseUniqueID, which is a field based on the ICM’s 64-bit unique key (RouterCallKeyDay and RouterCallKeyCallID). This event does not indicate the creation of a Call object on the CTIOS server—only that the contact is being tracked. This is sufficient to be able to get and set data, which enables some powerful data-prefetching applications. When a OnCallBeginEvent follows for this same contact, the ICMEnterpriseUniqueID field will be send along with the call data. At that point, a custom application can set the call data on the appropriate call object.

Syntax

C++: void OnCallTranslationRouteEvent(Arguments& args)
COM: void OnCallTranslationRouteEvent (IArguments * args)
VB:  session_OnCallTranslationRouteEvent (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMEnterpriseUniqueID</td>
<td>STRING</td>
<td>This string is a globally unique key for this contact, which corresponds to the ICM 64 bit key. This parameter can be used to match this contact to a follow-on call event.</td>
</tr>
</tbody>
</table>
### ISessionEvents Interface

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call’s records in the ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>INT</td>
<td>The call key created by the ICM. The ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>RouterCallKeySequenceNumber</td>
<td>INT</td>
<td>Together with RouterCallKeyDay and RouterCallKeyCallID fields forms the TaskID.</td>
</tr>
<tr>
<td>NumNamedVariables</td>
<td>SHORT</td>
<td>Number of Named variables.</td>
</tr>
<tr>
<td>NumNamedArrays</td>
<td>SHORT</td>
<td>Number of Named Arrays.</td>
</tr>
<tr>
<td>ANI</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>UserToUserInfo</td>
<td>STRING</td>
<td>The ISDN user-to-user information element.</td>
</tr>
<tr>
<td>DNIS</td>
<td>STRING</td>
<td>The DNIS number to which this call will arrive on the ACD/PBX.</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits</td>
<td>STRING</td>
<td>The digits entered by the caller in response to IVR prompting.</td>
</tr>
<tr>
<td>CallVariable1</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>CallVariable10</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>ECC</td>
<td>ARGUMENTS</td>
<td>A nested Arguments structure of key-value pairs for all of the ECC data arriving with this call.</td>
</tr>
</tbody>
</table>
OnConnection

The OnConnection event is generated after the Connect method succeeds. It returns the name of the connected server and the connection time of day. The client application need not take any special action but may use it to display connection status.1

Syntax

C++: void OnConnection(Arguments& args)
COM: void OnConnection (IArguments * args)
VB: session_OnConnection (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventTime</td>
<td>INT</td>
<td>Integer value with time of day expressed in milliseconds.</td>
</tr>
<tr>
<td>CurrentServer</td>
<td>STRING</td>
<td>Name or tcp/ip address of the current connected CTI OS server.</td>
</tr>
</tbody>
</table>

OnConnectionClosed

The OnConnectionClosed message is generated when a connection is terminated. This message has no fields.
OnConnectionFailure

The OnConnectionFailure event is generated when an established connection fails. It returns the name of the failed connected server and the failure time of day. Retry is automatic and can be followed by an OnConnection event when connection is successfully reestablished. The client application need not take any special action but may use this event to display connection status.

Syntax

C++: void OnConnectionFailure(Arguments& args)
COM: void OnConnectionFailure (IArguments * args)
VB: session_OnConnectionFailure (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventTime</td>
<td>INT</td>
<td>Integer value with time of day expressed in milliseconds.</td>
</tr>
<tr>
<td>FailedServer</td>
<td>STRING</td>
<td>Name or tcp/ip address of the server that has failed to respond. See ReasonCode.</td>
</tr>
<tr>
<td>ReasonCode</td>
<td>SHORT</td>
<td>SERVER_CONNECTIONBROKEN, SERVER_MISSINGHEARTBEATS</td>
</tr>
</tbody>
</table>

OnConnectionRejected

The OnConnectionRejected event indicates that the client has tried to make a connection using incompatible versions of the CTI OS Server and CTI OS CIL.

Syntax

C++: void OnConnectionRejected (Arguments& args)
**ISessionEvents Interface**

**COM:** void OnConnectionRejected (IArguments * args)
**VB:** Session_OnConnectionRejected (ByVal args As CtiosCLIENTLib.IArguments)

### Parameters

**args**

Not currently used, reserved for future use.

**OnCTIOSFailure**

The OnCTIOSFailure event indicates that the CTI Server has fired either a FailureConf or a SystemEvent.

### Syntax

**C++:** void OnCTIOSFailure (Arguments& args)
**COM:** void OnCTIOSFailure (IArguments * args)
**VB:** Session_OnCTIOSFailure (ByVal args As CtiosCLIENTLib.IArguments)

### Parameters

**args**

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FailureCode</td>
<td>INT</td>
<td>A value according to an enumerated value, as shown immediately following this table.</td>
</tr>
<tr>
<td>SystemEventID</td>
<td>INT</td>
<td>Present only if FailureCode equals ServerConnectionStatus. Contains a value according to an enumerated value, as shown immediately following this table.</td>
</tr>
</tbody>
</table>
Following are the enumerated values for Failure Code:

```c
enum enumCTIOS_FailureCode
{
    eDriverOutOfService     = 1,
    eServiceNotSupported    = eDriverOutOfService + 1,
    eOperationNotSupported  = eServiceNotSupported + 1,
    eInvalidPriviledge      = eOperationNotSupported + 1,
    eUnknownRequestID       = eInvalidPriviledge + 1,
    eUnknownEventID         = eUnknownRequestID + 1,
    eUnknownObjectID        = eUnknownEventID + 1,
    eRequiredArgMissing     = eUnknownObjectID + 1,
    eInvalidObjectState     = eRequiredArgMissing + 1,
    eServerConnectionStatus = eInvalidObjectState + 1,
    eInconsistentAgentData  = eServerConnectionStatus + 1
};
```

Following are the enumerated values for SystemEventID:

```c
enum enumCTIOS_SystemEventID
{
    eSysCentralControllerOnline     = 1,
    eSysCentralControllerOffline    = 2,
    eSysPeripheralOnline    = 3,
    eSysPeripheralOffline   = 4,
    eSysTextFYI     = 5,
    eSysPeripheralGatewayOffline    = 6,
    eSysCtiServerOffline    = 7,
    eSysCtiServerOnline     = 8,
    eSysHalfHourChange      = 9,
    eSysInstrumentOutOfService      = 10,
    eSysInstrumentBackInService     = 11,
    eSysCtiServerDriverOnline       = eSysInstrumentBackInService + 1,
    eSysCtiServerDriverOffline      = eSysCtiServerDriverOnline + 1,
    eSysCTIOSServerOffline  = eSysCtiServerDriverOffline + 1,
    eSysCTIOSServerOnline   = eSysCTIOSServerOffline + 1
}
```
Remarks

See the descriptions of the CtiOsEnums.FailureCode and CtiOsEnums.SystemEvent interfaces in the Javadoc for information on Java CIL enumerations.

**OnCurrentAgentReset**

The OnCurrentAgentReset message is generated when the current agent is removed from the session.

**Syntax**

- **C++**: `void OnCurrentAgentReset(Arguments& args)`
- **COM**: `void OnCurrentAgentReset (IArguments * args)`
- **VB**: `session_OnCurrentAgentReset (ByVal args As CtiOsCLIENTLib.IArguments)`

**Parameters**

- **args**

  Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>Unique object ID (if any) of the old current agent that was just removed.</td>
</tr>
</tbody>
</table>

**OnCurrentCallChanged**

The OnCurrentCallChanged message is generated when the current call has changed to another call.

**Syntax**

- **C++**: `void OnCurrentCallChanged(Arguments& args)`
- **COM**: `void OnCurrentCallChanged (IArguments * args)`

VB:     session_OnCurrentCallChanged (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>Unique object ID (if any) of the new current call.</td>
</tr>
</tbody>
</table>

OnFailureEvent

Not supported.

OnGlobalSettingsDownloadConf

The OnGlobalSettingsDownloadConf event is generated as the result of a RequestDesktopSettings method call on the Session (which sends an eGlobalSettingsDownloadRequest event to the server).

A major benefit of using CTI OS is the ability to set client configuration once in the CTI OS Server and then to download this configuration to each CTI OS client desktop. This is accomplished by an application calling the RequestDesktopSettings method on the Session object. The server will then, in turn, send this OnGlobalSettingsDownloadConf event back to the application in response.

The Arguments object passed as a parameter in this event contains the Desktop Settings configuration information. It may contain up to seven elements, each of which has the value of a nested Arguments array. Each of these Arguments arrays is a packed version of the configuration contained in the CTI OS Server, which is explained further in the Cisco ICM Software CTI OS System Manager’s Guide. This section also lists the default values under each key. Custom applications may add values at the lowest level under each key, but may not add new keys. Custom values added in this way will be passed through to the client in this event.
Chapter 6  Event Interfaces and Events

ISessionEvents Interface

It is important to note that some of this information is not always present in the conf event. Items that are not configured on the server will not be present in the Arguments for the conf. To gain an understanding of what is available and how to configure these items, see the following sections in Chapter 4 of the Cisco ICM Software CTI OS System Manager’s Guide.

- MainScreen
- Defining Connection Profiles
- Declaring ECC Variables
- Configuring the Call Appearance Grid
- Automatic Agent Statistics Grid Configuration
- Automatic Skill Group Statistics Grid Configuration

Syntax

C++: void OnGlobalSettingsDownloadConf(Arguments & args)
COM: void OnGlobalSettingsDownloadConf(IArguments * args)
VB:     session_OnGlobalSettingsDownloadConf(ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

An Arguments array containing the desktop settings. The following are the top level elements of this object and their contents.

- CallStatus – Contains an Arguments array, each member of which contains a keyword with the string representation of a valid CallStatus and as a value of a custom defined text string that can be displayed to an end user (for example, used in the CallAppearance ActiveX Control on the CTI OS Agent Desktop or the CTI OS Supervisor Desktop for IPCC Enterprise). The packing of this Arguments array is as follows:
The ISessionEvents Interface

"CallStatus", <Arguments array>

   "y", "Cleared"
   "I", "Dialing"
   ...

- **ECC** – This contains an Arguments array where each key is the name of an ECC variable minus the “user.” prefix (for example, for “user.Variable1” it would be “Variable1” or for “user.Array1” it would be “Array1”). Each value can be one of the following choices.
  - For each scalar ECC variable, the value is “ECC”.
  - For each array ECC variable, the value is “ECCARRAY[n]” where n is the position in the array.
  - These items are not added at setup time and must be added by editing the Windows Registry. The packing of this Arguments array is as follows:

"ECC", <Arguments array>

   "Variable1", "ECC"
   "Array1", "ECCARRAY[1]"
   ...

Cisco ICM Software CTI OS Developer’s Guide Release 6.0(0)
- **Grid** – This contains an Arguments array of up to three elements, each of which are Arguments arrays with the following keys: “AgentStatistics”, “CallAppearance”, and “SkillGroupStatistics”. The Arguments arrays for each of these enable the user to set configuration for the appearance of the columns for each grid, as described in the *Cisco ICM Software CTI OS System Manager’s Guide*. The packing of this Arguments array is as follows:

```plaintext
"Grid", <Arguments array>
  "CallAppearance", <Arguments array>
    "Column", <Arguments array>
      "I", <Arguments array>
        "Type", "Var1"
          "Header", "MyVar1"
    ...
  "AgentStatistics", <Arguments array>
  "SkillGroupStatistics", <Arguments array>
```

- **IPCCSilentMonitor** – This contains an Arguments array, where each member is a configured item described in the *Cisco ICM Software CTI OS System Manager’s Guide*. The packing of this Arguments array is as follows:

```plaintext
"IPCCSilentMonitor", <Arguments array>
  "HeartbeatInterval", 5
    ...
```

- **Login** – This includes a nested Arguments array with a value for each different connection profile, where the keyword is the name for the Connection Profile listed in the System Registry. Information included in each profile is described in the Defining Connection Profiles section of the *Cisco ICM Software CTI OS System Manager’s Guide*. The packing of this Arguments array is as follows:
- **ScreenPreferences** – This includes a nested Arguments array, which includes default members described in the *Cisco ICM Software CTI OS System Manager’s Guide* as well as custom values that an application would like to have configured. The packing of this Arguments array is as follows:

```
"ScreenPreferences", <Arguments array>
  "MainScreen", <Arguments array>
```

- **SoundPreferences** – This includes a nested Arguments array, which includes a setting for each of the sounds, including .wav files to be played, and whether or not each one is mute. It may also include custom name/value pairs for a custom application. The packing of this Arguments array is as follows:

```
"SoundPreferences", <Arguments array>
  "BringToFrontOnCall", 1
  "FlashOnCall", 0
  "CustomName", "CustomValue"
```
This configuration is stored in the Windows System Registry database and many of the values are set when the CTI OS Server Setup is run. Custom configuration may be set at a later time through the Windows Registry Editor.

OnHeartbeat

The OnHeartbeat event is generated when a heartbeat response is received from a CTI OS server. It returns the time of day.

Syntax

C++: void OnHeartbeat(Arguments& args)

COM: void Onheartbeat(IArguments * args)

VB: session_OnHeartbeat (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventTime</td>
<td>INT</td>
<td>Integer value with time of day expressed in milliseconds.</td>
</tr>
</tbody>
</table>
OnMissingHeartbeat

The OnMissingHeartbeat event is generated when an expected heartbeat is not received. It returns the number of consecutive heartbeats missed and time of day. When the number of heartbeats missed equals or exceeds the maximum number of heartbeats allowed (set in the MaxHeartbeats property), an OnConnectionFailed event is generated instead of an OnMissingHeartbeat event and the connection to a new CTI OS server is automatically restarted.

Syntax

C++: void OnMissingHeartbeat(Arguments& args)
COM: void OnMissingHeartbeat (IArguments * args)
VB: session_OnMissingHeartbeat (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventTime</td>
<td>INT</td>
<td>Integer value with time of day expressed in milliseconds.</td>
</tr>
<tr>
<td>ConsecutiveMissedHeartbeats</td>
<td>INT</td>
<td>Integer value with the number of heartbeats missed.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Integer value with the heartbeat interval, in milliseconds.</td>
</tr>
</tbody>
</table>

OnMonitorModeEstablished

The OnMonitorModeEstablished event is generated when Monitor Mode is established or released.
Chapter 6      Event Interfaces and Events

ISessionEvents Interface

Syntax

C++: void OnMonitorModeEstablished(Arguments & args)
COM: void OnMonitorModeEstablished (IArguments * args)
VB:   session_OnMonitorModeEstablished (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIL</td>
<td>STRING</td>
<td>ID of the client’s connection on the server.</td>
</tr>
<tr>
<td>ConnectionID</td>
<td>STRING</td>
<td>ID of the client’s connection on the server.</td>
</tr>
</tbody>
</table>

OnSnapshotDeviceConf

The OnSnapshotDeviceConf confirmation message is fired to the client as part of a snapshot operation. For AgentMode clients, the OnSnapshotDeviceConf will arrive at startup time, after the OnQueryAgentStateConf message. The OnSnapshotDeviceConf indicates the number of calls present at the device, and their UniqueObjectIDs.

Syntax

C++: void OnSnapshotDeviceConf (Arguments & args);
COM: HRESULT OnSnapshotDeviceConf ([in] IArguments * args);
VB:   Session_ OnSnapshotDeviceConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following fields.
The CIL uses this event to rectify the list of calls on a device when logging in after a failover, in case the status of calls on the device changes during the failure period. An example of such a scenario would be an agent talking on a call on a hardphone and a CTI failure occurs. The agent hangs up the call before CTI is recovered. Once CTI and the CIL recover, they use the snapshot to discover that the call it currently has is no longer on the device. CTI then fires an EndCall event to remove the call from its call list.

**OnSnapshotSkillGroupList**

Not supported.
ICallEvents Interface

The Call object fires events on the ICallEvents interface. The following events are published to subscribers of the ICallEvents interface.

**Note**

Many of the parameters that CTI OS receives from the CTI Server are inconsequential to most customer applications. The most important parameters for doing a screenpop are included with the events described in this section. The more inconsequential parameters are suppressed at the CTI OS Server, to minimize network traffic to the clients. However, you can enable the complete set of available event arguments by setting the following registry setting:

```
[HKLM\Cisco Systems\CTIOS\Server\CallObject\MinimizeEventArgs = 0].
```

OnAlternateCallConf

The OnAlternateCallConf event is fired to the client to indicate that an Alternate request was received by the CTI Server

**Syntax**

- **C++:** `void OnAlternateCallConf (Arguments & args);`
- **COM:** `HRESULT OnAlternateCallConf ([in] IArguments * args);`
- **VB:** `Session_OnAlternateCallConf (ByVal args as CTOSSCIENTLIB.IArguments)`

**Parameters**

- **args**

  Arguments array containing the following field.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
</tbody>
</table>
OnAnswerCallConf

The OnAnswerCallConf event is fired to the client to indicate that an Answer request was received by the CTI Server.

Syntax

C++: void OnAnswerCallConf (Arguments & args);
COM: HRESULT OnAnswerCallConf ([in] IArguments * args);
VB: Session_ OnAnswerCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args
Arguments array containing the following field.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
</tbody>
</table>

OnCallBegin

The OnCallBegin event is generated at the first association between a call and the CTI Client. The event passes the call identifier and the initial call context data. The ConnectionCallID identifies the call. This message always precedes any other event messages for that call.

Subsequent changes to the call context data (if any) are forwarded by an OnCallDataUpdate event containing the changed call data.

Note

There can be multiple calls with the same ConnectionCallID value.

Syntax

C++: void OnCallBegin(Arguments & args)
ICallEvents Interface

**COM:**
void OnCallBegin (IArguments * args)

**VB:**
session_OnCallBegin (ByVal args As CtiosCLIENTLib.IArguments)

## Parameters

**args**
Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>INT</td>
<td>The call key created by the ICM. The ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>RouterCallKeySequenceNumber</td>
<td>INT</td>
<td>Together with RouterCallKeyDay and RouterCallKeyCallID fields forms the TaskID.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>INT</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>ANI (optional)</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>DNIS (optional)</td>
<td>STRING</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>UserToUserInfo (Optional)</td>
<td>STRING</td>
<td>The ISDN user-to-user information element. unspecified, up to 131 bytes.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DialedNumber (Optional)</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits (Optional)</td>
<td>STRING</td>
<td>The digits entered by the caller in response to IVR prompting.</td>
</tr>
<tr>
<td>ServiceNumber (Optional)</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID (Optional)</td>
<td>INT</td>
<td>The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID (Optional)</td>
<td>INT</td>
<td>The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority (Optional)</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>CallWrapupData (Optional)</td>
<td>STRING</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>CallVariable1 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>CallVariable10 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

ICallEvents Interface

OnCallEnd

The OnCallEnd event is generated when the association between a call and the CTI Client is dissolved. The OnCallEnd event is the last event received for a Call.

Syntax

C++: void OnCallEnd(Arguments& args)
COM:  void OnCallEnd (IArguments * args)
VB:    session_OnCallEnd (ByVal args As CtiosCLIENTLib.IArguments)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallStatus (optional)</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ECC (optional)</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains all of the Expanded Call Context variables in use; for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ArrayVariable[0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ArrayVariable[1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ArrayVariable[n]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ScalarVariable</td>
</tr>
<tr>
<td>CTIClients (Optional)</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains the information about the number of clients that are using the call object; for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTIClient[1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTIClientSignature</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTIClientTimestamp</td>
</tr>
<tr>
<td>ICMEnterprise</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
<tr>
<td>UniqueID (Optional)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ICMEnterprise UniqueID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>

OnCallDataUpdate

Changes to the call context data will generate an OnCallDataUpdate event. Only the items that have changed will be in the event argument array. The initial call context is provided in the OnCallBegin event.

Syntax

C++:
```cpp
void OnCallDataUpdate(Arguments& args)
```

COM:
```cpp
void OnCallDataUpdate (IArguments * args)
```

VB:
```vbnet
session_OnCallDataUpdate (ByVal args As CtiosCLIENTLib.IArguments)
```

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call’s records in the ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>INT</td>
<td>The call key created by the ICM. The ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>RouterCallKeySequenceNumber</td>
<td>INT</td>
<td>Together with RouterCallKeyDay and RouterCallKeyCallID fields forms the TaskID.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>INT</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>ANI (optional)</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>DNIS (optional)</td>
<td>STRING</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>UserToUserInfo (Optional)</td>
<td>STRING</td>
<td>The ISDN user-to-user information element. unspecified, up to 131 bytes.</td>
</tr>
<tr>
<td>DialedNumber (Optional)</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits (Optional)</td>
<td>STRING</td>
<td>The digits entered by the caller in response to IVR prompting.</td>
</tr>
<tr>
<td>ServiceNumber (Optional)</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID (Optional)</td>
<td>INT</td>
<td>The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
</tbody>
</table>
### ICallEvents Interface

#### CallType
- **Type**: SHORT
- **Description**: The general classification of the call type.

#### UniqueObjectID
- **Type**: STRING
- **Description**: An object ID that uniquely identifies the call object.

#### RouterCallKeyDay
- **Type**: INT
- **Description**: Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call’s records in the ICM database. Only provided for Post-routed and Translation-routed calls.

#### RouterCallKeyCallID
- **Type**: INT
- **Description**: The call key created by the ICM. The ICM resets this counter at midnight.

#### RouterCallKeySequenceNumber
- **Type**: INT
- **Description**: Together with RouterCallKeyDay and RouterCallKeyCallID fields forms the TaskID.

#### ConnectionCallID
- **Type**: INT
- **Description**: The Call ID value assigned to this call by the peripheral or the ICM.

#### ANI (optional)
- **Type**: STRING
- **Description**: The calling line ID of the caller.

#### DNIS (optional)
- **Type**: STRING
- **Description**: The DNIS provided with the call.

#### UserToUserInfo (Optional)
- **Type**: STRING
- **Description**: The ISDN user-to-user information element. unspecified, up to 131 bytes.

#### DialedNumber (Optional)
- **Type**: STRING
- **Description**: The number dialed.

#### CallerEnteredDigits (Optional)
- **Type**: STRING
- **Description**: The digits entered by the caller in response to IVR prompting.

#### ServiceNumber (Optional)
- **Type**: INT
- **Description**: The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.

#### ServiceID (Optional)
- **Type**: INT
- **Description**: The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.
### ICallEvents Interface

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>CallWrapupData</td>
<td>STRING</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>CallVariable1</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CallVariable10</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ECC</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains all of the Expanded Call Context variables in use; for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ArrayVariable[0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ArrayVariable[1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ArrayVariable[n]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ScalarVariable</td>
</tr>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

ICallEvents Interface

The OnCallDelivered event may be generated when the call arrives at the agent’s teleset. Both parties (call connections) receive this event. With this event, the called party’s connection status becomes LCS_ALERTING but the calling party’s connection status remains LCS_INITIATE.

Note
Under certain switches, when a call is made outside of the ACD, this event may not be received. See OnCallReachedNetwork for more detail.

Syntax

C++:  void OnCallDelivered(Arguments& args)
COM:  void OnCallDelivered (IArguments * args)
VB:   session_OnCallDelivered (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTIClients</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains the information about the number of clients that are using the call object; for example: CTIClient[1] CTIClientSignature CTIClientTimestamp</td>
</tr>
<tr>
<td>ICMEnterprise UniqueID</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>
### ICallEvents Interface

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID (Optional)</td>
<td>INT</td>
<td>The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority (Optional)</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call. See Table 3-2.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>
Chapter 6      Event Interfaces and Events

ICallEvents Interface

The OnCallEstablished event may be generated when the call is answered at the agent's teleset. Both parties (call connections) receive this event when the call is answered. With this event, the call status of both parties becomes LCS_CONNECT.

**Note**  
Under certain switches, when a call is made outside of the ACD, this event may not be received. See OnCallReachedNetwork for more detail.

**Syntax**

C++: void OnCallEstablished(Arguments& args)
COM: void OnCallEstablished (IArguments * args)
VB:    session_OnCallEstablished (ByVal args As CtiosCLIENTLib.IArguments)

**Parameters**

args

Arguments array containing the following fields.
### ICallEvents Interface

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID (Optional)</td>
<td>INT</td>
<td>The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority (Optional)</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call. See Table 3-2.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>
OnCallHeld

Placing a call on hold at the agent’s teleset may generate an OnCallHeld event. With this event the connection status becomes LCS_HELD.

Syntax

C++: void OnCallHeld(Arguments& args)
COM: void OnCallHeld (IArguments * args)
VB: session_OnCallHeld (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>
OnCallRetrieved

Resuming a call previously placed on hold at the agent’s teleset may generate an OnCallRetrieved event. With this event the connection status becomes LCS_CONNECT.

Syntax

C++: `void OnCallRetrieved(Arguments& args)`
COM: `void OnCallRetrieved (IArguments * args)`
VB: `session_OnCallRetrieved (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args
Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
</tbody>
</table>

OnCallCleared

An OnCallCleared event is generated when the voice portion of all parties on a call is terminated, normally when the last device disconnects from a call. With this event the connection status becomes LCS_NULL.

Syntax

C++: `void OnCallCleared(Arguments& args)`
COM: `void OnCallCleared (IArguments * args)`
VB: session_OnCallCleared (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>

OnCallConnectionCleared

An OnCallConnectionCleared event is generated when a party drops from a call. With this event the connection status becomes LCS_NULL.

Syntax

C++: void OnCallConnectionCleared(Arguments& args)
COM: void OnCallConnectionCleared (IArguments * args)
VB: session_OnCallConnectionCleared (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.
OnCallOriginated

The initiation of a call from the peripheral may generate an OnCallOriginated event. Only the connection making the call receives this event. With this event the connection status becomes LCS_INITIATE.

Syntax

**C++:**
void OnCallOriginated(Arguments& args)

**COM:**
void OnCallOriginated (IArguments * args)

**VB:**
session_OnCallOriginated (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>
**ICallEvents Interface**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID (Optional)</td>
<td>INT</td>
<td>The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority (Optional)</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
</tbody>
</table>
OnCallFailed

The OnCallFailed event may be generated when a call cannot be completed. With this event the connection status becomes LCS_FAIL. This would most likely happen as a result of a MakeCall or a MakeConsultCall request, but can occur at any other point in the call’s lifetime if the call fails on an ACD. In this case, you should perform any required cleanup prior to arrival of an EndCall event.

Syntax

C++: void OnCallFailed(Arguments& args)
COM: void OnCallFailed (IArguments * args)
VB: session_OnCallFailed (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bitmask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
</tbody>
</table>
OnCallTransferred

The transfer of a call to another destination may generate an OnCallTransferred event. With this event the two connections at the controller’s device end and the status of the connections at the original caller’s device and the consulted device are unchanged.

Syntax

C++: void OnCallTransferred(Arguments & args)
COM: void OnCallTransferred (IArguments * args)
VB: session_OnCallTransferred (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call’s records in the ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RouterCalKeyCallID</td>
<td>INT</td>
<td>The call key created by the ICM. The ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>INT</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>ANI (optional)</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>DNIS (optional)</td>
<td>STRING</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>UserToUserInfo (Optional)</td>
<td>STRING</td>
<td>The ISDN user-to-user information element. unspecified, up to 131 bytes.</td>
</tr>
<tr>
<td>DialedNumber (Optional)</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits (Optional)</td>
<td>STRING</td>
<td>The digits entered by the caller in response to IVR prompting.</td>
</tr>
<tr>
<td>ServiceNumber (Optional)</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID (Optional)</td>
<td>INT</td>
<td>The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
</tbody>
</table>
### ICallEvents Interface

#### Keyword | Type | Description
--- | --- | ---
SkillGroupID (Optional) | INT | The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.

SkillGroupPriority (Optional) | SHORT | The priority of the skill group, or 0 when skill group priority is not applicable or not available.

CallWrapupData (Optional) | STRING | Call-related wrapup data.

CallVariable1 (Optional) | STRING | Call-related variable data.

... | ... | ...

CallVariable10 (Optional) | STRING | Call-related variable data.

CallStatus (Optional) | SHORT | The current status of the call.

ECC (optional) | ARGUMENTS | Arguments array that contains all of the Expanded Call Context variables in use; for example:

user.ArrayVariable[0]
user.ArrayVariable[1]
... 
user.ArrayVariable[n]
user.ScalarVariable

CTIClients (Optional) | ARGUMENTS | Arguments array that contains the information about the number of clients that are using the call object; for example:

CTIClient[1]
CTIClientSignature
CTIClientTimestamp

ICMEnterpriseUniqueID (Optional) | STRING | Required only when the call is pre-routed.
OnCallConferenced

The joining of calls into a conference call or the adding of a new call joining a conference may generate an OnCallConferenced event. With this event, the connections at the controller’s device merge to become one connection with a status of LCS_CONNECT, and the status of the connections at the original caller’s device and at the consulted device remain unchanged.

Syntax

C++: `void OnCallConferenced(Arguments& args)`
COM: `void OnCallConferenced (IArguments * args)`
VB: `session_OnCallConferenced (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

`args`

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>UniqueObjectId</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCalldID field forms the unique 64-bit key for locating this call’s records in the ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>RouterCalKeyCallID</td>
<td>INT</td>
<td>The call key created by the ICM. The ICM resets this counter at midnight.</td>
</tr>
</tbody>
</table>
### ICallEvents Interface

ConnectionCallID | INT | The Call ID value assigned to this call by the peripheral or the ICM.

ANI (optional) | STRING | The calling line ID of the caller.

DNIS (optional) | STRING | The DNIS provided with the call.

UserToUserInfo (Optional) | STRING | The ISDN user-to-user information element. unspecified, up to 131 bytes.

DialedNumber (Optional) | STRING | The number dialed.

CallerEnteredDigits (Optional) | STRING | The digits entered by the caller in response to IVR prompting.

ServiceNumber (Optional) | INT | The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.

ServiceID (Optional) | INT | The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.

SkillGroupNumber (Optional) | INT | The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.

SkillGroupID (Optional) | INT | The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.

SkillGroupPriority (Optional) | SHORT | The priority of the skill group, or 0 when skill group priority is not applicable or not available.
### ICallEvents Interface

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallWrapupData (Optional)</td>
<td>STRING</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>CallVariable1 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>CallVariable10 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallStatus (optional)</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ECC (optional)</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains all of the Expanded Call Context variables in use; for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ArrayVariable[0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ArrayVariable[1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ArrayVariable[n]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ScalarVariable</td>
</tr>
<tr>
<td>CTIClients (Optional)</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains the information about the number of clients that are using the call object; for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTIClient[1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTIClientSignature</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTIClientTimestamp</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>

### OnCallDiverted

The removal of a call from one delivery target and forwarded to a different target may generate an OnCallDiverted event.
ICallEvents Interface

Chapter 6      Event Interfaces and Events

Syntax

**C++:**
void OnCallDiverted(Arguments& args)

**COM:**
void OnCallDiverted (IArguments * args)

**VB:**
session_OnCallDiverted (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>INT</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>INT</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>DivertingDeviceType</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the DivertingDeviceID field.</td>
</tr>
<tr>
<td>CalledDeviceType</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID field.</td>
</tr>
</tbody>
</table>
Chapter 6      Event Interfaces and Events

ICallEvents Interface

OnAgentPrecallEvent

Not supported.

OnAgentPrecallAbortEvent

Not supported.

OnCallServiceInitiatedEvent

The initiation of telecommunications service ("dial tone") at the agent’s teleset may generate an OnCallServiceInitiatedEvent to the CTI Client. However, when the call is made through the software there is no way to detect the equivalent of the phone off hook. Therefore, after a call is made this event is received in sequence along with the OnCallOriginated and OnCallDelivered events. With this event the connection status becomes LCS_INITIATE.

Syntax

C++: void OnCallServiceInitiatedEvent (Arguments& args)
COM:  void OnCallServiceInitiatedEvent (IArguments * args)
VB:   session_OnCallServiceInitiatedEvent (ByVal args As CtiosCLIENTLib.IArguments)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalConnectionState</td>
<td>SHORT</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>SHORT</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>DivertingDeviceID</td>
<td>STRING</td>
<td>The device identifier of the device from which the call was diverted.</td>
</tr>
<tr>
<td>CalledDeviceID</td>
<td>STRING</td>
<td>The device identifier of the device to which the call was diverted.</td>
</tr>
</tbody>
</table>
## Parameters

`args` Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
</tbody>
</table>
OnCallQueuedEvent

The placing of a call in a queue pending the availability of some resource may generate an OnCallQueuedEvent message to the CTI Client. Clients with Client Events Service may receive this message when an outbound call is queued waiting for a trunk or other resource. Clients with All Events Service may also receive this message when inbound calls are queued.

Syntax

C++: void OnCallQueuedEvent (Arguments & args)
COM: void OnCallQueuedEvent (IArguments * args)
VB:session_OnCallQueuedEvent (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection DeviceID</td>
<td>INT</td>
<td>The identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of the connection identifier supplied in the ConnectionDeviceID</td>
</tr>
<tr>
<td>QueuedDeviceID</td>
<td>STRING</td>
<td>The device identifier of the queuing device.</td>
</tr>
<tr>
<td>QueuedDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the QueuedDeviceID.</td>
</tr>
<tr>
<td>CallingDeviceID</td>
<td>STRING</td>
<td>The device identifier of the calling device.</td>
</tr>
<tr>
<td>CallingDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID.</td>
</tr>
</tbody>
</table>
## ICallEvents Interface

### CalledDeviceID
- **Keyword:** CalledDeviceID
- **Type:** STRING
- **Description:** The device identifier of the called device.

### CalledDeviceIDType
- **Keyword:** CalledDeviceIDType
- **Type:** SHORT
- **Description:** Indicates the type of the device identifier supplied in the CalledDeviceID.

### LastRedirectedDeviceID
- **Keyword:** LastRedirectedDeviceID
- **Type:** STRING
- **Description:** The device identifier of the redirecting device.

### LastRedirectedDeviceIDType
- **Keyword:** LastRedirectedDeviceIDType
- **Type:** SHORT
- **Description:** Indicates the type of the device identifier supplied in the LastRedirectDeviceID.

### LocalConnectionState
- **Keyword:** LocalConnectionState
- **Type:** SHORT
- **Description:** The state of the local end of the connection.

### EventCause
- **Keyword:** EventCause
- **Type:** SHORT
- **Description:** Indicates a reason or explanation for the occurrence of the event.

### LineHandle
- **Keyword:** LineHandle
- **Type:** SHORT
- **Description:** Identifies the teleset line being used.

### LineType
- **Keyword:** LineType
- **Type:** SHORT
- **Description:** Indicates the type of the teleset line.

### ServiceID
- **Keyword:** ServiceID
- **Type:** INT
- **Description:** The ICM ServiceID of the service that the call is attributed to.

### ServiceNumber
- **Keyword:** ServiceNumber
- **Type:** INT
- **Description:** The service that the call is attributed to, as known to the peripheral.

### SkillGroupID
- **Keyword:** SkillGroupID
- **Type:** INT
- **Description:** The ICM SkillGroupID of the agent SkillGroup the call is attributed to.

### SkillGroupNumber
- **Keyword:** SkillGroupNumber
- **Type:** INT
- **Description:** The number of an agent SkillGroup queue that the call has been added to, as known to the peripheral.

### SkillGroupPriority
- **Keyword:** SkillGroupPriority
- **Type:** SHORT
- **Description:** The priority of the skill group, or 0 when skill group priority is not applicable or not available.
The explicit removal of a call from the ACD queue may generate an OnCallDequeuedEvent message to the CTI Client.

### Syntax

**C++:**

```cpp
void OnCallDequeuedEvent(Arguments& args)
```

**COM:**

```cpp
void OnCallDequeuedEvent (IArguments * args)
```

**VB:**

```vbnet
session_OnCallDequeuedEvent (ByVal args As CtiosCLIENTLib.IArguments)
```

### Parameters

**args**

Arguments array containing the following fields.

#### Description

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumQueued</td>
<td>SHORT</td>
<td>The number of calls in the queue for this service.</td>
</tr>
<tr>
<td>NumSkillGroups</td>
<td>SHORT</td>
<td>The number of Skill Group queues that the call has queued to, up to a maximum of 20. This value also indicates the number of SkillGroup Number, SkillGroupID and SkillGroup Priority floating fields present in the floating part of the message.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>INT</td>
<td>The identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of the connection identifier supplied in the ConnectionDeviceID.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>SHORT</td>
<td>The state of the local end of the connection.</td>
</tr>
</tbody>
</table>
ICallEvents Interface

OnCallReachedNetworkEvent

The connection of an outbound call to another network may generate an OnCallReachedNetworkEvent. With some switches outside of the ACD, this may be the last event the outbound connection receives. For these switches, you may not assume that when the called party receives and answers the call that the OnCallDelivered and OnCallEstablished events will be received.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventCause</td>
<td>SHORT</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>SHORT</td>
<td>Identifies the teleset line being used.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The ICM ServiceID of the service that the call is attributed to.</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The ICM SkillGroupID of the agent SkillGroup the call is attributed to.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The number of an agent SkillGroup queue that the call has been added to, as known to the peripheral.</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>NumQueued</td>
<td>SHORT</td>
<td>The number of calls in the queue for this service.</td>
</tr>
<tr>
<td>NumSkillGroups</td>
<td>SHORT</td>
<td>The number of Skill Group queues that the call has queued to, up to a maximum of 20. This value also indicates the number of SkillGroupNumber, SkillGroupID and SkillGroupPriority floating fields present in the floating part of the message.</td>
</tr>
</tbody>
</table>
Chapter 6      Event Interfaces and Events

ICallEvents Interface

Syntax

C++: void OnCallReachedNetworkEvent (Arguments & args)
COM: void OnCallReachedNetworkEvent (IArguments * args)
VB:    session_OnCallReachedNetworkEvent (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionDeviceID</td>
<td>STRING</td>
<td>The identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of the connection identifier supplied in the ConnectionDeviceID.</td>
</tr>
<tr>
<td>TrunkUsedDeviceID</td>
<td>STRING</td>
<td>The device identifier of the selected trunk.</td>
</tr>
<tr>
<td>TrunkUsedDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the TrunkUsedDeviceID.</td>
</tr>
<tr>
<td>CalledDeviceID</td>
<td>STRING</td>
<td>The device identifier of the called device.</td>
</tr>
<tr>
<td>CalledDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>SHORT</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>SHORT</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>SHORT</td>
<td>Identifies the teleset line being used.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>TrunkNumber (optional)</td>
<td>INT</td>
<td>The number representing a trunk.</td>
</tr>
<tr>
<td>TrunkGroupNumber (optional)</td>
<td>INT</td>
<td>The number representing a trunk group.</td>
</tr>
</tbody>
</table>
**OnCallStartRecordingConf**

The OnCallStartRecordingConf event is fired to the client to indicate that a StartRecord request was received by the CTI Server.

**Syntax**

```cpp
void OnCallStartRecordingConf (Arguments & args);
```

```com
HRESULT OnCallStartRecordingConf ([in] IArguments * args);
```

```vb
Session_.OnCallStartRecordingConf (ByVal args as CTIOSCLIENTLIB.IArguments)
```

**Parameters**

*args*

Arguments array containing the following field.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
</tbody>
</table>

**OnCallStopRecordingConf**

The OnCallStopRecordingConf event is fired to the client to indicate that a StopRecord request was received by the CTI Server.

**Syntax**

```cpp
void OnCallStopRecordingConf (Arguments & args);
```

```com
HRESULT OnCallStopRecordingConf ([in] IArguments * args);
```

```vb
Session_.OnCallStopRecordingConf (ByVal args as CTIOSCLIENTLIB.IArguments)
```
Chapter 6  Event Interfaces and Events

ICallEvents Interface

Parameters

args

Arguments array containing the following field.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
</tbody>
</table>

OnClearCallConf

The OnClearCallConf event is fired to the client to indicate that a Clear request was received by the CTIServer.

Syntax

C++: void OnClearCallConf (Arguments & args);

COM: HRESULT OnClearCallConf ([in] IArguments * args);

VB: Session_ OnClearCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following field.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
</tbody>
</table>

OnClearConnectionConf

The OnClearConnectionConf event is fired to the client to indicate that a ClearConnection request was received by the CTIServer.
ICallEvents Interface

Syntax

C++: void OnClearConnectionConf (Arguments & args);
COM: HRESULT OnClearConnectionConf ([in] IArguments * args);
VB: Session_OnClearConnectionConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following field.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
</tbody>
</table>

OnConferenceCallConf

The OnConferenceCallConf event is fired to the client to indicate that a ConferenceCall or SingleStepConferenceCall request was received by the CTIServer.

Syntax

C++: void OnConferenceCallConf (Arguments & args);
COM: HRESULT OnConferenceCallConf ([in] IArguments * args);
VB: Session_OnConferenceCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following field.
OnControlFailureConf

The OnControlFailureConf event is generated when a request to the peripheral (the ACD) fails.

Syntax

C++: void OnControlFailureConf(Arguments& args)
COM:  void OnControlFailureConf(IArguments * args)
VB:   subroutine OnControlFailureConf (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>Peripheral ID.</td>
</tr>
<tr>
<td>FailureCode</td>
<td>SHORT</td>
<td>Code ID.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID that represents a specific client.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
</tbody>
</table>
The OnConsultationCallConf event is fired to the client to indicate that a MakeConsultCall request was received by the CTIServer.

### Syntax

**C++:**
```c++
void OnConsultationCallConf (Arguments & args);
```

**COM:**
```c
HRESULT OnConsultationCallConf ([in] IArguments * args);
```

**VB:**
```vbnet
Session_OnConsultationCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)
```

### Parameters

**args**

Arguments array containing the following field.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
</tbody>
</table>
OnHoldCallConf

The OnHoldCallConf event is fired to the client to indicate that a Hold request was received by the CTIServer.

Syntax

C++: void OnHoldCallConf (Arguments & args);
COM: HRESULT OnHoldCallConf ([in] IArguments * args);
VB: Session. OnHoldCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following field.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
</tbody>
</table>

OnMakePredictiveCallConf

Not supported.

OnReconnectCallConf

The OnReconnectCallConf event is fired to the client to indicate that a Reconnect request was received by the CTIServer.

Syntax

C++: void OnReconnectCallConf (Arguments & args);
COM: HRESULT OnReconnectCallConf ([in] IArguments * args);
VB: Session. OnReconnectCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)
Parameters

args

Arguments array containing the following field.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
</tbody>
</table>

OnReleaseCallConf

Not supported.

OnRetrieveCallConf

The OnRetrieveCallConf event is fired to the client to indicate that a RetrieveCall request was received by the CTIServer.

Syntax

C++: void OnRetrieveCallConf (Arguments & args);
COM: HRESULT OnRetrieveCallConf ([in] IArguments * args);
VB: Session_ OnRetrieveCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following field.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
</tbody>
</table>
OnSendDTMFConf

The OnSendDTMFConf event is fired to the client to indicate that a SendDTMF request was received by the CTIServer.

Syntax

C++:  void OnSendDTMFConf (Arguments & args);
COM: HRESULT OnSendDTMFConf ([in] IArguments * args);
VB: Session_ OnSendDTMFConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args
Not used; reserved for future use.

OnSnapshotCallConf

The OnSnapshotCallConf event is generated when a SnapshotCall request for a specific call is successful. It will contain all the information known about the specific connection at that point in time.

Syntax

C++:  void OnSnapshotCallConf (Arguments& args)
COM:  void OnSnapshotCallConf (IArguments * args)
VB:    session_OnSnapshotCallConf (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.
### ICallEvents Interface

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>INT</td>
<td>The call key created by the ICM. The ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>NumNamedVariables</td>
<td>SHORT</td>
<td>Number of Named variables.</td>
</tr>
<tr>
<td>NumNamedArrays</td>
<td>SHORT</td>
<td>Number of Named Arrays.</td>
</tr>
<tr>
<td>NumCallDevices</td>
<td>SHORT</td>
<td>Number of devices associated with the call.</td>
</tr>
<tr>
<td>CalledDeviceID</td>
<td>STRING</td>
<td>The device identifier of the called device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>INT</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ANI</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>UserToUserInfo</td>
<td>STRING</td>
<td>The ISDN user-to-user information element associated with the call.</td>
</tr>
<tr>
<td>DNIS</td>
<td>STRING</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits</td>
<td>STRING</td>
<td>The digits entered by the caller in response to IVR prompting.</td>
</tr>
</tbody>
</table>
If the MinimizeEventArgs registry entry is set to 0, the SnapshotCallConf event contains the following additional fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallWrapupData</td>
<td>STRING</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>CallVariable1 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>CallVariable10 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CustomerPhoneNumber</td>
<td>STRING</td>
<td>The customer phone number associated with the call.</td>
</tr>
<tr>
<td>CustomerAccountNumber</td>
<td>STRING</td>
<td>The customer account number associated with the call.</td>
</tr>
<tr>
<td>ECC</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains all of the Expanded Call Context variables in use; for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ArrayVariable[0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ArrayVariable[1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ArrayVariable[n]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user.ScalarVariable</td>
</tr>
<tr>
<td>CTIClients (Optional)</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains the information about the number of clients that are using the call object; for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTIClient[1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTIClientSignature</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTIClientTimestamp</td>
</tr>
</tbody>
</table>
OnTransferCallConf

The OnTransferCallConf event is fired to the client to indicate that a TransferCall or SingleStepTransferCall request was received by the CTIServer.

Syntax

**C++:** void OnTransferCallConf (Arguments & args);

**COM:** HRESULT OnTransferCallConf ([in] IArguments * args);
IAgentEvents Interface

The Agent object fires events on the IAgentEvents interface. The following events are published to subscribers of the IAgentEvents interface.

OnAgentStateChange

The OnAgentStateChange event is generated when the agent state at the ACD changes. This may be as a response to a Login, Logout or SetAgentState request.

Syntax

```cpp
void OnAgentStateChange(Arguments& args)
```

```vbnet
session_OnAgentStateChange (ByVal args As CtiosCLIENTLib.IArguments)
```

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
</tbody>
</table>
### IAgentEvents Interface

#### Keyword | Type | Description
--- | --- | ---
PeripheralID | INT | The ICM PeripheralID of the ACD where the agent state change occurred.
PeripheralType | SHORT | The type of the peripheral.
AgentState | SHORT | One of the values in Table 6-1 representing the current overall state of the associated agent.
SkillGroupNumber | INT | The number of the agent SkillGroup affected by the state change, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupID | INT | The ICM SkillGroupID of the agent SkillGroup affected by the state change. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
StateDuration | INT | The number of seconds since the agent entered this state (typically 0).
SkillGroupPriority | SHORT | The priority of the skill group, or 0 when skill group priority is not applicable or not available.
EventReasonCode | SHORT | A peripheral-specific code indicating the reason for the state change.
SkillGroupState | SHORT | Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.
AgentID | STRING | The agent’s ACD login ID.
AgentExtension | STRING | The agent’s ACD teleset extension.
CTIClientSignature (Optional) | STRING | The Client Signature of the CTI Client that is associated with this agent.
Enablement Mask | | Contains the bit-mask that specifies what buttons can be enabled or disabled when the agent is on this state.
UniqueObjectID | STRING | A unique object ID for the agent object.
### IAgentEvents Interface

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>The agent’s ACD instrument number.</td>
</tr>
<tr>
<td>AgentAvailability</td>
<td>INT</td>
<td>An agent is <em>available</em> to work on a task in this Media Routing Domain if the agent meets all of these conditions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The agent is routable for this Media Routing Domain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The agent is not in Not Ready state for skill groups in other Media Routing Domain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The agent is <em>temp routable</em>, meaning that the agent is not in Reserved, Active, Work-Ready, or Work-Not Ready state on a non-interruptible task in another Media Routing Domain.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The agent has not reached the maximum task limit for this Media Routing Domain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An available agent is eligible to be assigned a task. Who can assign a task to the agent is determined by whether or not the agent is Routable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An agent is <em>ICMAvailable</em> in MRD X if he is available in X and Routable with respect to X.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An agent is <em>ApplicationAvailable</em> in MRD X if he is available in X and not Routable with respect to X. Otherwise an agent is NotAvailable in MRD X.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOT AVAILABLE = 0,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICM AVAILABLE = 1,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>APPLICATION AVAILABLE=2</td>
</tr>
</tbody>
</table>
The `OnAgentStatistics` event is fired to the client to indicate that a request to enable agent statistics (via the `EnableAgentStatistics` method) was received by the CTIServer. The event is received at intervals determined by a registry setting on the server (HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems\Ctios\Server\Agent\PollingIntervalSec), unless the value of the registry setting HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems\Ctios\Server\Agent\PollingIntervalSec).

### Table 6-1 AgentState Values

<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Description</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eLogin</td>
<td>The agent has logged on to the ACD. It does not necessarily indicate that the agent is ready to accept calls.</td>
<td>0</td>
</tr>
<tr>
<td>eLogout</td>
<td>The agent has logged out of the ACD and cannot accept any additional calls.</td>
<td>1</td>
</tr>
<tr>
<td>eNotReady</td>
<td>The agent is unavailable for any call work.</td>
<td>2</td>
</tr>
<tr>
<td>eAvailable</td>
<td>The agent is ready to accept a call.</td>
<td>3</td>
</tr>
<tr>
<td>eTalking</td>
<td>The agent is currently talking on a call (inbound, outbound, or inside).</td>
<td>4</td>
</tr>
<tr>
<td>eWorkNotReady</td>
<td>The agent is performing after call work, but will not be ready to receive a call when completed.</td>
<td>5</td>
</tr>
<tr>
<td>eWorkReady</td>
<td>The agent is performing after call work, and will be ready to receive a call when completed.</td>
<td>6</td>
</tr>
<tr>
<td>eBusyOther</td>
<td>The agent is busy performing a task associated with another active SkillGroup.</td>
<td>7</td>
</tr>
<tr>
<td>eReserved</td>
<td>The agent is reserved for a call that will arrive at the ACD shortly.</td>
<td>8</td>
</tr>
<tr>
<td>eUnknown</td>
<td>The agent state is currently unknown.</td>
<td>9</td>
</tr>
<tr>
<td>eHold</td>
<td>The agent currently has all calls on hold.</td>
<td>10</td>
</tr>
</tbody>
</table>

The `OnAgentStatistics` event is fired to the client to indicate that a request to enable agent statistics (via the `EnableAgentStatistics` method) was received by the CTIServer. The event is received at intervals determined by a registry setting on the server (HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems\Ctios\Server\Agent\PollingIntervalSec), unless the value of the registry setting HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems\Ctios\Server\Agent\PollingIntervalSec).
PollForAgentStatesAtEndCall is 1. If PollForAgentStatesAtEndCall is set to 1, the OnAgentStatistics event is received only at the time of an EndCall event.

The table under Parameters details all the agent statistics that could be received. To optimize bandwidth, the default configuration on the server is set to minimize the agent statistics sent. Only the statistics that the Agent Statistics grid is configured for are sent to the client. The registry key that determines this is: (HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems\Ctios\EnterpriseDesktopSettings\All\Desktops\Grid\AgentStatistics\Columns\Number\DisableStatsMinimization).

**Syntax**

**C++**:

```c++
void OnAgentStatistics (Arguments & args);
```

**COM**:

```c++
HRESULT OnAgentStatistics ([in] IArguments * args);
```

**VB**:

```vbnet
Session_ OnAgentStatistics (ByVal args as CTIOSCLIENTLIB.IArguments)
```

**Parameters**

`args`

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the agent is located.</td>
<td>INT</td>
</tr>
<tr>
<td>AvailTimeSession</td>
<td>Total time, in seconds, the agent was in the Available state for any skill group.</td>
<td>INT</td>
</tr>
<tr>
<td>LoggedOnTimeSession</td>
<td>Total time, in seconds, the agent has been logged on.</td>
<td>INT</td>
</tr>
<tr>
<td>NotReadyTimeSession</td>
<td>Total time, in seconds, the agent was in the Not Ready state for all skill groups.</td>
<td>INT</td>
</tr>
<tr>
<td>AgentOutCallsSession</td>
<td>Total number of completed outbound ACD calls made by agent.</td>
<td>INT</td>
</tr>
<tr>
<td>Event Name</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>AgentOutCallsTalkTime Session</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>AgentOutCallsTime Session</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>AgentOutCallsHeld Session</td>
<td>The total number of completed outbound ACD calls the agent has placed on hold at least once.</td>
<td>INT</td>
</tr>
<tr>
<td>AgentOutCallsHeldTime Session</td>
<td>Total number of seconds outbound ACD calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>HandledCallsSession</td>
<td>The number of inbound ACD calls handled by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>HandledCallsTalkTime Session</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by the agent. Includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>HandledCallsAfterCall TimeSession</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>HandledCallsTime Session</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by the agent. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>Event Name</td>
<td>Description</td>
<td>Data Type</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>IncomingCallsHeld Session</td>
<td>The total number of completed inbound ACD calls the agent placed on hold at least once.</td>
<td>INT</td>
</tr>
<tr>
<td>IncomingCallsHeldTime Session</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>InternalCallsSession</td>
<td>Number of internal calls initiated by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>InternalCallsTime Session</td>
<td>Number of seconds spent on internal calls initiated by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>InternalCallsRcvdSession</td>
<td>Number of internal calls received by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>InternalCallsRcvdTime Session</td>
<td>Number of seconds spent on internal calls received by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>InternalCallsHeld Session</td>
<td>The total number of internal calls the agent placed on hold at least once.</td>
<td>INT</td>
</tr>
<tr>
<td>InternalCallsHeldTime Session</td>
<td>Total number of seconds completed internal calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>AutoOutCallsSession</td>
<td>Total number of AutoOut (predictive) calls completed by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>AutoOutCallsTalkTime Session</td>
<td>Total talk time, in seconds, of AutoOut (predictive) calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>Event Name</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>AutoOutCallsTime</td>
<td>Total handle time, in seconds, for AutoOut (predictive) calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>Session</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoOutCallsHeld</td>
<td>The total number of completed AutoOut (predictive) calls the agent has placed on hold at least once.</td>
<td>INT</td>
</tr>
<tr>
<td>Session</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoOutCallsHeldTime</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>Session</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PreviewCallsSession</td>
<td>Total number of outbound Preview calls completed by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>PreviewCallsTalkTime</td>
<td>Total talk time, in seconds, of outbound Preview calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>Session</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PreviewCallsTime</td>
<td>Total handle time, in seconds, outbound Preview calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>Session</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PreviewCallsHeld</td>
<td>The total number of completed outbound Preview calls the agent has placed on hold at least once.</td>
<td>INT</td>
</tr>
<tr>
<td>Session</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PreviewCallsHeldTime</td>
<td>Total number of seconds outbound Preview calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>Session</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### IAgentEvents Interface

<table>
<thead>
<tr>
<th>Event Description</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationCallsSession</td>
<td>Total number of agent reservation calls completed by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>ReservationCallsTalkTimeSession</td>
<td>Total talk time, in seconds, of agent reservation calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>ReservationCallsTimeSession</td>
<td>Total handle time, in seconds, agent reservation calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>ReservationCallsHeldSession</td>
<td>The total number of completed agent reservation calls the agent has placed on hold at least once.</td>
<td>INT</td>
</tr>
<tr>
<td>ReservationCallsHeldTimeSession</td>
<td>Total number of seconds agent reservation calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>BargeInCallsSession</td>
<td>Total number of supervisor call barge-ins completed.</td>
<td>INT</td>
</tr>
<tr>
<td>InterceptCallsSession</td>
<td>Total number of supervisor call intercepts completed.</td>
<td>INT</td>
</tr>
<tr>
<td>MonitorCallsSession</td>
<td>Total number of supervisor call monitors completed.</td>
<td>INT</td>
</tr>
<tr>
<td>WhisperCallsSession</td>
<td>Total number of supervisor whisper calls completed.</td>
<td>INT</td>
</tr>
<tr>
<td>EmergencyCallsSession</td>
<td>Total number of emergency calls.</td>
<td>INT</td>
</tr>
<tr>
<td>AvailTimeToday</td>
<td>Total time, in seconds, the agent was in the Available state for any skill group.</td>
<td>INT</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>LoggedOnTimeToday</td>
<td>Total time, in seconds, the agent has been logged on.</td>
<td>INT</td>
</tr>
<tr>
<td>NotReadyTimeToday</td>
<td>Total time, in seconds, the agent was in the Not Ready state for all skill groups.</td>
<td>INT</td>
</tr>
<tr>
<td>AgentOutCallsToday</td>
<td>Total number of completed outbound ACD calls made by agent.</td>
<td>INT</td>
</tr>
<tr>
<td>AgentOutCallsTalkTimeToday</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>AgentOutCallsTimeToday</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>AgentOutCallsHeldToday</td>
<td>The total number of completed outbound ACD calls the agent has placed on hold at least once.</td>
<td>INT</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeToday</td>
<td>Total number of seconds outbound ACD calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>HandledCallsToday</td>
<td>The number of inbound ACD calls handled by the agent.</td>
<td>INT</td>
</tr>
</tbody>
</table>
### IAgentEvents Interface

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandledCallsTalkTimeToday</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by the agent. Includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>HandledCallsAfterCallTimeToday</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>HandledCallsTimeToday</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by the agent. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>IncomingCallsHeldToday</td>
<td>The total number of completed inbound ACD calls the agent placed on hold at least once.</td>
<td>INT</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeToday</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>InternalCallsToday</td>
<td>Number of internal calls initiated by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>InternalCallsTimeToday</td>
<td>Number of seconds spent on internal calls initiated by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>InternalCallsRcvdToday</td>
<td>Number of internal calls received by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeToday</td>
<td>Number of seconds spent on internal calls received by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>InternalCallsHeldToday</td>
<td>The total number of internal calls the agent placed on hold at least once.</td>
<td>INT</td>
</tr>
<tr>
<td>InternalCallsHeldTimeToday</td>
<td>Total number of seconds completed internal calls were placed on hold.</td>
<td>INT</td>
</tr>
</tbody>
</table>
## IAgentEvents Interface

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoOutCallsToday</td>
<td>Total number of AutoOut (predictive) calls completed by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>AutoOutCallsTalkTimeToday</td>
<td>Total talk time, in seconds, of AutoOut (predictive) calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>AutoOutCallsTimeToday</td>
<td>Total handle time, in seconds, for AutoOut (predictive) calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>AutoOutCallsHeldToday</td>
<td>The total number of completed AutoOut (predictive) calls the agent has placed on hold at least once.</td>
<td>INT</td>
</tr>
<tr>
<td>AutoOutCallsHeldTimeToday</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>PreviewCallsToday</td>
<td>Total number of outbound Preview calls completed by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>Event Description</td>
<td>Description</td>
<td>Data Type</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>PreviewCallsTalkTimeToday</td>
<td>Total talk time, in seconds, of outbound Preview calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>PreviewCallsTimeToday</td>
<td>Total handle time, in seconds, outbound Preview calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>PreviewCallsHeldToday</td>
<td>The total number of completed outbound Preview calls the agent has placed on hold at least once.</td>
<td>INT</td>
</tr>
<tr>
<td>PreviewCallsHeldTimeToday</td>
<td>Total number of seconds outbound Preview calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>ReservationCallsToday</td>
<td>Total number of agent reservation calls completed by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>ReservationCallsTalkTimeToday</td>
<td>Total talk time, in seconds, of agent reservation calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
</tbody>
</table>
### OnChatMessage

The `OnChatMessage` event is generated when an asynchronous text message from another user (agent) is received.

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationCallsTimeToday</td>
<td>Total handle time, in seconds, agent reservation calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
<td>INT</td>
</tr>
<tr>
<td>ReservationCallsHeldToday</td>
<td>The total number of completed agent reservation calls the agent has placed on hold at least once.</td>
<td>INT</td>
</tr>
<tr>
<td>ReservationCallsHeldTimeToday</td>
<td>Total number of seconds agent reservation calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>BargeInCallsToday</td>
<td>Total number of supervisor call barge-ins completed.</td>
<td>INT</td>
</tr>
<tr>
<td>InterceptCallsToday</td>
<td>Total number of supervisor call intercepts completed.</td>
<td>INT</td>
</tr>
<tr>
<td>MonitorCallsToday</td>
<td>Total number of supervisor call monitors completed.</td>
<td>INT</td>
</tr>
<tr>
<td>WhisperCallsToday</td>
<td>Total number of supervisor whisper calls completed.</td>
<td>INT</td>
</tr>
<tr>
<td>EmergencyCallsToday</td>
<td>Total number of emergency calls.</td>
<td>INT</td>
</tr>
<tr>
<td>AgentExtension <em>(required)</em></td>
<td>The agent’s ACD teleset extension.</td>
<td>STRING</td>
</tr>
<tr>
<td>AgentID <em>(required)</em></td>
<td>The agent’s ACD login ID.</td>
<td>STRING</td>
</tr>
<tr>
<td>AgentInstrument <em>(required)</em></td>
<td>The agent’s ACD instrument number.</td>
<td>STRING</td>
</tr>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

IAgentEvents Interface

Syntax

C++: void OnChatMessage(Arguments& args)
COM: void OnChatMessage (IArguments * args)
VB: session_OnChatMessage (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution</td>
<td>STRING</td>
<td>Currently the only supported value is “agent&quot;.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>The AgentID of the message target.</td>
</tr>
<tr>
<td>Target</td>
<td>STRING</td>
<td>The AgentID of the message target.</td>
</tr>
<tr>
<td>Message</td>
<td>STRING</td>
<td>The text message provided by the sender.</td>
</tr>
<tr>
<td>Source</td>
<td>STRING</td>
<td>The AgentID of the message sender.</td>
</tr>
</tbody>
</table>

OnControlFailureConf

The OnControlFailureConf event is generated when the previously issued request, identified by the InvokeID field failed. It is sent in place of the corresponding confirmation message for that request.

Syntax

C++: void OnControlFailureConf(Arguments& args)
COM: void OnControlFailureConf (IArguments * args)
VB: session_OnControlFailureConf (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.
### IAgentEvents Interface

**Keyword** | **Type** | **Description**
--- | --- | ---
InvokeID | INT | InvokeID of the request that failed
FailureCode | SHORT | A value specifying the reason that the request failed. See Table 6-2 for a list of the Control Failure Codes.
AgentID | STRING | Agent ID that represents a specific client.
UniqueObjectID | STRING | An object ID that uniquely identifies the call object.
MessageType | INT | Contains the CTI OS Command Request ID that failed to execute. The message types that can be included in this parameter are those to used to control Call, Agent State and Supervisor actions. Refer to Appendix A, “CTI OS Keywords and Enumerated Types” for a complete list.
ErrorMessage | STRING | String text containing the description of the failure.

#### Table 6-2  Control Failure Codes

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTI_NO_ERROR</td>
<td>No error occurred.</td>
<td>0</td>
</tr>
<tr>
<td>E_CTI_INVALID_VERSION</td>
<td>The CTI Server does not support the protocol version number requested by the CTI client.</td>
<td>1</td>
</tr>
<tr>
<td>E_CTI_INVALID_MESSAGE_TYPE</td>
<td>A message with an invalid message type field was received.</td>
<td>2</td>
</tr>
<tr>
<td>E_CTI_INVALID_FIELD_TAG</td>
<td>A message with an invalid floating field tag was received.</td>
<td>3</td>
</tr>
<tr>
<td>E_CTI_SESSION_NOT_OPEN</td>
<td>No session is currently open on the connection.</td>
<td>4</td>
</tr>
</tbody>
</table>
Table 6-2  Control Failure Codes (continued)

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTI_SESSION_ALREADY_OPEN</td>
<td>A session is already open on the connection.</td>
<td>5</td>
</tr>
<tr>
<td>E_CTI_REQUIRED_DATA_MISSING</td>
<td>The request did not include one or more floating items that are required.</td>
<td>6</td>
</tr>
<tr>
<td>E_CTI_INVALID_PERIPHERAL_ID</td>
<td>A message with an invalid PeripheralID value was received.</td>
<td>7</td>
</tr>
<tr>
<td>E_CTI_INVALID_AGENT_DATA</td>
<td>The provided agent data item(s) are invalid.</td>
<td>8</td>
</tr>
<tr>
<td>E_CTI_AGENT_NOT_LOGGED_ON</td>
<td>The indicated agent(s) are invalid.</td>
<td>9</td>
</tr>
<tr>
<td>E_CTI_DEVICE_IN_USE</td>
<td>The indicated agent teleset is already associated with a different CTI client.</td>
<td>10</td>
</tr>
<tr>
<td>E_CTI_NEW_SESSION_OPENED</td>
<td>This session is being terminated due to a new session open request from the client.</td>
<td>11</td>
</tr>
<tr>
<td>E_CTI_FUNCTION_NOT_AVAILABLE</td>
<td>A request message was received for a function or service that was not granted to the client.</td>
<td>12</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALLID</td>
<td>A request message was received with an invalid CallID value.</td>
<td>13</td>
</tr>
<tr>
<td>E_CTI_PROTECTED_VARIABLE</td>
<td>The CTI client may not update the requested variable.</td>
<td>14</td>
</tr>
<tr>
<td>E_CTI_CTI_SERVER_OFFLINE</td>
<td>The CTI Server is not able to function normally. The CTI client should close the session upon receipt of this error.</td>
<td>15</td>
</tr>
<tr>
<td>E_CTI_TIMEOUT</td>
<td>The CTI Server failed to respond to a request message within the time-out period, or no messages have been received from the CTI client within the IdleTimeout period.</td>
<td>16</td>
</tr>
<tr>
<td>E_CTI_UNSPECIFIED_FAILURE</td>
<td>An unspecified error occurred.</td>
<td>17</td>
</tr>
</tbody>
</table>
Table 6-2  Control Failure Codes (continued)

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTI_INVALID_TIMEOUT</td>
<td>The IdleTimeout field contains a value that is less than 20 seconds (4 times the minimum heartbeat interval of 5 seconds).</td>
<td>18</td>
</tr>
<tr>
<td>E_CTI_INVALID_SERVICE_MASK</td>
<td>The ServicesRequested field has unused bits set. All unused bit positions must be zero.</td>
<td>19</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_MSG_MASK</td>
<td>The CallMsgMask field has unused bits set. All unused bit positions must be zero.</td>
<td>20</td>
</tr>
<tr>
<td>E_CTI_INVALID_AGENT_STATE_MASK</td>
<td>The AgentStateMask field has unused bits set. All unused bit positions must be zero.</td>
<td>21</td>
</tr>
<tr>
<td>E_CTI_INVALID_RESERVED_FIELD</td>
<td>A Reserved field has a non-zero value.</td>
<td>22</td>
</tr>
<tr>
<td>E_CTI_INVALID_FIELD_LENGTH</td>
<td>A floating field exceeds the allowable length for that field type.</td>
<td>23</td>
</tr>
<tr>
<td>E_CTI_INVALID_DIGITS</td>
<td>A STRING field contains characters that are not digits (“0” through “9”).</td>
<td>24</td>
</tr>
<tr>
<td>E_CTI_BAD_MESSAGE_FORMAT</td>
<td>The message is improperly constructed. This may be caused by omitted or incorrectly sized fixed message fields.</td>
<td>25</td>
</tr>
<tr>
<td>E_CTI_INVALID_TAG_FOR_MSG_TYPE</td>
<td>A floating field tag is present that specifies a field that does not belong in this message type.</td>
<td>26</td>
</tr>
<tr>
<td>E_CTI_INVALID_DEVICE_ID_TYPE</td>
<td>A DeviceIDType field contains an invalid value.</td>
<td>27</td>
</tr>
<tr>
<td>E_CTI_INVALID_LCL_CONN_STATE</td>
<td>A LocalConnectionState field contains an invalid value.</td>
<td>28</td>
</tr>
<tr>
<td>E_CTI_INVALID_EVENT_CAUSE</td>
<td>An EventCause field contains an invalid value.</td>
<td>29</td>
</tr>
<tr>
<td>E_CTI_INVALID_NUM_PARTIES</td>
<td>The NumParties field contains a value that exceeds the maximum (16).</td>
<td>30</td>
</tr>
</tbody>
</table>
Table 6-2  Control Failure Codes (continued)

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTI_INVALID_SYS_EVENT_ID</td>
<td>The SystemEventID field contains an invalid value.</td>
<td>31</td>
</tr>
<tr>
<td>E_CTI_INCONSISTENT_AGENT_DATA</td>
<td>The provided agent extension, agent id, and/or agent instrument values are inconsistent with each other.</td>
<td>32</td>
</tr>
<tr>
<td>E_CTI_INVALID_CONNECTION_ID_TYPE</td>
<td>A ConnectionDeviceIDType field contains an invalid value.</td>
<td>33</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_TYPE</td>
<td>The CallType field contains an invalid value.</td>
<td>34</td>
</tr>
<tr>
<td>E_CTI_NOT_CALL_PARTY</td>
<td>A CallDataUpdate or Release Call request specified a call that the client is not a party to.</td>
<td>35</td>
</tr>
<tr>
<td>E_CTI_INVALID_PASSWORD</td>
<td>The ClientID and Client Password provided in an OPEN_REQ message is incorrect.</td>
<td>36</td>
</tr>
<tr>
<td>E_CTI_CLIENT_DISCONNECTED</td>
<td>The client TCP/IP connection was disconnected without a CLOSE_REQ.</td>
<td>37</td>
</tr>
<tr>
<td>E_CTI_INVALID_OBJECT_STATE</td>
<td>An invalid object state value was provided.</td>
<td>38</td>
</tr>
<tr>
<td>E_CTI_INVALID_NUM_SKILL_GROUPS</td>
<td>An invalid NumSkillGroups value was provided.</td>
<td>39</td>
</tr>
<tr>
<td>E_CTI_INVALID_NUM_LINES</td>
<td>An invalid NumLines value was provided.</td>
<td>40</td>
</tr>
<tr>
<td>E_CTI_INVALID_LINE_TYPE</td>
<td>An invalid LineType value was provided.</td>
<td>41</td>
</tr>
<tr>
<td>E_CTI_INVALID_ALLOCATION_STATE</td>
<td>An invalid AllocationState value was provided.</td>
<td>42</td>
</tr>
<tr>
<td>E_CTI_INVALID_ANSWERING_MACHINE</td>
<td>An invalid AnsweringMachine value was provided.</td>
<td>43</td>
</tr>
</tbody>
</table>
### Table 6-2  Control Failure Codes (continued)

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTI_INVALID_CALL_MANNER_TYPE</td>
<td>An invalid CallMannerType value was provided.</td>
<td>44</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_PLACEMENT_TYPE</td>
<td>An invalid CallPlacementType value was provided.</td>
<td>45</td>
</tr>
<tr>
<td>E_CTI INVALID CONSULT_TYPE</td>
<td>An invalid ConsultType value was provided.</td>
<td>46</td>
</tr>
<tr>
<td>E_CTI_INVALID_FACILITY_TYPE</td>
<td>An invalid FacilityType value was provided.</td>
<td>47</td>
</tr>
<tr>
<td>E_CTIINVALID_MSG_TYPE_FOR_VERSION</td>
<td>The provided MessageType is invalid for the opened protocol version.</td>
<td>48</td>
</tr>
<tr>
<td>E_CTI_INVALID_TAG_FOR_VERSION</td>
<td>A floating field tag value is invalid for the opened protocol version.</td>
<td>49</td>
</tr>
<tr>
<td>E_CTI_INVALID_AGENT_WORK_MODE</td>
<td>An invalid AgentWorkMode value was provided.</td>
<td>50</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_OPTION</td>
<td>An invalid call option value was provided.</td>
<td>51</td>
</tr>
<tr>
<td>E_CTI_INVALID_DESTINATION_COUNTRY</td>
<td>An invalid destination country value was provided.</td>
<td>52</td>
</tr>
<tr>
<td>E_CTI_INVALID_ANSWER_DETECT_MODE</td>
<td>An invalid answer detect mode value was provided.</td>
<td>53</td>
</tr>
<tr>
<td>E_CTI_MUTUALLY_EXCLUS_DEVICEID_TYPES</td>
<td>A peripheral monitor request may not specify both a call and a device.</td>
<td>54</td>
</tr>
<tr>
<td>E_CTI_INVALID_MONITORID</td>
<td>An invalid monitorID value was provided.</td>
<td>55</td>
</tr>
</tbody>
</table>
Table 6-2  Control Failure Codes (continued)

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTI_SESSION_MONITOR_ALREADY_EXISTS</td>
<td>A requested session monitor was already created.</td>
<td>56</td>
</tr>
<tr>
<td>E_CTI_SESSION_MONITOR IS CLIENTS</td>
<td>A client may not monitor its own session.</td>
<td>57</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_CONTROL_MASK</td>
<td>An invalid call control mask value was provided.</td>
<td>58</td>
</tr>
<tr>
<td>E_CTI_INVALID_FEATURE_MASK</td>
<td>An invalid feature mask value was provided.</td>
<td>59</td>
</tr>
<tr>
<td>E_CTI_INVALID_TRANSFER_CONFERENCE_SETUP_MASK</td>
<td>An invalid transfer conference setup mask value was provided.</td>
<td>60</td>
</tr>
<tr>
<td>E_CTI_INVALID_ARRAY_INDEX</td>
<td>An invalid named array index value was provided.</td>
<td>61</td>
</tr>
<tr>
<td>E_CTI_INVALID_CHARACTER</td>
<td>An invalid character value was provided.</td>
<td>62</td>
</tr>
<tr>
<td>E_CTI_CLIENT_NOT_FOUND</td>
<td>There is no open session with a matching ClientID.</td>
<td>63</td>
</tr>
<tr>
<td>E_CTI_SUPERVISOR_NOT_FOUND</td>
<td>The agent’s supervisor is unknown or does not have an open CTI session.</td>
<td>64</td>
</tr>
<tr>
<td>E_CTI_TEAM_NOT_FOUND</td>
<td>The agent is not a member of an agent team.</td>
<td>65</td>
</tr>
<tr>
<td>E_CTI_NO_CALL_ACTIVE</td>
<td>The specified agent does not have an active call.</td>
<td>66</td>
</tr>
<tr>
<td>E_CTI_NAMED_VARIABLE_NOT_CONFIGURED</td>
<td>The specified named variable is not configured in the ICM database.</td>
<td>67</td>
</tr>
</tbody>
</table>
Table 6-2 Control Failure Codes (continued)

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTI_NAMED_ARRAY_NOT_CONFIGURED</td>
<td>The specified named array is not configured in the ICM database.</td>
<td>68</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_VARIABLE_MASK</td>
<td>The specified call variable mask in not valid.</td>
<td>69</td>
</tr>
<tr>
<td>E_CTI_ELEMENT_NOT_FOUND</td>
<td>An internal error occurred manipulating a named variable or named array element.</td>
<td>70</td>
</tr>
<tr>
<td>E_CTI_INVALID_DISTRIBUTION_TYPE</td>
<td>The specified distribution type is invalid.</td>
<td>71</td>
</tr>
<tr>
<td>E_CTI_INVALID_SKILL_GROUP</td>
<td>The specified skill group is invalid.</td>
<td>72</td>
</tr>
<tr>
<td>E_CTI_TOO_MUCH_DATA</td>
<td>The total combined size of named variables and named arrays may not exceed the limit of 2000 bytes.</td>
<td>73</td>
</tr>
<tr>
<td>E_CTI_VALUE_TOO_LONG</td>
<td>The value of the specified named variable or named array element exceeds the maximum permissible length.</td>
<td>74</td>
</tr>
<tr>
<td>E_CTI_SCALAR_FUNCTION_ON_ARRAY</td>
<td>A NamedArray was specified with a NamedVariable tag.</td>
<td>75</td>
</tr>
<tr>
<td>E_CTI_ARRAY_FUNCTION_ON_SCALAR</td>
<td>A NamedVariable was specified with a NamedArray tag.</td>
<td>76</td>
</tr>
<tr>
<td>E_CTI_INVALID_NUM_NAMED_VARIABLES</td>
<td>The value in the NumNamedVariables field is different than the number of NamedVariable floating fields in the message.</td>
<td>77</td>
</tr>
<tr>
<td>E_CTI_INVALID_NUM_NAMEDARRAYS</td>
<td>The value in the NumNamedArrays field is different than the number of NamedArray floating fields in the message.</td>
<td>78</td>
</tr>
</tbody>
</table>
OnMakeCallConf

The OnMakeCallConf event confirms the successful completion of the MakeCall request. It conveys the information detailed in the table under Parameters.

Syntax

**C++:**  
`int OnMakeCallConf (Arguments & args);`

**COM:**  
`HRESULT OnMakeCallConf ([in] IArguments * args);`

**VB:**  
`Session_ OnMakeCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)`

Parameters

`args`

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewConnectionCallID</td>
<td>The Call ID value assigned to the call by the peripheral or the ICM.</td>
<td>INT</td>
</tr>
<tr>
<td>NewConnectionDeviceID</td>
<td>Indicates the type of the connection identifier supplied in the New ConnectionDeviceID floating field.</td>
<td>SHORT</td>
</tr>
<tr>
<td>IDType</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LineHandle</td>
<td>This field identifies the telset line used, if known. Otherwise this field is set to 0xffff.</td>
<td>SHORT</td>
</tr>
<tr>
<td>LineType</td>
<td>Indicates the type of the telset line given in the LineHandle field.</td>
<td>SHORT</td>
</tr>
<tr>
<td>NewConnectionDeviceID</td>
<td>The identifier of the connection between the call and the device.</td>
<td>STRING</td>
</tr>
<tr>
<td>(required)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OnPostLogout

The OnPostLogout event is generated after the agent has logged out.
IAgentEvents Interface

Chapter 6 Event Interfaces and Events

Syntax

C++: void OnPostLogout (Arguments & args)
COM: void OnPostLogout (IArguments * args)
VB: session_OnPostLogout (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where the agent state change occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>AgentState</td>
<td>SHORT</td>
<td>One of the values in Table 6-1 representing the current overall state of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the associated agent.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The number of the agent SkillGroup affected by the state change, as known</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to the peripheral. May contain the special value NULL_SKILL_GROUP when not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The ICM SkillGroupID of the agent SkillGroup affected by the state change.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>May contain the special value NULL_SKILL_GROUP when not applicable or not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>available.</td>
</tr>
<tr>
<td>StateDuration</td>
<td>INT</td>
<td>The number of seconds since the agent entered this state (typically 0).</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>applicable or not available.</td>
</tr>
<tr>
<td>EventReasonCode</td>
<td>SHORT</td>
<td>A peripheral-specific code indicating the reason for the state change.</td>
</tr>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

IAgentEvents Interface

Remarks

When PG failover occurs, it is possible that the client application will receive an OnPostLogout event with an EventReasonCode of CTOIS_IPCC_FORCED_LOGOUT_REASON_CODE. For example, this may happen on an IPCC system after reconnecting to a different server during a failover, because there is a race condition of the PG logging the agent out and the client reconnecting to the other server before it happens. If this happens, the client application should not disconnect from CTI OS Server.

OnPreLogout

The OnPreLogout event just before the agent is logged out. It allows for any cleanup or logic that needs to be done before logout is completed.

Syntax

C++: void OnPreLogout (Arguments & args)
COM: void OnPreLogout (IArguments * args)
VB: session_OnPreLogout (ByVal args As CtiosCLIENTLib.IArguments)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SkillGroupState</td>
<td>SHORT</td>
<td>Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>The agent’s ACD login ID.</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING</td>
<td>The agent’s ACD teleset extension.</td>
</tr>
<tr>
<td>CTIClientSignature (Optional)</td>
<td>STRING</td>
<td>The Client Signature of the CTI Client that is associated with this agent.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when the agent is on this state.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>A unique object ID for the agent object.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>The agent’s ACD instrument number.</td>
</tr>
</tbody>
</table>
Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where the agent state change occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>AgentState</td>
<td>SHORT</td>
<td>One of the values in Table 6-1 representing the current overall state of the associated agent.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The number of the agent SkillGroup affected by the state change, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The ICM SkillGroupID of the agent SkillGroup affected by the state change. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>StateDuration</td>
<td>INT</td>
<td>The number of seconds since the agent entered this state (typically 0).</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>EventReasonCode</td>
<td>SHORT</td>
<td>A peripheral-specific code indicating the reason for the state change.</td>
</tr>
<tr>
<td>SkillGroupState</td>
<td>SHORT</td>
<td>Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>The agent’s ACD login ID.</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING</td>
<td>The agent’s ACD teleset extension.</td>
</tr>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

IAgentEvents Interface

OnSetAgentModeEvent

The OnSetAgentModeEvent event indicates that the client has made a successful AgentMode connection.

Syntax

C++: void OnSetAgentModeEvent (Arguments& args)
COM:  void OnSetAgentModeEvent (IArguments * args)
VB:   Session_OnSetAgentModeEvent (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTIClientSignature</td>
<td>STRING</td>
<td>The Client Signature of the CTI Client that is associated with this agent.</td>
</tr>
<tr>
<td>Enablement Mask</td>
<td></td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when the agent is on this state.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>A unique object ID for the agent object.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>The agent’s ACD instrument number.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>STRING</td>
<td>ID of the ICM Peripheral ACD associated with the agent.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>The agent’s ID.</td>
</tr>
<tr>
<td>UniqueObject ID</td>
<td>STRING</td>
<td>The new unique object ID for the agent object.</td>
</tr>
<tr>
<td>ClientAgent TemporaryID</td>
<td>STRING</td>
<td>Temporary ID used before server passes the new unique object ID.</td>
</tr>
</tbody>
</table>
**OnSetAgentStateConf**

The OnSetAgentStateConf confirmation message is fired to the client to indicate that the SetAgentState request was received by the CTI Server. This confirmation message does not indicate that the agent has changed to the desired state; rather, the programmer should expect one or more OnAgentStateChange events to indicate the change of state.

**Syntax**

```cpp
c++: int OnSetAgentStateConf (Arguments & args);
```

```com
HResult OnSetAgentStateConf ([out] IArguments * args);
```

```vb
Session_ OnSetAgentStateConf (ByVal args as
CTIOSCLIENTLIB.IArguments)
```

**Parameters**

`args`

Not used; reserved for future use.
### OnLogoutFailed

The `OnLogoutFailed` event is always generated before (or along with) an `OnControlFailureConf` event and is identical to it but is generated only when a Logout request fails.

#### Syntax

**C++**: void OnLogoutFailed (Arguments & args)

**COM**: void OnLogoutFailed (IArguments * args)

**VB**: session_OnLogoutFailed (ByVal args As CtiosCLIENTLib.IArguments)

#### Parameters

`args`

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvokeID</td>
<td>INT</td>
<td>InvokeID of the request that failed</td>
</tr>
<tr>
<td>FailureCode</td>
<td>SHORT</td>
<td>A value specifying the reason that the request failed. See Table 6-2 for a list of the Control Failure Codes.</td>
</tr>
<tr>
<td>Peripheral</td>
<td>INT</td>
<td>Peripheral-specific error data, if available.</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>INT</td>
<td>Zero otherwise.</td>
</tr>
</tbody>
</table>

### OnNewAgentTeamMember

The `OnNewAgentTeamMember` event informs the supervisor about a new agent team member. The event is typically received in response to a `RequestAgentTeamList` request from the supervisor object. It is also received when CTI OS Server receives an `AGENT_TEAM_CONFIG_EVENT` indicating a change in agent team configuration (add/remove).

#### Syntax

**C++**: void OnNewAgentTeamMember (Arguments & args)
IAgentEvents Interface

COM: void OnNewAgentTeamMember (IArguments * args)
VB:    session_OnNewAgentTeamMember (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array that can contain the following fields. Not all fields are always returned. Skillgroup and AgentInstrument are not returned if the agent is not logged in.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>STRING</td>
<td>The ICM PeripheralID of the agent’s ACD.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>Unique object ID of the agent object for this agent.</td>
</tr>
<tr>
<td>AgentState</td>
<td>SHORT</td>
<td>One of the values in Table 6-1 representing the current state of the associated agent.</td>
</tr>
<tr>
<td>NumSkillGroups</td>
<td>INT</td>
<td>The number of skill groups that the agent is currently associated with, up to a maximum of 20.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent’s ACD login.</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING</td>
<td>Agent’s ACD teleset extension.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>Agent’s ACD instrument number.</td>
</tr>
<tr>
<td>AgentLastName</td>
<td>STRING</td>
<td>Agent’s last name.</td>
</tr>
<tr>
<td>AgentFirstName</td>
<td>STRING</td>
<td>Agent’s first name.</td>
</tr>
<tr>
<td>AgentName</td>
<td>STRING</td>
<td>Agent’s full name.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when the agent is on the state specified in the AgentState field.</td>
</tr>
<tr>
<td>SupervisorID</td>
<td>STRING</td>
<td>The ID of the agent’s supervisor.</td>
</tr>
<tr>
<td>AgentFlags</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

IAgentEvents Interface

OnEmergencyCall

The OnEmergencyCall event indicates that a CTI client (with Supervisory capabilities) is handling the indicated call as an emergency call. This event only applies to ACDs with Supervisor capabilities.

Syntax

C++:  void OnEmergencyCall(Arguments& args)
COM: void OnEmergencyCall (IArguments * args)
VB:  session_OnEmergencyCall (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where the call is located.</td>
</tr>
<tr>
<td>Connection CallID</td>
<td>INT</td>
<td>The Call ID value assigned to the call by the peripheral or the ICM.</td>
</tr>
</tbody>
</table>
IAgentEvents Interface

### Remarks

Supported for use with IPCC only.

### OnStartMonitoringAgent

The OnStartMonitoringAgent event is generated when a new agent is selected to be monitored in response to a StartMonitoringAgent() request.

### Syntax

**C++:**
```cpp
void OnStartMonitoringAgent (Arguments& args)
```

**COM:**
```cpp
void OnStartMonitoringAgent (IArguments * args)
```

**VB:**
```vbnet
session_OnStartMonitoringAgent (ByVal args As CtiosCLIENTLib.IArguments)
```

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionDeviceID</td>
<td>INT</td>
<td>The identifier of the connection between the call and the agent’s device.</td>
</tr>
<tr>
<td>IDType</td>
<td>INT</td>
<td>Indicates the type of the connection identifier supplied in the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConnectionDeviceID floating field.</td>
</tr>
<tr>
<td>SessionID</td>
<td>INT</td>
<td>The CTI client SessionID of the CTI client making the notification.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>INT</td>
<td>The identifier of the connection between the call and the agent’s device.</td>
</tr>
<tr>
<td>ClientID (required)</td>
<td>STRING</td>
<td>The ClientID of the client making the notification.</td>
</tr>
<tr>
<td>ClientAddress (Required)</td>
<td>STRING</td>
<td>The IP address of the client making the notification.</td>
</tr>
<tr>
<td>AgentExtension (Required)</td>
<td>STRING</td>
<td>The Agent’s teleset extension.</td>
</tr>
<tr>
<td>AgentID (required)</td>
<td>STRING</td>
<td>The Agent’s ACD login ID.</td>
</tr>
<tr>
<td>AgentInstrument (required)</td>
<td>STRING</td>
<td>The Agent’s ACD instrument number.</td>
</tr>
</tbody>
</table>
Chapter 6      Event Interfaces and Events

IAgentEvents Interface

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>Unique object ID for the supervisor object.</td>
</tr>
<tr>
<td>AgentReference</td>
<td>STRING</td>
<td>String containing the Agent ID for the agent to be monitored.</td>
</tr>
<tr>
<td>SupervisorID</td>
<td>STRING</td>
<td>String containing the supervisor's AgentID</td>
</tr>
<tr>
<td>SupervisorKey</td>
<td>STRING</td>
<td>Supervisor's unique object ID.</td>
</tr>
<tr>
<td>BargedInCallID</td>
<td>STRING</td>
<td>If the supervisor has barged in on the agent's call, the unique object ID of that call.</td>
</tr>
<tr>
<td>SupervisorAgentState</td>
<td>STRING</td>
<td>The supervisor's agent state.</td>
</tr>
</tbody>
</table>

Remarks

This is a Supervisor specific event. It is supported for use with IPCC only.

OnStopMonitoringAgent

The OnStopMonitoringAgent event is generated monitoring of an agent is dropped in response to a StopMonitoringAgent() request.

Syntax

```cpp
void OnStopMonitoringAgent (Arguments & args)
```

```vbnet
session_OnStopMonitoringAgent (ByVal args As CtiosCLIENTLib.IArguments)
```
Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>Unique object ID for the supervisor object.</td>
</tr>
<tr>
<td>AgentReference</td>
<td>STRING</td>
<td>String containing the Agent ID for the agent to be monitored.</td>
</tr>
<tr>
<td>SupervisorID</td>
<td>STRING</td>
<td>String containing the supervisor's AgentID.</td>
</tr>
<tr>
<td>SupervisorKey</td>
<td>STRING</td>
<td>Supervisor’s unique object ID.</td>
</tr>
<tr>
<td>BargedInCallID</td>
<td>STRING</td>
<td>If the supervisor has barged in on the agent’s call, the unique object ID of that call.</td>
</tr>
<tr>
<td>SupervisorAgentState</td>
<td>STRING</td>
<td>The supervisor’s agent state.</td>
</tr>
</tbody>
</table>

Remarks

This is a Supervisor specific event. It is supported for use with IPCC only.

OnQueryAgentStateConf

The OnQueryAgentStateConf event is generated and returned by the server at login as a response to the QueryAgentState() request. A user cannot issue this request.

Syntax

C++: `void OnQueryAgentStateConf(Arguments& args)`
COM: `void OnQueryAgentStateConf (IArguments * args)`
VB: `session_OnQueryAgentStateConf (ByVal args As CtiosCLIENTLib.IArguments)`
### Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent's ACD login.</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING</td>
<td>Agent’s ACD teleset extension.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>Agent’s ACD instrument number.</td>
</tr>
<tr>
<td>AgentState</td>
<td>SHORT</td>
<td>One of the values in Table 6-1 representing the current state of the associated agent.</td>
</tr>
<tr>
<td>NumSkillGroups</td>
<td>INT</td>
<td>The number of skill groups that the agent is currently associated with, up to a maximum of 20.</td>
</tr>
<tr>
<td>SkillGroup[jj]</td>
<td>ARGUMENTS</td>
<td>Argument array that contains Skill Group information for the j-th element less than NumSkillGroups. The message will contain up to NumSkillGroups elements of this type.</td>
</tr>
<tr>
<td>MRDID</td>
<td>INT</td>
<td>Media Routing Domain ID as configured in ICM and the ARM client.</td>
</tr>
<tr>
<td>NumTasks</td>
<td>INT</td>
<td>The number of tasks currently assigned to the agent – this is the number that ICM software compares to the MaxTaskLimit to decide if the agent is available to be assigned additional tasks. This includes active tasks as well as those that are offered, paused, and in wrapup.</td>
</tr>
<tr>
<td>AgentMode</td>
<td>SHORT</td>
<td>The mode that the agent will be in when the login completes. ROUTABLE = 0, NOT ROUTABLE = 1</td>
</tr>
<tr>
<td>MaxTaskLimit</td>
<td>INT</td>
<td>The maximum number of tasks that the agent can be simultaneously working on.</td>
</tr>
</tbody>
</table>
### IAgentEvents Interface

**ICMAgentID** INT  The ICM Skill Target ID, a unique agent identifier for ICM.

**AgentAvailability Status** INT  An agent is *available* to work on a task in this Media Routing Domain if the agent meets all of these conditions:

- The agent is routable for this Media Routing Domain
- The agent is not in Not Ready state for skill groups in other Media Routing Domain
- The agent is *temp routable*, meaning that the agent is not in Reserved, Active, Work-Ready, or Work-Not Ready state on a non-interruptible task in another Media Routing Domain.
- The agent has not reached the maximum task limit for this Media Routing Domain

An available agent is eligible to be assigned a task. Who can assign a task to the agent is determined by whether or not the agent is Routable.

An agent is *ICMAvailable* in MRD X if he is available in X and Routable with respect to X. An agent is *ApplicationAvailable* in MRD X if he is available in X and not Routable with respect to X. Otherwise an agent is *NotAvailable* in MRD X.

- NOT AVAILABLE = 0,
- ICM AVAILABLE = 1,
- APPLICATION AVAILABLE=2

Each SkillGroup\([j]\) field in the message contains the following information.
The OnAgentInfoEvent event is generated as a response to a query to the Agent Lookup Service and carries the agent’s name. This query is generated by the CTI OS server when it is configured to do agent name lookup. The OnAgentInfoEvent event is sent to the client if the server has obtained the information.

**Syntax**

**C++:**
```cpp
void OnAgentInfoEvent(Arguments& args)
```

**COM:**
```cpp
void OnAgentInfoEvent (IArguments * args)
```

**VB:**
```vbnet
session_OnAgentInfoEvent (ByVal args As CtiosCLIENTLib.IArguments)
```

**Parameters**

- **args**

Arguments array containing the following fields.
OnSilentMonitorStartRequest

The OnSilentMonitorStartRequest event is generated as a response to a Silent Monitor request (via the SuperviseCall request).

Syntax

C++: void OnSilentMonitorStartRequest(Arguments& args)
COM: void OnSilentMonitorStartRequest (IArguments * args)
VB: session_OnSilentMonitorStartRequest (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupervisorID</td>
<td>INT</td>
<td>ID of the Supervisor who issued the Silent Monitor request.</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING</td>
<td>Extension of the agent that the supervisor wants to monitor.</td>
</tr>
</tbody>
</table>
Remarks

Supported for use with IPCC only.

OnSilentMonitorStopRequest

Description

The OnSilentMonitorStopRequest event is generated as a response to a SuperviseCall request to stop ongoing silent monitoring.

Syntax

C++:   void OnSilentMonitorStopRequest(Arguments& args)
COM:   void OnSilentMonitorStopRequest (IArguments * args)
VB:   session_OnSilentMonitorStopRequest (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupervisorID</td>
<td>INT</td>
<td>ID of the Supervisor who issued the Silent Monitor request.</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING</td>
<td>Extension of the agent that the supervisor wants to monitor.</td>
</tr>
</tbody>
</table>
Remarks

Supported for use with IPCC only.

OnUserMessageConf

Not supported.

SkillGroupEvents Interface

The SkillGroup object fires events on the ISkillGroupEvents interface. The following events are published to subscribers of the ISkillGroupEvents interface.

OnSkillGroupStatisticsUpdated

The OnSkillGroupStatisticsUpdated event is generated when skill group statistics are reported. The update frequency of OnSkillGroupStatisticsUpdated can be configured on the CTI OS server (see the Cisco ICM Software CTI OS System Manager’s Guide).

Syntax

C++: void OnSkillGroupStatisticsUpdated (Arguments& args)
COM: void OnSkillGroupStatisticsUpdated (IArguments * args)
VB: skillgroup_OnSkillGroupStatisticsUpdated (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.
The statistics event will also contain all the statistics fields listed in Table 11-2 in a nested arguments array named STATISTICS.

**IButtonEnablementEvents**

This interface allows a client application to receive events that indicate what buttons can be enabled on the user interface, given the current agent and current call states.

**OnButtonEnablementChange**

The OnButtonEnablementChange event is received by a client in agent mode whenever CIL receives an agent or call event that carries the EnablementMask field in its parameters. This event allows the client application to enable or disable elements on the user interface. The fields in the event are the same as in OnButtonEnablementChange.

\[
\begin{array}{|l|l|l|}
\hline
\textbf{Keyword} & \textbf{Type} & \textbf{Description} \\
\hline
\text{PeripheralID} & \text{INT} & \text{The ICM PeripheralID of the ACD on which the agent resides.} \\
\hline
\text{SkillGroupNumber} & \text{INT} & \text{The number of the agent skill group as known to the peripheral. May contain the special value NULL\_SKILL\_GROUP when not available.} \\
\hline
\text{SkillGroupID} & \text{INT} & \text{The ICM SkillGroupID of the skill group. May contain the special value NULL\_SKILL\_GROUP when not available.} \\
\hline
\end{array}
\]

C++: void OnButtonEnablementChange (Arguments& args)  
COM: void OnButtonEnablementChange (IArguments * args)  
VB: session_ OnButtonEnablementChange (ByVal args As CtiosCLIENTLib.IArguments)
Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call. See Table 6-3.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>ID of the object (e.g., agent, call) that the event is meant for.</td>
</tr>
<tr>
<td>MessageID</td>
<td>INT</td>
<td>The event that triggered the button enablement change.</td>
</tr>
</tbody>
</table>

Table 6-3 represents the C++/COM/VB enumerations. Enumerations for Java can be found in the description of CtiOsEnums.ButtonEnablement in the Javadoc. It is strongly recommended that bits be referenced by the enumeration rather than the actual number in the bit mask.

Table 6-3  Table of Enablement Bits

<table>
<thead>
<tr>
<th>Button</th>
<th>Bit Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE_ALL</td>
<td>0x00400000</td>
</tr>
<tr>
<td>ENABLE_ANSWER</td>
<td>0X00000001</td>
</tr>
<tr>
<td>ENABLE_RELEASE</td>
<td>0X00000002</td>
</tr>
<tr>
<td>ENABLE_HOLD</td>
<td>0X00000004</td>
</tr>
<tr>
<td>ENABLE_RETRIEVE</td>
<td>0X00000008</td>
</tr>
<tr>
<td>ENABLE_MAKECALL</td>
<td>0X00000010</td>
</tr>
<tr>
<td>ENABLE_TRANSFER_INIT</td>
<td>0X00000020</td>
</tr>
<tr>
<td>ENABLE_TRANSFER_COMPLETE</td>
<td>0X00000040</td>
</tr>
<tr>
<td>ENABLE_SINGLE_STEP_TRANSFER</td>
<td>0X00000080</td>
</tr>
<tr>
<td>ENABLE_CONFERENCE_INIT</td>
<td>0X00000100</td>
</tr>
</tbody>
</table>
### Table 6-3  Table of Enablement Bits (continued)

<table>
<thead>
<tr>
<th>Enablement Event</th>
<th>Mask Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE_CONFERENCE_COMPLETE</td>
<td>0X00000200</td>
</tr>
<tr>
<td>ENABLE_SINGLE_STEP_CONFERENCE</td>
<td>0X00000400</td>
</tr>
<tr>
<td>ENABLE_ALTERNATE</td>
<td>0X00000800</td>
</tr>
<tr>
<td>ENABLE_RECONNECT</td>
<td>0X00001000</td>
</tr>
<tr>
<td>ENABLE_WRAPUP</td>
<td>0X00002000</td>
</tr>
<tr>
<td>ENABLE_INSIDE_MAKECALL</td>
<td>0X00004000</td>
</tr>
<tr>
<td>ENABLE_OUTSIDE_MAKECALL</td>
<td>0X00008000</td>
</tr>
<tr>
<td>ENABLE_SUPERVISOR_ASSIST</td>
<td>0X00010000</td>
</tr>
<tr>
<td>ENABLE_EMERGENCY_CALL</td>
<td>0X00020000</td>
</tr>
<tr>
<td>ENABLE_BAD_LINE_CALL</td>
<td>0X00040000</td>
</tr>
<tr>
<td>ENABLE_STATISTICS</td>
<td>0X00080000</td>
</tr>
<tr>
<td>ENABLE_CHAT</td>
<td>0X00100000</td>
</tr>
<tr>
<td>ENABLE_RECORD</td>
<td>0X00200000</td>
</tr>
<tr>
<td>ENABLE_LOGIN</td>
<td>0X01000000</td>
</tr>
<tr>
<td>ENABLE_LOGOUT</td>
<td>0X02000000</td>
</tr>
<tr>
<td>ENABLE_LOGOUT_WITH_REASON</td>
<td>0x04000000</td>
</tr>
<tr>
<td>ENABLE_READY</td>
<td>0X08000000</td>
</tr>
<tr>
<td>ENABLE_NOTREADY</td>
<td>0X10000000</td>
</tr>
<tr>
<td>ENABLE_NOTREADY_WITH_REASON</td>
<td>0X20000000</td>
</tr>
<tr>
<td>ENABLE_WORKREADY</td>
<td>0X40000000</td>
</tr>
<tr>
<td>ENABLE_WORKNOTREADY</td>
<td>0x80000000</td>
</tr>
<tr>
<td>DISABLE_READY</td>
<td>0xF7FFFFFF</td>
</tr>
<tr>
<td>DISABLE_NOTREADY</td>
<td>0xCFFFFFFF</td>
</tr>
<tr>
<td>DISABLE_WORKREADY</td>
<td>0xBFFFFFFF</td>
</tr>
</tbody>
</table>

**Supervisor Button Enablement Masks**

<table>
<thead>
<tr>
<th>Mask Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE_SET_AGENT_LOGOUT</td>
</tr>
</tbody>
</table>
The OnSupervisorButtonChange is received by a client in agent mode working as supervisor whenever CIL receives a Monitored Agent, Monitored call event that carries the SupervisorBtnEnablementMask field in its parameters. This event allows the client application to enable or disable elements on the user interface. The fields in the event are the same as in OnButtonEnablementChange.

C++: void OnSupervisorButtonChange (Arguments& args)
COM: void OnSupervisorButtonChange (IArguments * args)

### Table 6-3 Table of Enablement Bits (continued)

<table>
<thead>
<tr>
<th>Enablement Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE_SET_AGENT_READY</td>
<td>0x00000002</td>
</tr>
<tr>
<td>ENABLE_SILENTMONITOR</td>
<td>0x00000004</td>
</tr>
<tr>
<td>ENABLE_BARGE_IN</td>
<td>0x00000004</td>
</tr>
<tr>
<td>ENABLE_INTERCEPT</td>
<td>0x00000008</td>
</tr>
<tr>
<td>ENABLE_CLEAR</td>
<td>0x00000010</td>
</tr>
<tr>
<td>DISABLE_SET_AGENT_LOGOUT</td>
<td>0xFFFFFFFF</td>
</tr>
<tr>
<td>DISABLE_SET_AGENT_READY</td>
<td>0xFFFFFFFD</td>
</tr>
<tr>
<td>DISABLE_SILENTMONITOR</td>
<td>0xFFFFFFFB</td>
</tr>
<tr>
<td>DISABLE_BARGE_IN</td>
<td>0xFFFFFFFB</td>
</tr>
<tr>
<td>DISABLE_INTERCEPT</td>
<td>0xFFFFFF7</td>
</tr>
<tr>
<td>DISABLE_CLEAR</td>
<td>0xFFFFFFEF</td>
</tr>
<tr>
<td>DISABLE_SUPERVISE_CALL</td>
<td>DISABLE_BARGE_IN, DISABLE_INTERCEPT, DISABLE_CLEAR, DISABLE_SILENTMONITOR</td>
</tr>
<tr>
<td>DISABLE_SET_AGENT_STATE</td>
<td>DISABLE_SET_AGENT_LOGOUT, DISABLE_SET_AGENT_READY</td>
</tr>
<tr>
<td>DISABLE_ALL_AGENT_SELECT</td>
<td>DISABLE_SUPERVISE_CALL, DISABLE_SET_AGENT_STATE, DISABLE_SILENTMONITOR</td>
</tr>
</tbody>
</table>
ISilentMonitorEvents

The silent monitor manager object fires events on the ISilentMonitorEvents interface. The following events are published to subscribers of the ISilentMonitorEvents interface.

Note

The events in this section are supported for use with IPCC only.

OnSilentMonitorStartRequestedEvent

The OnSilentMonitorStartRequestedEvent event is fired to the subscriber to indicate that a silent monitor session request has arrived and that it will be established on its behalf if the DoDefaultMessageHandling parameter is set to True. The default behavior is to start sending audio and establish the session automatically. If the subscriber wishes to process the event by itself, it must set

VB:  Session_OnSupervisorButtonChange (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupervisorBtn EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call. See Table 6-3.</td>
</tr>
</tbody>
</table>
the DoDefaultMessageHandling parameter to False and invoke
AcceptSilentMonitoring when it is ready to start the session of call
ReportSMSessionStatus to the monitoring client.

CTI OS server generates this event when ever a remote application calls the
StartSilentMonitorRequest method.

**Syntax**

**C++**: void OnSilentMonitorStartRequestedEvent(Arguments & args);
**COM**: HRESULT OnSilentMonitorStartRequestedEvent([in] Arguments* args);
**VB**: Session_ OnSilentMonitorStartRequestedEvent(ByVal args as CTIOSCLIENTLIB.IArguments)

**Parameters**

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID of the agent to be monitored. This message will contain either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>DeviceID</td>
<td>STRING</td>
<td>Device ID of the agent to be monitored. This message will contain either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where the silent monitor start has been requested.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the monitoring application</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port of the monitoring application</td>
</tr>
</tbody>
</table>
The OnSilentMonitorStartedEvent event is fired to the subscriber to indicate that a silent monitor session has been started on its behalf and that audio transmission to the monitoring client has been started.

**Syntax**

```cpp
void OnSilentMonitorStartedEvent(Arguments & args);
```

```com
HRESULT OnSilentMonitorStartedEvent([in] Arguments* args);
```

```vb
Sub Session_OnSilentMonitorStartedEvent(ByVal args As CTIOSCLIENTLIB.IArguments)
```

### Parameters

- **SMSessionKey**
  - Type: UNSIGNED SHORT
  - Description: Unique identifier for the Silent Monitor Session.

- **HeartbeatInterval**
  - Type: INT
  - Description: Heartbeat interval for the silent monitor session.

- **HeartbeatTimeout**
  - Type: INT
  - Description: Timeout for no activity.

- **OriginatingServerID**
  - Type: STRING
  - Description: TCP/IP Address:Port of the CTIOS server from which the request originated.

- **OriginatingClientID**
  - Type: STRING
  - Description: Client Identification of the monitoring application.

- **DoDefaultMessage Handling**
  - Type: BOOLEAN
  - Description: When this parameter is set to True, it instructs the SilentMonitorManager to immediately start sending audio and establish the silent monitor session. If this parameter is set to False, it instructs the SilentMonitorManager to not send voice and to not establish the silent monitor session. In this case, it is the responsibility of the subscriber to report this status accordingly.
Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID of the agent being monitored. This message will contain either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>DeviceID</td>
<td>STRING</td>
<td>Device ID of the agent being monitored. This message will contain either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where silent monitoring has started.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the monitoring application</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port of the monitoring application</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Heartbeat interval for the silent monitor session</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Timeout for no activity.</td>
</tr>
<tr>
<td>OriginatingServerID</td>
<td>STRING</td>
<td>TCP/IP Address:Port of the CTIOS server from which the request originated</td>
</tr>
<tr>
<td>OriginatingClientID</td>
<td>STRING</td>
<td>Client Identification of the monitoring application</td>
</tr>
</tbody>
</table>
OnSilentMonitorStopRequestedEvent

The OnSilentMonitorStopRequestedEvent event is fired to the subscriber to indicate that a silent monitor session was stopped on his behalf. CTI OS server generates this event whenever a remote application calls the StopSilentMonitorRequest method.

Syntax

C++: void OnSilentMonitorStopRequestedEvent(Arguments & args);
COM: HRESULT OnSilentMonitorStopRequestedEvent([in] Arguments* args);
VB: Session_OnSilentMonitorStopRequestedEvent(ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID of the agent who had been monitored. This message will contain either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>DeviceID</td>
<td>STRING</td>
<td>Device ID of the agent who had been monitored. This message will contain either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where silent monitoring has stopped.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the monitoring application.</td>
</tr>
</tbody>
</table>
OnSilentMonitorStatusReportEvent

The OnSilentMonitorStatusReportEvent event indicates a change in status of a silent monitor session. This event sent only to the monitoring application.

Syntax

C++: void OnSilentMonitorStatusReportEvent (Arguments & args);
COM: HRESULT OnSilentMonitorStatusReportEvent ([in] Arguments* args);
VB: Session_ OnSilentMonitorStatusReportEvent (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following fields.

| Keyword                     | Type           | Description                                                                
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObject ID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
<tr>
<td>StatusCode</td>
<td>SHORT</td>
<td>One of the ISilentMonitorEvent status codes in Table 6-4.</td>
</tr>
<tr>
<td>OriginatingServerID</td>
<td>STRING</td>
<td>TCP/IP Address:Port of the CTIOS server from which the request originated</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OriginatingClientID</td>
<td>STRING</td>
<td>Client Identification of the monitoring application</td>
</tr>
<tr>
<td>TargetCILClientID</td>
<td>STRING</td>
<td>CIL Client ID of the monitoring application</td>
</tr>
</tbody>
</table>

**Table 6-4 ISilentMonitorEvent Status Codes**

<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Numeric Value (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General Codes</strong></td>
<td></td>
</tr>
<tr>
<td>eSMStatusUnknown</td>
<td></td>
</tr>
<tr>
<td>eSMStatusOK</td>
<td>-1</td>
</tr>
<tr>
<td>eSMStatusFailed</td>
<td>0</td>
</tr>
<tr>
<td>eSMStatusComError</td>
<td>1</td>
</tr>
<tr>
<td>eSMStatusMonitorStarted</td>
<td>2</td>
</tr>
<tr>
<td>eSMStatusMonitorStopped</td>
<td>3</td>
</tr>
<tr>
<td>eSMStatusHeartbeatTimeout</td>
<td>4</td>
</tr>
<tr>
<td>eSMStatusOutOfMemory</td>
<td>5</td>
</tr>
<tr>
<td>eSMStatusPortUnavailable</td>
<td>6</td>
</tr>
<tr>
<td>eSMStatusIncorrectStateForThisAction</td>
<td>7</td>
</tr>
<tr>
<td>eSMStatusResourceError</td>
<td>8</td>
</tr>
<tr>
<td>eSMStatusRejectedBadParameter</td>
<td>9</td>
</tr>
<tr>
<td>eSMStatusWinsockError</td>
<td>A</td>
</tr>
<tr>
<td>eSMStatusMediaTerminationNotPresent</td>
<td>B</td>
</tr>
<tr>
<td>eSMStatusIPPhoneInformationNotAvailable</td>
<td>C</td>
</tr>
<tr>
<td><strong>Silent Monitor Session Codes</strong></td>
<td></td>
</tr>
<tr>
<td>eSMStatusSessionTerminatedAbnormally</td>
<td>D</td>
</tr>
<tr>
<td>eSMStatusRejectedAlreadyInSession</td>
<td>0x10000000</td>
</tr>
</tbody>
</table>
The OnStartSilentMonitorConf event is sent to the monitoring application to indicate that a StartSilentMonitorRequest has been processed at the CTIOS server.

Syntax

C++: void OnStartSilentMonitorConf (Arguments & args);
COM: HRESULT OnStartSilentMonitorConf ([in] Arguments* args);
vb: Session_OnStartSilentMonitorConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

```
args
Arguments array containing the following fields.
```

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID of the agent to be monitored. This message will contain either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>DeviceID</td>
<td>STRING</td>
<td>Device ID of the agent to be monitored. This message will contain either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where the silent monitor start has been requested.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the monitoring application.</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port of the monitoring application.</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Heartbeat interval for the silent monitor session.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Timeout for no activity.</td>
</tr>
</tbody>
</table>
OnStopSilentMonitorConf

This OnStopSilentMonitorConf event is sent to the monitoring application to indicate that a StopSilentMonitorRequest has been processed at the CTIOS server.

Syntax

C++: void OnStopSilentMonitorConf (Arguments & args);
COM: HRESULT OnStopSilentMonitorConf ([in] Arguments* args);
VB: Session_ OnStopSilentMonitorConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OriginatingServerID</td>
<td>STRING</td>
<td>TCP/IP Address:Port of the CTIOS server from which the request originated.</td>
</tr>
<tr>
<td>OriginatingClientID</td>
<td>STRING</td>
<td>Client Identification of the monitoring application.</td>
</tr>
<tr>
<td>MonitoredUniqueObjectId</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID of the agent who had been monitored. This message will contain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>DeviceID</td>
<td>STRING</td>
<td>Device ID of the agent who had been monitored. This message will contain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>either AgentID or DeviceID, but not both.</td>
</tr>
</tbody>
</table>
The OnSilentMonitorSessionDisconnected event is sent to the application to report errors if the connection fails between the monitoring and monitored clients.

**Syntax**

**C++:**
```cpp
void OnSilentMonitorSessionDisconnected (Arguments & args);
```

**COM:**
```cpp
HRESULT OnSilentMonitorSessionDisconnected ([in] Arguments* args);
```

**VB:**
```vbnet
Session_OnSilentMonitorSessionDisconnected (ByVal args as CTIOSCLIENTLIB.IArguments)
```

**Parameters**

`args`
Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored</td>
</tr>
</tbody>
</table>
The `OnRTPStreamTimedoutEvent` event is sent to the monitored application to report that no RTP voice packets have been received from the monitored IP Phone.

**Syntax**

**C++:**
```c++
void OnRTPStreamTimedoutEvent (Arguments & args);
```

**COM:**
```cpp
HRESULT OnRTPStreamTimedoutEvent ([in] Arguments* args);
```

**VB:**
```cpp
Session_OnRTPStreamTimedoutEvent (ByVal args as CTIOSCLIENTLIB.IArguments)
```

**Parameters**

`args`

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObject ID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
<tr>
<td>StatusCode</td>
<td>SHORT</td>
<td>One of the ISilentMonitorEvent status codes in Table 6-4.</td>
</tr>
</tbody>
</table>
**IMonitoredAgentEvents Interface**

*Note*  
The events in this section are supported for use with IPCC only.

This interface fires Agent events to a supervisor for his team members. IMonitoredAgentEvents are triggered by the supervisor sending a StartMonitoringAllAgentTeams request (see Chapter 9, “Agent Object”). For details on the event parameters please see the IAgentEvents interface.

The most common event being handled is the OnMonitoredAgentStateChange event, which informs a supervisor of agent state changes of agents in the supervisor’s team. All the parameters are the same as for regular OnAgentStateChange events, except for an additional keyword called CTIOS_MONITORED, which indicates that this event is for a monitored agent.

List of Monitored Agent events:

- OnMonitoredAgentStateChange([in] IArguments *pIArguments);
- OnMonitoredAgentInfoEvent([in] IArguments *pIArguments);

**IMonitoredCallEvents Interface**

*Note*  
The events in this section are supported for use with IPCC only.

This interface fires Call events to a supervisor for one of his agent team members. When the supervisor sends a StartMonitoringAgent request (see Chapter 9, “Agent Object”), the supervisor will start receiving MonitoredCallEvents for this “currently” monitored agent. Monitored call events will be received until the supervisor sends a StopMonitoringAgent request for this agent.

The IMonitoredCallEvents interface includes OnMonitoredCallBegin, OnMonitoredCallEnd, and OnMonitoredCallDataUpdate as well as other call events (see list below). These events are described in detail for the ICallEventsInterface. The only difference is that the arguments array contains an additional keyword call CTIOS_MONITORED, indicating that this event is for a monitored call.
List of Monitored Call events:

- OnMonitoredCallBegin([in] IArguments *pIArguments);
- OnMonitoredCallEnd([in] IArguments *pIArguments);
- OnMonitoredCallDataUpdate([in] IArguments *pIArguments);
- OnMonitoredCallDelivered([in] IArguments *pIArguments);
- OnMonitoredCallEstablished([in] IArguments *pIArguments);
- OnMonitoredCallHeld([in] IArguments *pIArguments);
- OnMonitoredCallRetrieved([in] IArguments *pIArguments);
- OnMonitoredCallCleared([in] IArguments *pIArguments);
- OnMonitoredCallConnectionCleared([in] IArguments *pIArguments);
- MonitoredCallReachedNetworkEvent([in] IArguments *pIArguments);
- OnMonitoredCallOriginated([in] IArguments *pIArguments);
- OnMonitoredCallFailed([in] IArguments *pIArguments);
- OnMonitoredCallTransferred([in] IArguments *pIArguments);
- OnMonitoredCallConferenced([in] IArguments *pIArguments);
- OnMonitoredCallDiverted([in] IArguments *pIArguments);
- OnMonitoredTranslationRoute([in] IArguments *pIArguments);
- OnMonitoredCallAgentPreCallEvent([in] IArguments *pIArguments);
- OnMonitoredCallAgentPreCallAbortEvent([in] IArguments *pIArguments);
- MonitoredCallServiceInitiatedEvent([in] IArguments *pIArguments);
- MonitoredCallQueuedEvent([in] IArguments *pIArguments);
- MonitoredCallDequeuedEvent([in] IArguments *pIArguments);

Events in Java CIL

To subscribe for events in the Java CIL, use the AddEventListener method. This method has the following syntax:

```csharp
int AddEventListener(IGenericEvents Listener, int iListID)
```
where Listener is the IGenericEvents object that is subscribing for events and iListID is the ID of the subscriber list to add this listener to. Java subscriber list IDs are part of the CtiOsEnums.SubscriberList interface; each C++/COM/VB event interface has a corresponding Java subscriber list (for example, C++/COM/VB ISessionEvents corresponds to Java eSessionList). See the Javadoc file for more details on the CtiOsEnums.SubscriberList interface.

The IGenericEvents interface, though it contains the C++/COM/VB events documented in this chapter, does not have a callback method for each event. Instead, the OnEvent method passes the event ID code and arguments for each event. The OnEvent method has the following syntax:

```java
void OnEvent(int iEventID, Arguments rArgs)
```

where iEventID is the event ID code for the event and rArgs is the arguments for the event. The arguments for each Java event are the same as for the corresponding C++/COM/VB event. See the Javadoc file for details on the IGenericEvents interface.

To unsubscribe for events in the Java CIL, use the RemoveEventListener method. This method has the following syntax:

```java
int RemoveEventListener(IGenericEvents Listener, int iListID)
```

where Listener is the IGenericEvents object that is unsubscribing for events and iListID is the ID of the subscriber list to remove this listener from.
CtiOs Object

All of the interface objects in the CTIOS Client Interface Library support some common features, such as the IsValid and GetValue methods. This chapter describes these common features.

The CCtiOsObject class is the common base class for the objects in the CTIOS client interface library. It is implemented as follows:

- **In C++**: all of the interface objects (CAgent, CCall, CCtiOsSession, CSkillGroup) derive from it. Thus, all the interface methods described in this chapter are directly available on the C++ objects.

- **In COM (VB and C++)**: the COM objects for Agent, Call, Session, and SkillGroup publish a subset of these methods (as is appropriate for the language), and the underlying implementation of the objects uses the C++ CCtiOsObject class to provide these features.

- **In Java**: all of the interface objects (Agent, Call, Session, and SkillGroup) derive from it. Thus, all the interface methods described in this chapter are directly available on the Java objects.

The CCtiOsObject provides basic services including:

- Dynamic management of the object properties
- Object lifetime control using a reference counting mechanism.
- Run-time class information
Methods

Table 7-1 lists the available CCtiOsObject class methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DumpProperties</td>
<td>Returns a string listing all of an object’s properties’ names and values.</td>
</tr>
<tr>
<td>GetAllProperties</td>
<td>Returns all of the object’s properties as Args (name/value pairs).</td>
</tr>
<tr>
<td>GetElement</td>
<td>Returns the value of an element.</td>
</tr>
<tr>
<td>GetLastError</td>
<td>Returns the last error that occurred on the calling thread.</td>
</tr>
<tr>
<td>GetNumProperties</td>
<td>Returns the number of properties of an object.</td>
</tr>
<tr>
<td>GetPropertyName</td>
<td>Returns a property name in a string format.</td>
</tr>
<tr>
<td>GetPropertyType</td>
<td>Returns the data type of the specified property.</td>
</tr>
<tr>
<td>GetValue, GetValueInt, GetValueString, GetValueArray</td>
<td>Returns the value of a specified property.</td>
</tr>
<tr>
<td>IsValid</td>
<td>Checks to see if the property of an object is valid.</td>
</tr>
</tbody>
</table>

DumpProperties

The DumpProperties method returns all the properties of the object. This method builds a string showing all of the properties in the form “key1 = value1; key2 = value2;...”.

Syntax

```cpp
string DumpProperties ()
```

```com
HRESULT DumpProperties (/*[out,retval]*/ BSTR* bstrValue)
```

```vb
DumpProperties () As String
```

```java
String DumpProperties ()
```
Parameters

bstrValue

The output parameter (return parameter in VB) containing a string listing the names and values of the object’s properties.

Return Value

COM: Default HRESULT return value. See Chapter 3, “CIL Coding Conventions.”

All Others: The string listing the names of all the object’s properties.

GetAllProperties

The GetAllProperties method returns all of the object’s properties and their values. For the properties that are calls, agents, or skillgroups, their string UniqueObjectIDs are returned, not the objects themselves. To get the objects themselves use GetObjectFromObjectID, explained in Chapter 8, “Session Object.”

Syntax

C++: bool GetAllProperties (Arguments** arguments)

COM: HRESULT GetAllProperties /*[out]*/ IArguments** arguments, /*[out,retval]*/ VARIANT_BOOL* errorcode

VB: GetAllProperties arguments As (CTIOSCLIENTLib.IArguments) As Bool

Java: Arguments GetAllProperties()

Parameters

arguments

Output parameter in the form of an arguments array that has all of the property names and values of the object.

errorcode

An output parameter (return parameter in VB) that contains a boolean indicating success or lack thereof.
Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.” The C++ and Java versions of this method return NULL if the value requested is not found or if there is an error.

GetElement

Given a property of type Arguments whose name is specified by the key parameter, the GetElement method returns the Arg at position element of that Arguments array.

Syntax

\[
\begin{align*}
\text{C++: Arg& GetElement (string& key, int element)} \\
\text{Arg& GetElement (int key, int element)} \\
\text{Arg& GetElement (char* key, int element)} \\
\text{COM: HRESULT GetElement /*[in]*/ VARIANT* key, /*[in]*/ int element, /*[out, retval]*/ IArg** pIArg)} \\
\text{VB: GetElement (key As VARIANT) As CTIOSCLIENTLib.IArg} \\
\text{Java: Arg GetElement(String key, int element)} \\
\text{Arg GetElement(int key, int element)}
\end{align*}
\]

Parameters

key

A key designating the name of the Arguments property whose element you want.

element

An integer that is an index of the desired element of the property.

pIArg

An output parameter (return parameter in VB) containing an IArg with the value of the desired element.
Return Value

An Arg reference containing the value of the desired element.
The C++ and Java versions of this method return NULL if the value requested is
not found or if there is an error.

GetLastError (Java only)

The GetLastError method returns the last error that occurred on the calling thread.

Syntax

int GetLastError ()

Parameters

None.

Returns

An Integer object containing the error, or null if the object is not found or if there
is an error.

GetNumProperties

The GetNumProperties method returns the number of properties of that object.

Syntax

C++: int GetNumProperties ()
COM: HRESULT GetNumProperties /*[out,retval]*/ int * num
VB: GetNumProperties () As Long
Java: int GetNumProperties ()
Parameters

    num

    An output parameter (return value in VB and C++) that contains an integer
    that is the number of properties currently a part of the object.

Return Value

COM: Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
All Others: An integer that is the number of properties currently a part of the
object.

GetPropertyName

The GetPropertyName method returns the name of a property in a string format.

Syntax

    C++: string GetPropertyName (int index)
    COM: HRESULT GetPropertyName (/* [in] index, /*[out,retval]*/ name)
    VB: GetPropertyName (index As Integer) As String
    Java: int GetPropertyName (int iIndex)

Parameters

    index

    An integer parameter specifying the property or index number.
    name

    A string output parameter containing the property’s name.

Return Value

COM: Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
All Others: A string that contains the property’s name.
GetPropertyType

The GetPropertyType method returns the data type of the specified property.

Syntax

**C++:**

```cpp
int GetPropertyType (string& key)
int GetPropertyType (int key)
int GetPropertyType (char* key)
```

**COM:**

```cpp
HRESULT GetPropertyType (/*[in]*/ VARIANT* key, /*[out,retval]*/
int* value)
```

**VB:**

```vb
GetPropertyType (key As VARIANT) As Int
```

**Java:**

```java
int GetPropertyType (string key)
int GetPropertyType (int key)
```

Parameters

- **key**
  
The key containing the name of the property whose type you want.

- **value**
  
  An integer pointer to the value of the type

Return Value

**COM:** Default HRESULT return value. See Chapter 3, “CIL Coding Conventions.”

**Others:** An integer indicating the property’s type with the following possible values:

<table>
<thead>
<tr>
<th>Argument Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARG_NOTSET</td>
<td>Argument type not determined</td>
</tr>
<tr>
<td>ARG_INT</td>
<td>Signed integer</td>
</tr>
<tr>
<td>ARG_SHORT</td>
<td>2 bytes signed integer</td>
</tr>
<tr>
<td>ARG_BOOL</td>
<td>1 byte integer</td>
</tr>
<tr>
<td>ARG_STRING</td>
<td>STL character string</td>
</tr>
</tbody>
</table>
The `GetValue` method returns the value of the specified property. Use this method if you don’t know the type of the property. Otherwise, use the more specific `GetValue` methods discussed later in this chapter. Do not use this method for properties of type `IDispatch*`; instead, use `GetCurrentCall`, `GetCurrentAgent`, `GetAllCalls`, `GetAllAgents`, and `GetAllSkillGroups` as explained in Chapter 8, “Session Object.”

### Syntax

**C++**:

```cpp
Arg& GetValue (string& key)
Arg& GetValue (int key)
Arg& GetValue (char* key)
```

**COM**:

```com
HRESULT GetValue (/*[in]*/ VARIANT* key, /*[out,retval]*/ IArg** value)
```

**VB**:

```vb
GetValue (key As VARIANT) As CTIOSCLIENTLib.IArg
```

**Java**:

```java
Arg  GetValue( String key )
Arg& GetValue (int key)
```

### Parameters

- **key**
  - A string reference that contains the name of the property whose value you want.
  
- **value**
An output value of type Arg** containing the property with the designated name. To get the value of the property, call GetType() on the Arg and then call the specific GetValue method, based on the type.

**Return Value**

**COM:** Default HRESULT return value. See Chapter 3, “CIL Coding Conventions.”

**Others:** An Arg containing the specified property, as described in the explanation of the value parameter.

**GetValueArray**

The GetValueArray method returns the Arguments array value of the specified property. Use this method when you know that the property is of Arguments array type, such as ECC call variables.

**Syntax**

**C++:**
```cpp
Arg& GetValueArray (string& key)
Arg& GetValueArray (enum_Keywords key)
Arg& GetValue (char * key)
```

**COM:**
```cpp
HRESULT GetValueArray (/*[in]*/ VARIANT * key, /*[out,retval]*/
    IArguments ** value)
```

**VB:**
```vbnet
GetValueArray (key As VARIANT) As CTIOSCLIENTLib.IArguments
```

**Java:**
```java
Arguments GetValueArray (String key )
Arguments GetValueArray (int key)
```

**Parameters**

**key**
A string reference that contains the name of the property whose value you want.

**value**
An output parameter (return value in VB and C++) containing an IArguments** to the returned value of the property.
Return Value

**COM:** Default HRESULT return value. See Chapter 3, “CIL Coding Conventions.”

**Others:** An reference to an Arguments array that is the value of the specified property.

GetValueBoolObj (Java only)

The GetValueBool method retrieves a Boolean with the specified key from the array.

**Syntax**

Boolean GetValueBoolObj(int iKey)
Boolean GetValueBoolObj(String sKey)

**Parameters**

Key

Key ID for the object to be retrieved.

**Returns**

A Boolean object representation of the contained value or null if error.

GetValueInt

The GetValueInt method returns the integer value of the specified property. Use this method when you know that the property has an integer type.

**Syntax**

C++: int GetValueInt (string& key)
int GetValueInt (int key)
## Parameters

### key
A string reference that contains the name of the property whose value you want.

### value
An output parameter (return parameter in VB) containing an integer pointer to the returned value of the property.

## Return Value

### COM:
Default HRESULT return value. See Chapter 3, “CIL Coding Conventions.”

### Others:
An integer containing the value of the specified property.

### GetValueIntObj (Java only)

Gets the contained value as an integer.

### Syntax

```java
Integer GetValueIntObj( int iKey)
Integer GetValueIntObj( String sKey)
```

### Parameters

#### key
Key ID of the value to be retrieved.
Returns

An Integer object containing a 32 bit signed integer value or null if error.

GetValueShortObj (Java only)

Retrieves a 16 bit short with the specified key from the array.

Syntax

Short GetValueShortObj( int iKey)
Short GetValueShortObj( short sKey)

Parameters

dkey

Key ID of the value to be retrieved.

Returns

A Short object containing a 16 bit short value or null if error.

GetValueString

The GetValueString method returns the string value of the property with the specified name. Use this method when you know that the property is of string type.

Syntax

C++:
string GetValueString (string& key)
string GetValueString (int key)
string GetValueString (char* key)

COM:
HRESULT GetValueString /*[in]*/ VARIANT* key, /*[out,retval]*/
BSTR* value)

VB:
GetValueString (key As VARIANT) As String

Java:
String GetValueString( String key )
String GetValueString (int key)

Parameters

key
A key containing the name of the property whose value you want.

value
An output parameter (return parameter in VB) containing a BSTR pointer to
the returned string value of the property.

Return Value

COM: Default HRESULT return value. See Chapter 3, “CIL Coding
Conventions.”

Others: A string containing the value of the specified property.

GetValueUIntObj (Java only)

Retrieves a 32 bit unsigned integer with the specified key from the array.

Syntax

Long GetValueUIntObj( int iKey)
Long GetValueUIntObj( String sKey)

Parameters

key
Key ID of the value to be retrieved.

Returns

A Long object containing the 32 bit unsigned integer value or null if error.
GetValuеUSHortObj (Java only)

Retrieves a 16 bit unsigned short with the specified key from the array.

Syntax

```
Integer GetValueUShortObj( int iKey)
Integer GetValueUShortObj( String sKey)
```

Parameters

key

Key ID of the value to be retrieved.

Returns

An Integer object containing the 16 bit unsigned short value or null if error.

IsValid

The IsValid method tests to see if the object includes the specified property.

Syntax

```
C++: bool IsValid (string& key)
    bool IsValid (char* key)
    bool IsValid (int key)
COM: HRESULT IsValid (/*[in]*/ VARIANT* key, /*[out,retval]*/
    VARIANT_BOOL* value)
VB: IsValid (key As VARIANT) as Bool
Java: boolean IsValid( String key)
    boolean IsValid (int key)
```

Parameters

key

A key containing the name of the property whose validity you are testing.
value

An output parameter (return parameter in VB) containing a
VARIANT_BOOL pointer indicating whether or not a property with the
specified name exists for the object.

Return Value

COM: Default HRESULT return value. See Chapter 3, “CIL Coding
Conventions.”
Others: A boolean indicating whether or not a property with the specified name
exists for the object.

ReportError (Java only)

The ReportError method sets the value of the LastError property to iErrCode and
writes the error to the log as critical.

Syntax

int ReportError(int iError)

Parameters

Error

The error to report.

Returns

The same error code that was passed in through iErrCode.
SetValue (Java only)

The SetValue method adds a new property to the object’s property list and gives it the provided value. If the property already exists, it replaces its value with the one provided.

Syntax

boolean SetValue(int iKeyID, Arg rArg)
boolean SetValue(int iKeyID, Arguments rArgs)
boolean SetValue(int iKeyID, boolean bValue)
boolean SetValue(int iKeyID, CtiOsObject rObj)
boolean SetValue(int iKeyID, int iValue)
boolean SetValue(int iKeyID, short nValue)
boolean SetValue(int iKeyID, java.lang.String sValue)
boolean SetValue(java.lang.String sPropName, Arg rArg)
boolean SetValue(java.lang.String sPropName, Arguments rArgs)
boolean SetValue(java.lang.String sPropName, boolean bValue)
boolean SetValue(java.lang.String sPropName, CtiOsObject rObj)
boolean SetValue(java.lang.String sPropName, int iValue)
boolean SetValue(java.lang.String sPropName, short nValue)
boolean SetValue(java.lang.String sPropName, java.lang.String sValue)
boolean SetValueUInt (int key, long value)
boolean SetValueUInt (String key, long value)
boolean SetValueUShort (int key, int value)
boolean SetValueUShort (String key, int value)

Parameters

key
The key whose value is to be set.

value
The value to use in setting the element with the designated key.

Returns

True if successfully added, false if not.
**Session Object**

The Client Interface Library’s Session object is used to establish a connection to an active CTI OS server. The main functions of the Session object are:

- Managing the connection to the CTI OS Server
- Distributing events to the appropriate objects and event subscribers
- Creating and managing the collections of Agent, Call, and SkillGroup objects
- Automatically recovering from failures

Typically, an application will have a single instance of the Session object, which is used by all other CIL objects to send and receive events. However, there are no restrictions on the number or types of Session objects one application can employ. It is possible, and sometimes desirable, to establish and manage multiple independent Sessions, for example to use multiple current event streams. If there is more than one Session object monitoring the same Agent or Call, each Session object will receive its own events. The order in which events are received is not guaranteed when there are multiple Session objects.

For a step-by-step explanation of using the Session object to connect with CTI OS Server, see the section “Disconnecting from CTI OS Server” in Chapter 4, “Building Your Application.”

The Session object creates new Call, Agent, and SkillGroup objects upon receipt of an event for that object if the targeted object does not already exist. The Session object maintains collections of all Agents, Calls, SkillGroups, and WaitObjects. Object lifetime is managed by the Session object, and thus it is important that the client application not delete the objects, which would render the object reference
invalid and lead to unpredictable results. When the Session is Released, the connection to CTI OS server is dropped. Any remaining Agent, Call, Skill Group, or WaitObjects will be released.

The remainder of this chapter describes the data properties and interface methods of the Session object.

# Session Object Properties

Table 8-1 lists the available Session properties.

---

**Note**

The data type listed for each keyword is the standardized data type discussed in the section “CTIOS CIL Data Types” in Chapter 3, “CIL Coding Conventions.” See Table 3-1 for the appropriate language specific types for these keywords.

---

### Table 8-1 Session Properties

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectedSince</td>
<td>INT</td>
<td>Time of day in milliseconds when connected.</td>
</tr>
<tr>
<td>ConnectionMode</td>
<td>INT</td>
<td>eAgentConnection, eMonitorConnection, or eNotConnected.</td>
</tr>
<tr>
<td>CtiosA</td>
<td>STRING</td>
<td>Name or TCP/IP address passed as CTI OS server A.</td>
</tr>
<tr>
<td>CtiosB</td>
<td>STRING</td>
<td>Name or TCP/IP address passed as CTI OS server B</td>
</tr>
<tr>
<td>CurrentAgent</td>
<td>object reference</td>
<td>Returns reference to current agent object set by the SetAgent method. Object reference is incremented by one and must be released when no longer used.</td>
</tr>
</tbody>
</table>
Table 8-1 Session Properties (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentCall</td>
<td>object reference</td>
<td>Valid only if in Agent Connect mode. When there is more than one call, this references the current call. The current call is the call selected. For additional information, refer to CurrentCall in Chapter 10, “Call Object.”</td>
</tr>
<tr>
<td>CurrentPort</td>
<td>INT</td>
<td>TCP/IP address of the current connected CTI OS server. May be port A or B.</td>
</tr>
<tr>
<td>CurrentServer</td>
<td>STRING</td>
<td>Name or TCP/IP address of the current connected CTI OS server. The value is blank when the client is not connected to any server. The name may able be blank if it has temporarily lost the current connection even if it is trying to reconnect. Otherwise, the name of the server should be the name of CTI OS server A or B.</td>
</tr>
<tr>
<td>ForcedDisconnect</td>
<td>INT</td>
<td>The presence of this keyword</td>
</tr>
<tr>
<td>Heartbeat</td>
<td>INT</td>
<td>Heartbeat time, expressed in seconds. If not set, default heartbeats are configurable on the CTI OS server.</td>
</tr>
<tr>
<td>LastError</td>
<td>INT</td>
<td>Last error code, if any. Otherwise this value is 0.</td>
</tr>
<tr>
<td>MaxHeartbeats</td>
<td>INT</td>
<td>Max heartbeats that can be missed before switching CTI OS servers. Default is 3 missed heartbeats.</td>
</tr>
<tr>
<td>MessageFilter</td>
<td>STRING</td>
<td>Message expression.</td>
</tr>
<tr>
<td>Object References</td>
<td>ARGUMENTS</td>
<td>Array of object references maintained by the session object. Typically includes Agent References, CallReferences, and SkillGroupReferences. Can also include EmailReferences or Chat References.</td>
</tr>
<tr>
<td>PortA</td>
<td>INT</td>
<td>TCP/IP port for ctiosA.</td>
</tr>
</tbody>
</table>

Cisco ICM Software CTI OS Developer’s Guide Release 6.0(0)
Table 8-1  
Table 8-1  Session Properties (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PortB</td>
<td>INT</td>
<td>TCP/IP port for ctiosB.</td>
</tr>
<tr>
<td>TryingPort</td>
<td>INT</td>
<td>TCP/IP address of the server where a connection is being attempted. May be port A or B.</td>
</tr>
<tr>
<td>TryingServer</td>
<td>STRING</td>
<td>Contains the name or TCP/IP address of the server where a connection is being attempted. The value is blank if no connection is being attempted (see CTIOS_CURRENT SERVER). The name of the server should be the name of CTI OS server A or B.</td>
</tr>
<tr>
<td>TryingSince</td>
<td>INT</td>
<td>Time of day in milliseconds when try began.</td>
</tr>
</tbody>
</table>

Table 8-2 lists the available session object methods.

Table 8-2  Session Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddEventListener</td>
<td>Subscribes a Java IGenericEvents object as a listener on a particular subscriber list.</td>
</tr>
<tr>
<td>AddListener methods</td>
<td>Registers the subscriber for an event listener.</td>
</tr>
<tr>
<td>Connect</td>
<td>Establishes a connection to a CTI OS server.</td>
</tr>
<tr>
<td>CreateSilentMonitorManager</td>
<td>Creates a SilentMonitorManager object instance.</td>
</tr>
<tr>
<td>CreateWaitObject</td>
<td>Creates and returns the pointer to a new CWaitObject.</td>
</tr>
<tr>
<td>DestroySilentMonitorManager</td>
<td>Deletes a SilentMonitorManager object instance.</td>
</tr>
</tbody>
</table>
Table 8-2  Session Object Methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestroyWaitObject</td>
<td>Destroys the specified wait object</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Closes the connection to the CTI OS server.</td>
</tr>
<tr>
<td>DumpProperties</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetAllAgents</td>
<td>Returns a collection of all the agents in the session.</td>
</tr>
<tr>
<td>GetAllCalls</td>
<td>Returns a collection of all the calls in the session.</td>
</tr>
<tr>
<td>GetAllProperties</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetAllSkillGroups</td>
<td>Returns a collection of all the skill groups in the session.</td>
</tr>
<tr>
<td>GetCurrentAgent</td>
<td>Returns the currently selected agent.</td>
</tr>
<tr>
<td>GetCurrentCall</td>
<td>Returns the currently selected call.</td>
</tr>
<tr>
<td>GetCurrentSilentMonitor</td>
<td>Returns a pointer to the SilentMonitorManager object instance that is set as the current manager in the CTI OS session object.</td>
</tr>
<tr>
<td>GetElement</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetNumProperties</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetObjectFromObjectID</td>
<td>Returns a Call, Agent, or SkillGroup, given the object's UniqueObjectID.</td>
</tr>
<tr>
<td>GetPropertyName</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetPropertyType</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetValue</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetValueArray</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetValueInt</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetValueString</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>IsAgent</td>
<td>Checks the current agent and returns true if the current agent is an agent and not a supervisor.</td>
</tr>
<tr>
<td>IsSupervisor</td>
<td>Checks the current agent and returns true if the current agent is a supervisor.</td>
</tr>
</tbody>
</table>
AddEventListener (Java only)

The AddEventListener method subscribes a Java IGenericEvents object as a listener on a particular subscriber list.

Syntax

```java
int AddEventListener(IGenericEvents Listener, int iListID)
```

Parameters

- **Listener**
  
The IGenericEvents object that is subscribing for events.
Chapter 8  Session Object

Methods

ListID

The ID of the subscriber list to which the Listener is to be added.

Returns

A CtiOs_Enums.CilError code indicating success or failure.

AddListener Methods (C++ only)

The AddListener methods registers the subscriber for a specified event listener.

Syntax

```cpp
int AddEventListener(Arguments & rArguments);
int AddSessionEventListener(ISessionEvents * pSessionEvents);
int AddCallEventListener(ICallEvents * pCallEvents);
int AddAgentEventListener(IAgentEvents * pAgentEvents);
int AddSkillGroupEventListener(ISkillGroupEvents * pSkillGroupEvents);
int AddButtonEnablementEventListener(IButtonEnablementEvents * pButtonEvents);
int AddAllInOneEventListener(IAllInOne * pAllInOneEvents);
int AddSilentMonitorEventListener(ISilentMonitorEvents * pSilentMonitorEvents);
int AddSessionEventGenericListener(IGenericEvents * pSessionEvents);
int AddCallEventGenericListener(IGenericEvents * pCallEvents);
int AddAgentEventGenericListener(IGenericEvents * pAgentEvents);
int AddSkillGroupEventGenericListener(IGenericEvents * pSkillGroupEvents);
int AddButtonEnablementEventGenericListener(IButtonEnablementEvents * pButtonEvents);
int AddAllInOneEventGenericListener(IAllInOne * pAllInOneEvents);
int AddSilentMonitorEventGenericListener(ISilentMonitorEvents * pSilentMonitorEvents);
```

Remarks

See also the section “Subscribing for Events in C++” in Chapter 4, “Building Your Application.”
Connect

The Connect method establishes a connection to a CTI OS server.

Syntax

**C++**

```cpp
int Connect(Arguments& args)
```

**COM**

```csharp
HRESULT Connect(IArguments *args, int *errorcode)
```

**VB**

```vbnet
Connect(args As CTIOSCLIENTLib.IArguments) As Long
```

**Java**

```java
int Connect(Arguments args)
```

Parameters

**args**

An arguments array containing the connection parameters listed in Table 8-3.

**Table 8-3 Connect Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtiosA</td>
<td>STRING</td>
<td>Name or TCP/IP address passed as CTI OS server A.</td>
</tr>
<tr>
<td>CtiosB</td>
<td>STRING</td>
<td>Name or TCP/IP address passed as CTI OS server B</td>
</tr>
<tr>
<td>PortA (optional)</td>
<td>INT</td>
<td>TCP/IP port for ctiosA, default = 42028.</td>
</tr>
<tr>
<td>PortB (optional)</td>
<td>INT</td>
<td>TCP/IP port for ctiosB, default = 42028.</td>
</tr>
<tr>
<td>Heartbeat (optional)</td>
<td>INT</td>
<td>Heartbeat time, expressed in seconds. If not set, default heartbeats are configurable on CTI OS server.</td>
</tr>
</tbody>
</table>

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
Remarks

A successful request will result in an OnConnection event.
A failure will result in an OnConnectionFailure event. This means that the CIL is
in failover. The CIL will continue to attempt to connect, alternating between hosts
CTIOS_CTIOSA and CTIOS_CTIOSB until connection succeeds at which point
CIL will fire OnConnection. If application wishes to stop failover, it must call
Disconnect.

In some cases, additional failure codes and events may occur:

- Connect will return a failure code of -1 if it cannot connect with the initial
  side of the duplexed CTIOS server pair chosen from the connect parameters.
  This error code requires no action on the part of the developer as the CIL will
  automatically attempt to connect using the parameters corresponding to the
  other side of the duplexed pair.
- The CIL will retry the connection attempt five times and then will not attempt
to reconnect any longer. The final OnConnectionFailure event will contain
  the keyword "FinalAttempt" which informs the client application that the CIL
  has discontinued its attempts to reconnect.

This behavior will only occur after global settings download has
completed. If global settings download has not completed, the CIL will
continue to retry until successful.

- The Connect method will cause an OnCTIOSFailure event to be fired to the
  client indicating the current state of the system. This is in addition to
  OnConnection or OnConnectionFailure.

The following error codes can occur:

- **CIL_OK** - no obvious errors, application should wait for an event indicating
  whether or not Connect has succeeded

- **CIL_FAIL** - initial attempt to connect with host has failed. CIL will fire
  OnConnectionFailure and go into failover mode. CIL will continue to attempt
  to connect, alternating between hosts CTIOS_CTIOSA and CTIOS_CTIOSB
  until connection succeeds at which point CIL will fire OnConnection. If
  application wishes to stop failover, it must call Disconnect.

- **E_CTIOS_INVALID_ARGUMENT** - a null Arguments parameter was
  supplied. Connect is aborted. No events are fired.
- **E_CTIOS_MISSING_ARGUMENT** - indicates that method call provided no value for both CTIOS_CTIOSA or CTIOS_CTIOSB. At least one of these values must be provided. Connect is aborted. No events are fired.

- **E_CTIOS_IN_FAILOVER** - a previous call to connect failed and CIL is currently in failover attempting to establish a connection. This will continue until a connection is established at which point the CIL will send OnConnection indicating that previous call to Connect has succeeded. If developer wishes to call Connect again with different parameters, he/she must call Disconnect prior to calling Connect again.

- **E_CTIOS_MODE_CONFLICT** - Session is not disconnected (i.e a previous call to Connect is in progress or session is already connected). Disconnect must be called before attempting to establish another connection. CIL may fire an OnConnection event corresponding to previous call to Connect if connection was in progress but will not fire one corresponding to this method call.

- **E_CTIOS_SESSION_NOT_CONNECTED** - unanticipated error. Connect is aborted. No events are fired.

### CreateSilentMonitorManager

The CreateSilentMonitorManager method creates a SilentMonitorManager object instance. To delete the object you must call DestroySilentMonitorManager.

### Syntax

**C++:**
```cpp
CSilentMonitorManager * CreateSilentMonitorManager (Arguments & args);
```

**COM:**
```csharp
HRESULT CreateSilentMonitorManager ( /*[in]*/ IArguments * args, 
/*[out,retval]*/ ISilentMonitorManager * * pISilentMonitor);
```

**VB:**
```vbnet
CreateSilentMonitorManager (ByVal args as CTIOSCLIENTLIB.IArguments) As CTIOSCLIENTLIB.ISilentMonitorManager
```

**Java:** Not available.
Parameters

args

Arguments array that contain the parameters listed below. When any of these parameters are specified the object is constructed with the corresponding property initialized to the specified value. If you want the object to be initialized with the default values specify an empty array.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Heartbeat interval for the silent monitor session.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Timeout for no activity.</td>
</tr>
<tr>
<td>MediaTerminationPort</td>
<td>INT</td>
<td>Required only if manager will be used in monitoring mode. TCP/IP port where monitored conversation will be sent for playback on system sound card.</td>
</tr>
</tbody>
</table>

Return Value

If successful it will return a pointer to a CSilentMonitorManager object. Otherwise, it will return a NULL. To identify the specific error, check the value of the LastError Session property (Table 8-1).

Remarks

Supported for use with IPCC only.

CreateWaitObject (C++ and Java only)

The CreateWaitObject method creates and returns the pointer to a new CWaitObject with the specified event mask.

Syntax

**C++:**
```cpp
CWaitObject * CreateWaitObject(Arguments & args);
```
Methods

Java: WaitObject CreateWaitObject(Arguments rObjParam)

Parameters

args (C++). rObjParam (Java)

A reference to an Arguments object that contains the list of events the object will wait for. The Arguments should contain values where the keys are “Event1” through “EventN” and the values are the enumerated event IDs.

Return Values

If successful it returns a pointer to the new Wait object. Otherwise, it returns NULL.

For more information about CWaitObject see Chapter 12, “Helper Classes.”

DestroySilentMonitorManager

The DestroySilentMonitorManager method deletes a SilentMonitorManager object instance.

Syntax

C++: int DestroySilentMonitorManager(CSilentMonitorManager * pSilentMonitor);
COM: HRESULT DestroySilentMonitorManager(/*[in]*/ ISilentMonitorManager * pSilentMonitor, /*[out,retval]*/ int * errorcode);
VB: DestroySilentMonitorManager (ByVal pSilentMonitor As CTIOSCLIENTLIB.ISilentMonitorManager) As Long
Java: Not available

Parameters

pSilentMonitor

Valid pointer to a SilentMonitorManager object created via CreateSilentMonitorManager.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

Supported for use with IPCC only.

DestroyWaitObject (C++ only)

The DestroyWaitObject method removes the specified CWaitObject from the Session and decrements its reference count.

Syntax

C++: `void DestroyWaitObject(CWaitObject * pWaitObject);`
Java: `void DestroyWaitObject(WaitObject rWaitObj);`

Parameters

WaitObject

A pointer to the CWaitObject to be destroyed.

Return Values

None.

Remarks

For more information about CWaitObject see Chapter 12, “Helper Classes.”
DisableSkillGroupStatistics (C++ and Java only)

The DisableSkillGroupStatistics method requests that sending real-time statistics to the session object be stopped.

Syntax

```c
int DisableSkillGroupStatistics(Arguments & args)
```

Parameters

- **args**
  
  This parameter has two required values for PeripheralId and SkillGroupNumber. See the Remarks section for a code example.

- **errorcode**
  
  An output parameter that contains an error code, if any.

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

C++ code example:

```c
Arguments & argsStatBroadcast = Arguments::CreateInstance();
argsStatBroadcast.AddItem(CTIOS_SKILLGROUPNUMBER, intSG);
argsStatBroadcast.AddItem(CTIOS_PERIPHERALID, m_periphID);
m_pSkGrStatSession->DisableSkillGroupStatistics ( argsStatBroadcast );
argsStatBroadcast.Release();
```
Chapter 8  Session Object

Methods

Disconnect

The Disconnect method disconnects the open connection to the CTI OS server. In Java, you can use the Disconnect method to interrupt failover.

Syntax

C++: void Disconnect (Arguments& args);
COM: HRESULT Disconnect (/* [in, optional */ IArguments *args,
/*/out]*)/ int * errorcode );
VB: Disconnect(args As CTIOSCLIENTLib.IArguments) As Long
Java: int Disconnect(Arguments args)

Parameters

args

An optional arguments array containing the
CTIOS_FORCEDDISCONNECT keyword, which forces a disconnect even if
the Session object rejects the disconnect. This keyword should be added to
the array if the session mode has not yet been set by SetAgent or
SetSessionMode at the time of the disconnect.
errorcode

An output parameter (return parameter in VB) that contains an error code
from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

DumpProperties

EnableSkillGroupStatistics (C++ and Java only)

The EnableSkillGroupStatistics method requests that sending real-time statistics to the session object be started.

Syntax

```c
int EnableSkillGroupStatistics(Arguments & args)
```

Parameters

args

This parameter has two required values for PeripheralId and SkillGroupNumber. See the Remarks section for a code example.

errorcode

An output parameter that contains an error code, if any.

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

C++ code example:

```c
Arguments & argsStatBroadcast = Arguments::CreateInstance();
argsStatBroadcast.AddItem(CTIOS_SKILLGROUPNUMBER, intSG);
argsStatBroadcast.AddItem(CTIOS_PERIPHERALID, m_periphID);
m_pSkGrStatSession->EnableSkillGroupStatistics ( argsStatBroadcast );
argsStatBroadcast.Release();
```
GetAllAgents

The GetAllAgents method provides an array of all agent objects that the session knows about.

Syntax

C++: int GetAllAgents(Arguments *args)
COM: HRESULT GetAllAgents(/*[out, retval]*/ VARIANT *args)
VB: GetAllAgents (args As VARIANT)
Java: Arguments GetAllAgents()

Parameters

args

C++ and Java: A pointer or a reference to an Arguments array where each member has a string key that is the UniqueObjectID of an agent and a value that is a reference to a CilRefArg that is a pointer to the agent object.

COM and VB: A pointer to a VARIANT containing a SAFEARRAY of pointers to IAgents.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

The C++ and Java versions of this method return NULL if the value requested is not found or if there is an error.

GetAllCalls

The GetAllCalls method provides a list of all call objects that the session knows about.
Methods

Chapter 8 Session Object

Syntax

**C++:**  
```cpp
int GetAllCalls (Arguments *args)
```

**COM:**  
```csharp
HRESULT GetAllCalls(/*[out, retval]*/ VARIANT *args)
```

**VB:**  
```vbnet
GetAllCalls (args As VARIANT)
```

**Java:**  
```java
Arguments GetAllCalls()
```

Parameters

`args`

**C++ and Java:** A pointer or a reference to an Arguments array where each member has a key that is the UniqueObjectID of a call and a value that is a reference to a CilRefArg that is a pointer to a call object.

**COM and VB:** A pointer to a VARIANT containing a SAFEARRAY of pointers to ICalls.

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

GetAllProperties


GetAllSkillGroups

The GetAllSkillGroups method provides a list of all skill group objects that the session knows about.

Syntax

**C++:**  
```cpp
int GetAllSkillGroups (Arguments *args)
```

**COM:**  
```csharp
HRESULT GetAllSkillGroups(/*[out, retval]*/ VARIANT *args)
```

**VB:**  
```vbnet
GetAllSkillGroups (args As VARIANT)
```

**Java:**  
```java
Arguments GetAllSkillGroups()
```
Parameters

Args

C++ and Java: A pointer or a reference to an Arguments array where each member has a string key that is the UniqueObjectID of an agent and a value that is a reference to a CilRefArg that is a pointer to the skill group object.

COM and VB: A pointer to a VARIANT containing a SAFEARRAY of pointers to ISKillGroups.

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

The C++ and Java versions of this method return NULL if the value requested is not found or if there is an error.

GetCurrentAgent

The GetCurrentAgent method returns the Agent specified when the Agent Node connection was established. Use this method rather than GetValue(“CurrentAgent”).

Syntax

C++: Agent* GetCurrentAgent()
COM: HRESULT GetCurrentAgent(/*[out, retval]*/ IAgent *agent)
VB: GetCurrentAgent() As CTIOSCLIENTLib.IAgent
Java: Agent GetCurrentAgent()

Parameters

Agent

An output parameter (return value in VB and C++) containing a pointer to a pointer to an ICall that is the currently selected call.
Return Values

C++: Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Others: A pointer to an Agent that is the current agent.

The C++ and Java versions of this method return NULL if the value requested is not found or if there is an error.

GetCurrentCall

The GetCurrentCall method returns the call that is currently selected. This method can be used as a way for controls to communicate between each other which call is selected and therefore should be acted upon. Use this method rather than GetValue(“CurrentCall”).

Syntax

C++: CCall GetCurrentCall()
COM: HRESULT GetCurrentCall(/*[out, retval]*/ ICall ** call)
VB: GetCurrentCall() As CTIOSCLIENTLib.ICall
Java: Call GetCurrentCall()

Parameters

call

An output parameter (return value in VB and C++) containing a pointer to a pointer to an ICall that is the currently selected call.

Return Values

C++: Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Others: A pointer to a Call that is the current call.

The C++ and Java versions of this method return NULL if the value requested is not found or if there is an error.
GetCurrentSilentMonitor

The GetCurrentSilentMonitor method returns a pointer to the SilentMonitorManager object instance that is set as the current manager in the session object.

Syntax

C++: CSilentMonitorManager * GetCurrentSilentMonitor();
COM: HRESULT GetCurrentSilentMonitor (/*[out,retval]*/ ISilentMonitorManager ** pSilentMonitor);
VB: GetCurrentSilentMonitor () As CTIOSCLIENTLIB.ISilentMonitorManager
Java: Not available

Return Values

Pointer to the current Silent Monitor Manager in the session object.

GetElement


GetNumProperties


GetObjectFromObjectID

Given a string containing the UniqueObjectID of a call, an agent, or a skill group, the GetObjectFromObjectID method returns a pointer to the associated object.

Syntax

C++: bool GetObjectFromObjectID (string& uniqueObjectID, CCtiosObject ** object);
COM: HRESULT GetObjectFromObjectID (/*[in]*/ BSTR uniqueObjectID, /*[out]*/ IDispatch ** object, /*[out, retval]*/ VARIANT_BOOL * errorcode);

VB: GetObjectFromObjectID(uniqueObjectID As String, object as IDispatch) As Boolean

Java: CtiOsObject GetObjectFromObjectID(java.lang.String sUID)

Parameters

uniqueObjectID
A string reference that contains the UniqueObjectID of the requested Call, Agent, or Skillgroup object.

object
A pointer to either a CTIOSObject in C++ (which is a CILRefArg) or an IDispatch * pointing to either an ICall, an IAgent, or an ISkillGroup in COM.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

C++: Default HRESULT return value. See Chapter 3, “CIL Coding Conventions.”

Others: A boolean indicating success or failure of the method.

The C++ and Java versions of this method return NULL if the value requested is not found or if there is an error.

Remarks

Many events use UniqueObjectIDs instead of the objects themselves. Use this method to get the object if it is necessary for processing.

GetPropertyByName

See Chapter 7, “CtiOs Object” for a description of the GetPropertyName method.
GetPropertyType

See Chapter 7, “CtiOs Object” for a description of the GetPropertyType method.

GetSystemStatus (Java and C++ only)

The GetSystemStatus method returns the current system status bitmask.

Syntax

int GetSystemStatus()

Parameters

None.

Returns.

The current system status bitmask.

GetValue Methods


IsAgent

The IsAgent method determines whether the current agent is an agent rather than a supervisor.

Syntax

C++:    bool IsAgent()
COM: HRESULT IsAgent (VARIANT_BOOL *bIsAgent)
VB:     IsAgent() As Boolean
**java:**

```java
boolean IsAgent()
```

## Parameters

- **bIsAgent**

  Output parameter (return parameter in VB) that returns true if the current AgentMode connection is for an agent and false if it is for a supervisor.

## Return Values

If the current agent is an agent and not a supervisor it returns true, otherwise it returns false.

## IsSupervisor

The `IsSupervisor` method checks if the current agent is a supervisor.

### Syntax

- **C++:**
  ```cpp
  bool IsSupervisor()
  ```

- **COM:**
  ```cpp
  HRESULT IsSupervisor (VARIANT_BOOL * bIsSupervisor)
  ```

- **VB:**
  ```vbnet
  IsSupervisor () As Boolean
  ```

- **Java:**
  ```java
  boolean IsSupervisorMode()
  ```

## Parameters

- **bIsSupervisor**

  Output parameter (return parameter in VB) that returns true if the current AgentMode connection is for a supervisor and false if it is for an agent.

## Return Values

If the current agent is a supervisor it returns true, otherwise it returns false.
### IsValid


### LogToServer

The LogToServer method logs a message into the CTI OS server log.

#### Syntax

**C++:**

```cpp
int LogToServer(Arguments & args);
```

**COM:**

```cpp
HRESULT LogToServer(/*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode);
```

**Java:**

```java
int LogToServer(Arguments args)
```

#### Parameters

**args**

Arguments array that contain the following parameters.

**Table 8-4 LogToServer Arguments Array Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextMessage</td>
<td>STRING</td>
<td>Text of message to be logged to server</td>
</tr>
<tr>
<td>TraceMask</td>
<td>INT</td>
<td>Trace mask with which to trace the text</td>
</tr>
</tbody>
</table>

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

### Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
RemoveEventListener (Java only)

The RemoveEventListener method unsubscribes a Java IGenericEvents object as a listener from a particular subscriber list.

Syntax

```java
int RemoveEventListener(IGenericEvents Listener, int iListID)
```

Parameters

- **Listener**
  - The IGenericEvents object that is unsubscribing from events.
- **ListID**
  - The ID of the subscriber list from which the Listener is to be removed.

Returns

A CtiOs_Enums.CilError code indicating success or failure.

RemoveListener Methods (C++ only)

The RemoveListener methods unregisters the subscriber from a specified event listener.

Syntax

```cpp
int RemoveEventListener(Arguments & rArguments);
int RemoveSessionEventListener(ISessionEvents * pSessionEvents);
int RemoveCallEventListener(ICallEvents * pCallEvents);
int RemoveAgentEventListener(IAgentEvents * pAgentEvents);
int RemoveSkillGroupEventListener(ISkillGroupEvents * pSkillGroupEvents);
int RemoveButtonEnablementEventListener(IButtonEnablementEvents * pButtonEvents);
int RemoveAllInOneEventListener(IAllInOne * pAllInOneEvents);
```
Chapter 8  Session Object

Methods

```c
int RemoveSilentMonitorEventListener(ISilentMonitorEvents * pSilentMonitorEvents);
int RemoveSessionEventGenericListener(IGenericEvents * pSessionEvents);
int RemoveCallEventGenericListener(IGenericEvents * pCallEvents);
int RemoveAgentEventGenericListener(IGenericEvents * pAgentEvents);
int RemoveSkillGroupEventGenericListener(IGenericEvents * pSkillGroupEvents);
int RemoveButtonEnablementEventGenericListener(IGenericEvents * pButtonEvents);
int RemoveAllInOneEventGenericListener(IGenericEvents * pAllInOneEvents);
int RemoveSilentMonitorEventGenericListener(IGenericEvents * pSilentMonitorEvents);
```

Remarks

See also the section “Subscribing for Events in C++” in Chapter 4, “Building Your Application.”

RequestDesktopSettings

The RequestDesktopSettings method sends a request to the CTI OS Server to download the configuration settings defined for a desktop application.

Syntax

```c++
int RequestDesktopSettings(Arguments& args)

COM: HRESULT RequestDesktopSettings(/* [in] */ IArguments *args, /*[out]*/ int * errorcode)

VB: RequestDesktopSettings (args As CTIOSCLIENTLib.IArguments) As Long

Java:int RequestDesktopSettings(int desktopType)
```

Parameters

```c
args
```

Input parameter in the form of a pointer or reference to an Arguments array containing one number. This number has a keyword of “DesktopType” and an integer value that is either:
Methods

Chapter 8  Session Object

– eAgentDesktop (0)
– eSupervisorDesktop (1)

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

A successful RequestDesktopSettings request results in an OnGlobalSettingsDownloadConf event.

SetAgent

The SetAgent method assigns an agent to this Session object. The agent object used as a parameter in this method should have the following properties set:

• CTIOS_AGENTID
• CTIOS_PERIPHERALID

Syntax

C++: int SetAgent(CAgent& agent)
COM: HRESULT SetAgent(/*[in]*/IAgent *agent, /*[out, retval]*/ int * errorcode)
VB: SetAgent (agent As CTIOSCLIENTLib.IAgent) As Long
Java: int SetAgent(Agent agentObject)

Parameters

agent

The agent to be assigned to the Session object.
errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

If the SetAgent request is successful, it returns a CIL_OK CtiOsEnums.CilError code and fires an OnSetAgentMode event to the client application.

In Java only, if SetAgent() is called on a session where the current agent is different from the agent that SetAgent is trying to set, the following occurs:

- The CIL automatically does a Disconnect on the current session object to Reset an agent.
- An OnCloseConnection event is received.
- A Connect is then performed.
- An OnConnection event is received, and the new agent is set.

In Java only, if the SetAgent request is unsuccessful it returns one of the following CtiOsEnums.CilError codes:

- E_CTIOS_INVALID_SESSION -- if session is not connected.
- E_CTIOS_PROP_ATTRIBUTES_ACCESS_FAILED -- if unable to get the connection mode property
- E_CTIOS_SET_AGENT_SESSION_DISCONNECT_REQUIRED -- if SetAgent request was during a Monitor Mode session. The client application will need to call Disconnect first to clean up the connection mode and then call Connect again.
- E_CTIOS_AGENT_ALREADY_IN_SESSION -- if the agent is already assigned to the session object. The client application will need to call Disconnect first to clean up the connection mode and then call Connect again.
- E_CTIOS_ARGUMENT_ALLOCATION_FAILED -- if the application is unable to allocate memory.
- E_CTIOS_PROP_ATTRIBUTES_ACCESS_FAILED -- if an error occurred while accessing agent properties.
SetCurrentCall

The SetCurrentCall method assigns a call as the session’s current call.

Syntax

C++: int SetCurrentCall(CCall& call)
COM: HRESULT SetCurrentCall ([in] ICall *call, [out, retval] errorcode
VB: SetCurrentCall (call As CTIOSCLIENTLib.ICall)
Java: int SetCurrentCall(Call callObject)

Parameters

call
   Call to assign as current call.
errorcode
   An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

A successful request results in an OnCurrentCallChanged event.
In Java if the call object specified in the call parameter is already the current call, the OnCurrentCallChanged event is not fired to the client application and a E_CTIOS_CALL_ALREADY_CURRENT_IN_SESSION code is returned.

SetCurrentSilentMonitor

The SetCurrentSilentMonitor method sets the SilentMonitorManager object instance specified as the current manager in the CTI OS session object.
Syntax

C++: int SetCurrentSilentMonitor(CSilentMonitorManager * pSilentMonitor);
COM: HRESULT SetCurrentSilentMonitor(/*[in]*/ ISilentMonitorManager * pSilentMonitor, /*[out,retval]*/ int * errorcode);
VB: SetCurrentSilentMonitor (ByVal pSilentMonitor As CTIOSCLIENTLIB.ISilentMonitorManager) As Long
Java: Not available

Parameters

pSilentMonitor
Valid pointer to a SilentMonitorManager object created via CreateSilentMonitorManager

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

Supported for use with IPCC only.

SetMessageFilter

The SetMessageFilter method specifies a filter for CTI OS Server to use to determine which events are sent to a monitor mode client.

Syntax

C++: int SetMessageFilter(string filter)
COM: HRESULT SetMessageFilter(/*[in]*/ BSTR filter, /*[out, retval]*/ int* errorcode)
### SetMessageFilter

**VB:**
```vbnet
SetMessageFilter (filter As String, retVal As Long)
```

**Java:**
```java
int SetMessageFilter(Arguments messageFilter)
```

#### Parameters

- **filter**
  A string containing the message filter, as explained in the section “Notes On Message Filters”.

- **errorcode**
  An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

#### Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

#### Remarks

The Session will receive an OnMonitorModeEstablished event when the filter is set on the server.

### SetSupervisorSilentMonitorMode

The SetSupervisorSilentMonitorMode method can be used to force supervisors into monitored mode. It is used, for example, by the CTIOS Agent desktop to indicate that supervisors logging on to the Agent Desktop can be monitored.

#### Syntax

- **C++:**
  ```cpp
  int SetSupervisorSilentMonitorMode (Arguments & args);
  ```

- **COM:**
  ```cpp
  HRESULT SetSupervisorSilentMonitorMode (/*[in]*/ IArguments * args, /*[out, retval]*/ int * errorcode);
  ```

- **VB:**
  ```vbnet
  SetSupervisorSilentMonitorMode (args As CTIOSCLIENTLib.IArguments);
  ```

- **Java:**
  Not available
Notes On Message Filters

A message filter is a condition that an event must meet in order to be sent to the client. It consists of a keyword/value pair, as explained in the following sections.

Message Filter Syntax

The CTI OS Server’s event filter mechanism enables the rapid creation of powerful CTI integration applications. The event filter allows the developer to create a custom event notification stream using a simple filter expression. The filter expression is sent from the Client Interface Library (CIL) to the CTI OS
server to request an event stream. The CTI OS server’s event filter parses the *filter expression*, and registers the CIL client for all events that match any of the filter’s conditions.

To set a *filter expression*, the Session object’s `SetMessageFilter()` method is used:

```vbscript
'dim sfilterexpression as string
m_session.SetMesageFilter sFilterExpression
```

The general form for a *filter expression* is `key=value`.

### A Simple Example

The most basic event filter is for all events for a specific agent. CTI OS uniquely identifies an agent object by its `UniqueObjectID` (refer to CIL architecture chapter for explanation of the `UniqueObjectID`). To establish an event stream for a unique agent, the following syntax would be used:

```
sFilterExpression = "UniqueObjectID=agent.5000.22866"
```

In this example, the key is the `UniqueObjectID`, and the value is `agent.5000.22866`. This is the same *filter expression* which is implicitly created when a CIL client connects to CTI OS in Agent Mode.

### General Form of Filter Syntax

The event filter syntax can be expressed in the following general form:

```
key1=value1 [,value2, ...] ; [key2=valueA [,valueB, ...]] ...
```

In this form, the *filter expression* must start with a key name (`key`). Following the key must be an equal sign (`=`), and at least one value (`value1`) must be specified. Optionally, additional values (e.g. `value2`, ...) for the same key might be specified (optional parts of the expression are indicated with square brackets `[]`). This will be interpreted as a logical OR among all of the values specified for that key, e.g. if any of those values is found, the condition will be satisfied.

For example, a *filter expression* with one key and multiple values might look like the following:
Notes On Message Filters

sFilterExpression = "AgentID=22866, 22867, 22868"

The interpretation of this filter is to match on any event with AgentID of 22866, 22867, or 22868.

Combining Filters

Multiple filters expressions (as described above) can be combined to create more complex expressions. The general form allows for any number of filters to be concatenated using the semicolon (;), which produces a logical AND effect.

For example, a filter expression with multiple keys, each with multiple values might look like the following:

sFilterExpression = "AgentID=22866, 22867, 22868; SkillGroupNumber=20, 21"

The interpretation of this filter is to match on any event with AgentID of 22866, 22867, or 22868 and with SkillGroupNumber of 20 or 21.

Filtering for Specific Events

One of the most powerful types of event filters for custom applications are filters that work on specific events.

An example of such an application would be an “all agents” real time display, listing the agent states of all known agents at the call center, using the eAgentStateEvent to receive agent updates. To request specific events, use the MessageID keyword, and the numeric value for the event ID that you wish to receive:

\`
"register for all eAgentStateEvents
sFilterExpression = "MessageID = 30"
\`

It is also possible to obtain multiple specific events. For example, consider an all calls real-time display application, using eCallBeginEvent and eCallEndEvent to add or remove calls from a view:

\`
"register for all eCallBeginEvents, eCallEndEvents
sFilterExpression = "MessageID = 23, 24"
\`
Many other permutations are demonstrated by the CILMonitor Visual Basic sample application, included on the CTI OS SDK media.

**Events Not Allowed In Filter Expressions**

The following events cannot be used in filter expressions:

- ePreLogoutEvent
- ePostLogoutEvent
- eOnConnection
- eOnConnectionClosed
- eOnConnectionFailure
- eOnHeartbeat
- eOnMissingHeartbeat
- eOnCurrentCallChanged
- eOnCurrentAgentReset
- Events that are part of the IMonitoredAgentEvents interface or the IMonitoredCallsInterface. This includes the following events:
  - eMonitoredAgentStateEvent
  - eMonitoredAgentInfoEvent
  - eMonitoredCallDeliveredEvent
  - eMonitoredCallEstablishedEvent
  - eMonitoredCallHeldEvent
  - eMonitoredCallRetrievedEvent
  - eMonitoredCallClearedEvent
  - eMonitoredCallConnectionClearedEvent
  - eMonitoredCallOriginatedEvent
  - eMonitoredCallFailedEvent
  - eMonitoredCallConferencedEvent
  - eMonitoredCallTransferredEvent
  - eMonitoredCallDivertedEvent
Notes On Message Filters

- eMonitoredCallServiceInitiatedEvent
- eMonitoredCallQueuedEvent
- eMonitoredCallTranslationRouteEvent
- eMonitoredCallBeginEvent
- eMonitoredCallEndEvent
- eMonitoredCallDataUpdateEvent
- eMonitoredCallReachedNetworkEvent
- eMonitoredCallDequeuedEvent
- eMonitoredAgentPrecallEvent
- eMonitoredAgentPrecallAbortEvent

To circumvent this restriction, use an equivalent message in the filter expression (for example, OnAgentStateEvent instead of OnMonitoredAgentStateEvent) and check in the message handler for the CTIOS_MONITORED parameter to be TRUE.

```cpp
void CMyEventSink::OnAgentStateEvent(Arguments & argParams)
{
    if (argParams.IsValid(CTIOS_MONITORED) &&
        argParams.GetValueBoolean(CTIOS_MONITORED))
    {
        //Do process the event
    }
}
```

Filtering Skillgroup Statistics

One of the most common applications for a filter mode application is the processing of only skill group statistics. For this purpose, the specialized filter "FilterTarget=SkillGroupStats" is defined. When set, this filter indicates that the CTI OS server should forward skill group statistics to the client application, whether or not any agents are logged in.
After the filter is set, the client application needs to invoke the EnableSkillGroupStatistics(...) method for each skill group that it is expecting to receive statistic. To stop receiving statistics for a given skill group, the application must invoke DisableSkillGroupStatistics.

'\register to receive skill group statistics
sFilterExpression="FilterTarget=SkillGroupStats"
'\call SetMessageFilter
m_session.SetMessageFilter sFilterExpression
'\Enable statistics for skills 78, 89 and 90 in peripheral 5004
Private Sub m_Session_OnMonitorModeEstablished(ByVal pArguments As Arguments)
    Dim m_Args = new Arguments
    'For Skill 78
    m_Args.AddItem "SkillGroupNumber", 78
    m_Args.AddItem "PeripheralID", 5004
    m_session.EnableSkillGroupStatistics m_Args
    'For Skill 89
    m_Args.Clear
    m_Args.AddItem "SkillGroupNumber", 89
    m_Args.AddItem "PeripheralID", 5004
    m_session.EnableSkillGroupStatistics m_Args
    'For Skill 90
    m_Args.Clear
    m_Args.AddItem "SkillGroupNumber", 90
    m_Args.AddItem "PeripheralID", 5004
    m_session.EnableSkillGroupStatistics m_Args
    'Don't need arguments any more
Set m_Arg = Nothing
End Sub

Private Sub MyObjectClass_OnCleanupApplication()
    Dim m_Args = new Arguments
    'For Skill 78
    m_Args.AddItem "SkillGroupNumber", 78
    m_Args.AddItem "PeripheralID", 5004
    m_session.DisableSkillGroupStatistics m_Args
    'For Skill 89
    m_Args.Clear
    m_Args.AddItem "SkillGroupNumber", 89
    m_Args.AddItem "PeripheralID", 5004
    m_session.DisableSkillGroupStatistics m_Args
    'For Skill 90
    m_Args.Clear
    m_Args.AddItem "SkillGroupNumber", 90
    m_Args.AddItem "PeripheralID", 5004
    m_session.DisableSkillGroupStatistics m_Args
    'Don't need arguments any more
Set m_Arg = Nothing
End Sub
Agent Object

The Agent object provides developers using the CTI OS Client Interface Library with an interface to Agent behavior. The Agent object exposes methods to perform all agent behaviors, such as logging in and setting the agent’s state.

The object stores specific agent information as properties, includes the AgentID, AgentPassword, AgentInstrument, AgentExtension, and SkillGroup(s). When the agent is logged into an ACD, the agent object will receive updates via AgentStateEvents and Agent Statistics updates.

The Agent object can be used in two different modes:

- In Agent Mode, the application should create an Agent object and inform the Session about the agent using `Session.SetAgent()`.
- In Monitor Mode, the client application sets a message filter, and if the event stream involves events for Agent object(s), those objects will be dynamically created at the CIL as needed.

Agent Object Properties

Table 9-1 lists the agent object properties.

| Note | The data type listed for each keyword is the standardized data type discussed in the section “CTIOS CIL Data Types” in Chapter 3, “CIL Coding Conventions.” See Table 3-1 for the appropriate language specific types for these keywords. |
Table 9-1  Agent Properties

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentAvailability</td>
<td>INT</td>
<td>One of the following values: UNKNOWN (-1), NOT AVAILABLE (0), ICM AVAILABLE (1), or APPLICATION AVAILABLE (2).</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING</td>
<td>Extension associated by ACD to agent.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Can be set prior to Login or after Logout.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>Instrument associated by ACD to agent.</td>
</tr>
<tr>
<td>AgentState</td>
<td>SHORT</td>
<td>One of the values in Table 6-1 representing the current state of the associated agent.</td>
</tr>
<tr>
<td>ClassIdentifier</td>
<td>INT</td>
<td>Identifies the type of this object.</td>
</tr>
<tr>
<td>Extension</td>
<td>STRING</td>
<td>Extension associated by ACD to agent.</td>
</tr>
<tr>
<td>IsSupervisor</td>
<td>INT</td>
<td>Indicates whether this agent is a supervisor.</td>
</tr>
<tr>
<td>LastError</td>
<td>INT</td>
<td>Last error code, if any. Otherwise this value is 0.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>ID of peripheral.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>INT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>SavedAgentID</td>
<td>STRING</td>
<td>On Spectrum, contains the user provided data for the agent specified by the AgentID property. On all other switches, this property is identical to the AgentID property.</td>
</tr>
<tr>
<td>Statistics</td>
<td>ARGUMENTS</td>
<td>An arguments array containing the statistics listed in Table 9-2.</td>
</tr>
</tbody>
</table>
Agent Statistics

Statistics can be accessed by first using GetValueArray on the Agent object to obtain the “Statistics” arguments array and then using GetValueInt on the “Statistics” arguments array to obtain the desired value:

' First get the statistics arguments
Dim args As Arguments
args = agent.GetValueArray("Statistics")

' Then get the desired statistics
Dim availTimeSession As Integer
Dim loggedOnTimeSession As Integer
availTimeSession = args.GetValueInt("AvailTimeSession")
bargeInCallsToday = args.GetValueInt("BargeInCallsToday")

Note: Not all the statistics values listed in Table 9-2 are present in every system configuration. Whether or not a particular statistic value is available depends both on the protocol version of CTI Server with which CTI OS connects and on the peripheral on which the agent resides.

Table 9-2  Agent Statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailTime Session</td>
<td>Total time, in seconds, the agent was in the Available state for any skill group.</td>
</tr>
<tr>
<td>LoggedOnTime Session</td>
<td>Total time, in seconds, the agent has been logged on.</td>
</tr>
<tr>
<td>NotReadyTime Session</td>
<td>Total time, in seconds, the agent was in the Not Ready state for all skill groups.</td>
</tr>
<tr>
<td>ICMAvailable TimeSession</td>
<td>Total time, in seconds, the agent was in the ICM Available state.</td>
</tr>
<tr>
<td>RoutableTime Session</td>
<td>Total time, in seconds, the agent was in the Routable state for all skill groups.</td>
</tr>
<tr>
<td>AgentOutCalls Session</td>
<td>Total number of completed outbound ACD calls made by agent.</td>
</tr>
</tbody>
</table>
### Table 9-2  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AgentOutCalls TalkTimeSession</strong></td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td><strong>AgentOutCalls Time Session</strong></td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td><strong>AgentOutCalls Held Session</strong></td>
<td>The total number of completed outbound ACD calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td><strong>AgentOutCalls HeldTime Session</strong></td>
<td>Total number of seconds outbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td><strong>HandledCalls Session</strong></td>
<td>The number of inbound ACD calls handled by the agent.</td>
</tr>
<tr>
<td><strong>HandledCalls TalkTime Session</strong></td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by the agent. Includes hold time associated with the call.</td>
</tr>
<tr>
<td><strong>HandledCalls AfterCall TimeSession</strong></td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.</td>
</tr>
</tbody>
</table>
## Table 9-2  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandledCalls Time Session</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by the agent. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>IncomingCalls Held Session</td>
<td>The total number of completed inbound ACD calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>IncomingCalls HeldTime Session</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>InternalCallsSession</td>
<td>Number of internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCalls TimeSession</td>
<td>Number of seconds spent on internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCalls RcvdSession</td>
<td>Number of internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCalls RcvdTime Session</td>
<td>Number of seconds spent on internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCalls HeldSession</td>
<td>The total number of internal calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCalls HeldTime Session</td>
<td>Total number of seconds completed internal calls were placed on hold.</td>
</tr>
<tr>
<td>AutoOutCalls Session</td>
<td>Total number of AutoOut (predictive) calls completed by the agent.</td>
</tr>
<tr>
<td>AutoOutCalls TalkTime Session</td>
<td>Total talk time, in seconds, of AutoOut (predictive) calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
### Table 9-2  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoOutCalls Time Session</td>
<td>Total handle time, in seconds, for AutoOut (predictive) calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCalls Held Session</td>
<td>The total number of completed AutoOut (predictive) calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>AutoOutCalls HeldTime Session</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold.</td>
</tr>
<tr>
<td>PreviewCalls Session</td>
<td>Total number of outbound Preview calls completed by the agent.</td>
</tr>
<tr>
<td>PreviewCalls TalkTime Session</td>
<td>Total talk time, in seconds, of outbound Preview calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCalls TimeSession</td>
<td>Total handle time, in seconds, outbound Preview calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCalls HeldSession</td>
<td>The total number of completed outbound Preview calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>PreviewCalls HeldTime Session</td>
<td>Total number of seconds outbound Preview calls were placed on hold.</td>
</tr>
</tbody>
</table>
### Table 9-2  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservation CallsSession</td>
<td>Total number of agent reservation calls completed by the agent.</td>
</tr>
<tr>
<td>Reservation CallsTalk Time Session</td>
<td>Total talk time, in seconds, of agent reservation calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>Reservation CallsTime Session</td>
<td>Total handle time, in seconds, agent reservation calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>Reservation CallsHeld Session</td>
<td>The total number of completed agent reservation calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>Reservation CallsHeld Time Session</td>
<td>Total number of seconds agent reservation calls were placed on hold.</td>
</tr>
<tr>
<td>BargeInCalls Session</td>
<td>Total number of supervisor call barge-ins completed.</td>
</tr>
<tr>
<td>InterceptCalls Session</td>
<td>Total number of supervisor call intercepts completed.</td>
</tr>
<tr>
<td>MonitorCalls Session</td>
<td>Total number of supervisor call monitors completed.</td>
</tr>
<tr>
<td>WhisperCalls Session</td>
<td>Total number of supervisor whisper calls completed.</td>
</tr>
<tr>
<td>EmergencyCallsSession</td>
<td>Total number of emergency calls.</td>
</tr>
<tr>
<td>AvailTimeToday</td>
<td>Total time, in seconds, the agent was in the Available state for any skill group.</td>
</tr>
</tbody>
</table>
Agent Statistics

Table 9-2  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoggedOnTime Today</td>
<td>Total time, in seconds, the agent has been logged on.</td>
</tr>
<tr>
<td>NotReadyTime Today</td>
<td>Total time, in seconds, the agent was in the Not Ready state for all skill groups.</td>
</tr>
<tr>
<td>ICMAvailable TimeToday</td>
<td>Total time, in seconds, the agent was in the ICM Available state.</td>
</tr>
<tr>
<td>RoutableTime Today</td>
<td>Total time, in seconds, the agent was in the Routable state for all skill groups.</td>
</tr>
<tr>
<td>AgentOutCalls Today</td>
<td>Total number of completed outbound ACD calls made by agent.</td>
</tr>
<tr>
<td>AgentOutCalls TalkTime</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCalls Time</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCalls HeldToday</td>
<td>The total number of completed outbound ACD calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>AgentOutCalls HeldTime</td>
<td>Total number of seconds outbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>HandledCalls Today</td>
<td>The number of inbound ACD calls handled by the agent.</td>
</tr>
</tbody>
</table>
### Table 9-2  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandledCalls TalkTime Today</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by the agent. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>HandledCalls AfterCall TimeToday</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.</td>
</tr>
<tr>
<td>HandledCalls TimeToday</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by the agent. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>IncomingCalls HeldToday</td>
<td>The total number of completed inbound ACD calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>IncomingCalls HeldTime Today</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>InternalCalls Today</td>
<td>Number of internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCalls TimeToday</td>
<td>Number of seconds spent on internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCalls RcvdToday</td>
<td>Number of internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCalls RcvdTime Today</td>
<td>Number of seconds spent on internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCalls HeldToday</td>
<td>The total number of internal calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCalls HeldTime Today</td>
<td>Total number of seconds completed internal calls were placed on hold.</td>
</tr>
<tr>
<td>AutoOutCalls Today</td>
<td>Total number of AutoOut (predictive) calls completed by the agent.</td>
</tr>
</tbody>
</table>
## Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoOutCalls TalkTime Today</td>
<td>Total talk time, in seconds, of AutoOut (predictive) calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCalls TimeToday</td>
<td>Total handle time, in seconds, for AutoOut (predictive) calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCalls HeldToday</td>
<td>The total number of completed AutoOut (predictive) calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>AutoOutCalls HeldTime Today</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold.</td>
</tr>
<tr>
<td>PreviewCalls Today</td>
<td>Total number of outbound Preview calls completed by the agent.</td>
</tr>
<tr>
<td>PreviewCalls TalkTimeToday</td>
<td>Total talk time, in seconds, of outbound Preview calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
### Table 9-2 Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviewCalls TimeToday</td>
<td>Total handle time, in seconds, outbound Preview calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCalls HeldToday</td>
<td>The total number of completed outbound Preview calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>PreviewCalls HeldTimeToday</td>
<td>Total number of seconds outbound Preview calls were placed on hold.</td>
</tr>
<tr>
<td>Reservation CallsToday</td>
<td>Total number of agent reservation calls completed by the agent.</td>
</tr>
<tr>
<td>Reservation CallsTalk TimeToday</td>
<td>Total talk time, in seconds, of agent reservation calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>Reservation CallsTimeToday</td>
<td>Total handle time, in seconds, agent reservation calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>Reservation CallsHeldToday</td>
<td>The total number of completed agent reservation calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>Reservation CallsHeld TimeToday</td>
<td>Total number of seconds agent reservation calls were placed on hold.</td>
</tr>
</tbody>
</table>
Table 9-2  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BargeInCalls Today</td>
<td>Total number of supervisor call barge-ins completed.</td>
</tr>
<tr>
<td>InterceptCalls Today</td>
<td>Total number of supervisor call intercepts completed.</td>
</tr>
<tr>
<td>MonitorCalls Today</td>
<td>Total number of supervisor call monitors completed.</td>
</tr>
<tr>
<td>WhisperCalls Today</td>
<td>Total number of supervisor whisper calls completed.</td>
</tr>
<tr>
<td>EmergencyCalls Today</td>
<td>Total number of emergency calls.</td>
</tr>
<tr>
<td>AvailTime Session</td>
<td>Total time, in seconds, the agent was in the Available state for any skill group.</td>
</tr>
<tr>
<td>LoggedOnTime Session</td>
<td>Total time, in seconds, the agent has been logged on.</td>
</tr>
<tr>
<td>NotReadyTime Session</td>
<td>Total time, in seconds, the agent was in the Not Ready state for all skill groups.</td>
</tr>
<tr>
<td>ICMAvailable TimeSession</td>
<td>Total time, in seconds, the agent was in the ICM Available state.</td>
</tr>
<tr>
<td>RoutableTime Session</td>
<td>Total time, in seconds, the agent was in the Routable state for all skill groups.</td>
</tr>
<tr>
<td>AgentOutCalls Session</td>
<td>Total number of completed outbound ACD calls made by agent.</td>
</tr>
</tbody>
</table>

Methods

Table 9-3 lists the supported Agent object methods.

Table 9-3  Agent Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisableAgentStatistics</td>
<td>Disables agent statistic messages.</td>
</tr>
<tr>
<td>DisableSkillGroupStatistics</td>
<td>Disables skill group statistic messages.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>DumpProperties</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>EnableAgentStatistics</td>
<td>Enables agent statistic messages.</td>
</tr>
<tr>
<td>EnableSkillGroupStatistics</td>
<td>Enables skill group statistic messages.</td>
</tr>
<tr>
<td>GetAgentState</td>
<td>Returns the current agent state.</td>
</tr>
<tr>
<td>GetAllProperties</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetElement</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetMonitoredAgent</td>
<td>Returns the agent object that is currently being monitored.</td>
</tr>
<tr>
<td>GetMonitoredCall</td>
<td>Returns the call object that is currently being monitored.</td>
</tr>
<tr>
<td>GetNumProperties</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetPropertyName</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetPropertyType</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetSkillGroups</td>
<td>Returns an array of SkillGroups objects</td>
</tr>
<tr>
<td>GetValue</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetValueArray</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetValueInt</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetValueString</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>IsAgent</td>
<td>Checks the current mode and returns true if agent mode.</td>
</tr>
<tr>
<td>IsSupervisor</td>
<td>Checks the current mode and returns true if supervisor mode.</td>
</tr>
<tr>
<td>IsValid</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>Login</td>
<td>Logs an agent into the ACD.</td>
</tr>
<tr>
<td>Logout</td>
<td>Logs an agent out of the ACD.</td>
</tr>
<tr>
<td>MakeCall</td>
<td>Initiates a call to a device or agent</td>
</tr>
<tr>
<td>MakeEmergencyCall</td>
<td>Lets an agent makes an emergency call to the supervisor.</td>
</tr>
</tbody>
</table>
Arguments Parameters

The following rules apply to the optional_args and reserved_args parameters in Call Object methods:

- In VB, you can ignore these parameters altogether. For example, you can treat the line:

Table 9-3  Agent Object Methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryAgentState</td>
<td>Gets the current agent state from CTI Server and retrieves it.</td>
</tr>
<tr>
<td>ReportBadCallLine</td>
<td>Informs the CTI OS Server of a bad line.</td>
</tr>
<tr>
<td>RequestAgentTeamList</td>
<td>Retrieves the current agent team list.</td>
</tr>
<tr>
<td>RequestSupervisorAssist</td>
<td>Allows the agent to call an available supervisor for assistance.</td>
</tr>
<tr>
<td>SendChatMessage</td>
<td>Send asynchronous messages between CTI clients</td>
</tr>
<tr>
<td>SetAgentState</td>
<td>Requests a new agent state.</td>
</tr>
<tr>
<td>SetValue</td>
<td>Sets the value of the property whose name is specified.</td>
</tr>
<tr>
<td>StartMonitoringAgent</td>
<td>Enables monitoring of a specified agent.</td>
</tr>
<tr>
<td>StartMonitoringAgentTeam</td>
<td>Enables monitoring of a specified agent team.</td>
</tr>
<tr>
<td>StartMonitoringAllAgentTeams</td>
<td>Enables monitoring of all agent teams.</td>
</tr>
<tr>
<td>StartMonitoringCall</td>
<td>Enables monitoring of a specified call object.</td>
</tr>
<tr>
<td>StopMonitoringAgent</td>
<td>Disables monitoring of a specified agent.</td>
</tr>
<tr>
<td>StopMonitoringAgentTeam</td>
<td>Disables monitoring of a specified agent team.</td>
</tr>
<tr>
<td>StopMonitoringAllAgentTeams</td>
<td>Disables monitoring of all agent teams.</td>
</tr>
<tr>
<td>SuperviseCall</td>
<td>Enables monitoring a call of an agent on your team.</td>
</tr>
</tbody>
</table>
Answer([reserved_args As IArguments]) As Long

as follows:
Answer()

- To ignore these parameters in COM you must send a NULL, as shown:
  Answer(NULL)

**DisableAgentStatistics**

The DisableAgentStatistics method is sent by an agent to request that real-time statistics stop being sent to that agent.

**Syntax**

**C++:**
```cpp
int DisableAgentStatistics (Arguments& reserved_args)
```

**COM:**
```cpp
HRESULT DisableAgentStatistics /*[in]*/ IArguments reserved_args, /* [out, retval]*/ int * errorcode
```

**VB:**
```vbnet
DisableAgentStatistics (reserved_args As CTIOSCLIENTLib.IArguments) As Long
```

**Java:**
```java
int DisableAgentStatistics (Arguments reservedargs);```

**Parameters**

reserved_args

Not currently used, reserved for future use.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

**Return Value**

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
DisableSkillGroupStatistics

The DisableSkillGroupStatistics method is sent by an agent to request that real-time statistics stop being sent to that agent.

Syntax

C++: int DisableSkillGroupStatistics (Arguments& optional_args)
COM: HRESULT DisableSkillGroupStatistics (/* [in, optional]*/
IArguments * optional_args, /* [out, retval]*/ int * errorcode)
VB: DisableSkillGroupStatistics (optional_args As CTIOSCLIENTLib.IArguments) As Long
Java: int DisableSkillGroupStatistics (Arguments optional_args);

Parameters

optional_args

An optional input parameter containing a pointer or a reference to an Arguments array containing a member that is a nested Arguments array with the keyword “SkillGroupNumbers”. Within this array, each number has a string key of an integer starting with “1” and an integer value that is a SkillGroupNumber to be disabled. If the parameter is NULL or missing, statistics will be disabled for all skill groups to which the agent belongs.

derrorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

EnableAgentStatistics

The EnableAgentStatistics method is sent by an agent to request that real-time statistics be sent to that agent.
Chapter 9      Agent Object

Methods

Syntax

C++:   int EnableAgentStatistics(Arguments& reserved_args)
COM:   HRESULT EnableAgentStatistics /*[in]*/  IArguments* reserved_args, /* [out, retval]*/  int * errorcode
VB:    EnableAgentStatistics (reserved_args As CTIOSCLIENTLib.IArguments) As Long
Java:  int EnableAgentStatistics (Arguments args)

Parameters

reserved_args
   Not currently used, reserved for future use.
errorcode
   An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

The CTI OS server sends agent statistics in an OnAgentStatistics event. See the OnAgentStatistics section in Chapter 6, “Event Interfaces and Events” for an explanation of the PollingIntervalSec and PollForAgentStatsAtEndCall registry settings and how these settings affect the refresh rate of agent statistics.

EnableSkillGroupStatistics

The EnableSkillGroupStatistics method is sent by an agent to request that real-time statistics be sent to that agent.

Syntax

C++:  int EnableSkillGroupStatistics (Arguments& optional_args)
COM: HRESULT EnableSkillGroupStatistics /*[in]*/ IArguments * optional_args, /* [out, retval]*/ int * errorcode

VB: EnableSkillGroupStatistics (optional_args As CTIOSCLIENTLib.IArguments) As Long

Java: int EnableSkillGroupStatistics (Arguments optional_args)

Parameters

optional_args

An optional input parameter containing a pointer or a reference to an Arguments array containing a member that is a nested Arguments array with the keyword SkillGroupNumbers. Within this array, each member has a string key of an integer starting with 1 and an integer value that is a skill group number to be enabled. If the parameter is NULL or missing, statistics will be enabled for all skill groups to which the agent belongs.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

The CTIOS server sends SkillGroup statistics in the OnSkillGroupStatisticsUpdated event of the SkillGroup object.

GetAgentState

The GetAgentState method returns the current state of the agent.

Syntax

C++: enumCTIOS_AgentState GetAgentState()
COM: HRESULT GetAgentState /*[in]*/ long *state
VB: GetAgentState () As Long
**Java:**
```java
int GetAgentState()
```

### Parameters

**state**

Output parameter (return parameter in VB) containing the current agent state in the form of one of the values in Table 6-1.

---

**GetAllProperties**


---

**GetElement**


---

**GetMonitoredAgent**

The GetMonitoredAgent method returns the agent object that is currently being monitored.

### Syntax

**C++:**
```cpp
CAgent* GetMonitoredAgent()
```

**COM:**
```cpp
HRESULT GetMonitoredAgent(/*[out, retval]*/IAgent **agent)
```

**VB:**
```vbnet
GetMonitoredAgent () As CTIOSCLIENTLib.IAgent
```

**Java:**
```java
Agent GetMonitoredAgent()
```

### Parameters

**agent**

Output parameter (return parameter in VB) that contains a pointer to a pointer to an Agent object containing the currently monitored agent.
Return Value

This method returns the current monitored agent. The Java version returns a null if the agent is not found.

Remarks

Supported for use with IPCC only.

GetMonitoredCall

The GetMonitoredCall method returns the call object that is currently being monitored.

Syntax

C++: `CCall* GetMonitoredCall()`
COM: `HRESULT GetMonitoredCall ([out, retval]*/ICall **call)`
VB: `GetMonitoredCall () As CTIOSCLIENTLib.ICall`
Java: `Call GetMonitoredCall()`

Parameters

call

Output parameter (return parameter in VB) that contains a pointer to a pointer to a Call object containing the currently monitored call.

Return Value

This method returns the current monitored call. The Java version returns a null if the call is not found.

Remarks

Supported for use with IPCC only.
GetNumProperties


GetPropertyType


GetSkillGroups

The GetSkillGroups method allows a client to retrieve a list that contains references to all the skill group objects to which the agent belongs.

Syntax

C++: Arguments & GetSkillGroups();
COM: HRESULT GetSkillGroups (/*[out, retval]*/ VARIANT * pVariantArgs);
VB: GetSkillGroups () As Variant
Java: Arguments GetSkillGroups()

Parameters

None.

Return Value

C++

In C++ the GetSkillGroups method returns an arguments array containing references to CSkillGroup objects.
Each element in the returned arguments array consists of a key/value pair, in which the element key is the Unique Object Id of the skill group object and the value is a reference to a CILRefArg object instance that contains the actual reference to a CSkillGroup object. To retrieve a reference to a skill group object, you need to do something similar to what is shown in the following code example.

```plaintext
Arguments & arSkills = m_Agent->GetSkillGroups();

if(Arguments::IsValidReference(arSkills)){
    for(int nI = 1; nI <= arSkills.NumElements(); nI ++){
        string strUOID = arSkills.GetKey(nI);
        CilRefArg & pRefArg = arSkills.GetValue(strUOID);
        if(Arg::IsValidReference(*pRefArg)){
            CSkillGroup * pSkill = pRefArg->GetValue();
            pRefArg->Release();
            cout << "Skill Object (" << strUOID << ") ;
            cout << "Skill Group Number: " << 
            pSkill->GetValueInt(CTIOS_SKILLGROUPNUMBER);
        }
    }
}
```

**COM**

In COM the GetSkillGroups method returns a pointer to a variant that encapsulates a Safearray where each element is a pointer to an ISkillGroup object. To retrieve references to skill group objects you need to do something similar to what is shown in the following code example.

```plaintext
HRESULT hr = S_OK;
VARIANT varSkills;

VariantInit(&varSkills)
hr  = m_Agent->GetSkillGroups(&varSkills);

if(SUCCEEDED(hr)){
    if(varSkills.vt  == (VT_ARRAY | VT_DISPATCH) ){
        long                  lNumElements = 0;
        SafeArrayGetUBound(varSkills.parray,1,&lNumElements);
        for(long nI = 0; nI < lNumElements; nI ++){
            ISkillGroup * pSkill = NULL;
```
hr=SafeArrayGetElement(varSkills.parray,&nI,&pSkill);
if(SUCCEEDED(hr)){
    int nSkillGrpNumber = 0;
    VARIANT vPropKey;
    VariantInit(&vPropKey);
    vPropKey.vt = VT_BSTR;
    vPropKey.bstr = OLESTR("SkillGroupNumber");
    pSkill->GetValueInt(vPropKey,&nSkillGrpNumber);
    pSkill->Release();
    VariantClear(&vPropKey);
}
}

VB

In VB the GetSkillGroups method returns a variant array where each element is a
reference to a CTIOSClientLib.SkillGroup object.

To retrieve references to skill group objects you need to do something similar to
what is shown in the following code example.

Dim obSkill As CTIOSClientLib.SkillGroup
Dim arSkills As Variant
Dim lNumElements as Long
arSkills = m_Agent.GetSkillGroups()
lNumElements = UBound(arSkills,1)
For nI = 0 to lNumElements
    Set obSkill = arSkills(nI)
    Print "SkillGroup" & obSkill.GetValueString(CStr("UniqueObjectId")) & _
    "Skill Group Number: " & obSkill.GetValueInt(CStr("SkillGroupNumber"))
Next

GetValue Methods

See Chapter 7, “CtiOs Object.” for descriptions of the GetValue, GetValueInt,
GetValueArray, and GetValueString methods.
IsAgent

The IsAgent method determines whether the AgentMode connection is for an agent rather than a supervisor.

Syntax

C++:    bool IsAgent()
COM:    HRESULT IsAgent (VARIANT_BOOL *bIsAgent)
VB:    IsAgent () As Boolean
Java:    boolean IsAgent()

Parameters

bIsAgent

Output parameter (return parameter in VB) that returns true if the current AgentMode connection is for an agent and false if it is for a supervisor.

Return Values

If the current session is in agent mode it returns true, otherwise it returns false.

IsSupervisor

The IsSupervisor method checks if the current mode of the session is supervisor mode.

Syntax

C++:    bool IsSupervisor()
COM:    HRESULT IsSupervisor (VARIANT_BOOL * bIsSupervisor)
VB:    IsSupervisor () As Boolean
Java:    boolean IsSupervisorMode()
Parameters

bIsSupervisor

Output parameter (return parameter in VB) that returns true if the current AgentMode connection is for a supervisor and false if it is for an agent.

Return Values

If the current session is in supervisor mode it returns true, otherwise it returns false.

Login

The Login method performs a login to the ACD (if supported). Generally, the minimum parameters required to log into an ACD are AgentID and AgentInstrument. Often, based on customer configuration, the minimum requirements include an ACD password (AgentPassword). Some switches require PositionID in place of (or in addition to) AgentInstrument. Optional arguments include Extension or AgentWorkMode.

Syntax

C++:
```
virtual int Login(Arguments & args);
```

COM:
```
HRESULT Login ( /*[in]*/ IArguments * pVariantArgs, /*[out]*/
int * errorcode );
```

VB:
```
Login (args As CTIOSCLIENTLib.IArguments) As Long
```

Java:
```
int Login(Arguments args)
```

Input Parameters

args

Arguments array that contains the login parameters listed in Table 9-4.
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”
Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

If the Login request is successful, it returns a CIL_OK CtiOs_Enums.CilError code. In addition, the requesting client should expect an AgentStateChange event if the request is successful with an Arguments member with keyword “AgentState” and value of the agent’s current state. (See GetAgentState for possible values.)

If the Login request is unsuccessful, the client will receive an OnControlFailureConf event and the request will return one of the following CtiOs_Enums.CilError codes:

- E_CTIOS_INVALID_SESSION -- either the agent is not associated with the session or the session is not connected.
- E_CTIOS_INVALID_ARGUMENT -- null or invalid arguments were provided.
- E_CTIOS_LOGIN_INCONSISTENT_ARGUMENTS -- Login request argument values for AgentId and/or PeripheralID do not match the values that were set by SetAgent() prior to the Login request.

Logout

The Logout method logs the agent out of the ACD. If the ACD configuration requires or supports other parameters, these can be passed in as logout parameters. Examples are AgentPassword (required by Alcatel for Logout) or logout reason codes (supported on Avaya Definity ECS, IPCC).

Syntax

C++: int Logout (Arguments& args)
COM: HRESULT Logout ([in]/ IArguments args, [out,retval]/ int * errorcode)
VB: Logout (args As CTIOSCLIENTLib.IArguments) As Long
Java: int Logout (Arguments args)
Input Parameters

args

Input parameter in the form of an Arguments array that contains the Logout parameters listed in Table 9-5.

Table 9-5  Logout Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventReasonCode</td>
<td>INT</td>
<td>Reason for logging out. Required for IPCC, optional for all other switches.</td>
</tr>
<tr>
<td>AgentPassword</td>
<td>STRING, maximum length 12</td>
<td>The agent’s password.</td>
</tr>
<tr>
<td>NumSkillGroups</td>
<td>INT</td>
<td>The number of Skill Groups that the agent is currently associated with, up to a maximum of 20.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The number of an agent skill group associated with the agent.</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>INT</td>
<td>The priority of an agent skill group associated with the agent.</td>
</tr>
<tr>
<td>AgentID (optional)</td>
<td>STRING, maximum length 12</td>
<td>The agent’s login ID.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING, maximum length 12</td>
<td>The agent’s instrument number.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM Peripheral ID of the ACD the agent is attached to.</td>
</tr>
</tbody>
</table>

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”
Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

If the request is successful, the client should receive an OnAgentStateChange event with an Arguments member with keyword “AgentState” and value eLogout. If it is unsuccessful, the client should receive an OnControlFailureConf event. The client should also receive an OnPreLogout event before the OnAgentStateChange event, and an OnPostLogout event afterwards.

MakeCall

The MakeCall method initiates a call to a device or agent. The simplest form of the request requires only a DialedNumber.

Syntax

C++: int MakeCall (Arguments& args)
COM: HRESULT MakeCall /*in*/ IArguments *args, /*out,retval*/ int *errorcode)
VB: MakeCall (args As CTIOSCLIENTLib.IArguments) As Long
Java: int MakeCall(Arguments args)

Input Parameters

args

Input parameter in the form of an Arguments array that contains the MakeCall parameters listed in Table 9-6.
### Table 9-6  MakeCall Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DialedNumber (required)</td>
<td>STRING, maximum length 40</td>
<td>The number to be dialed to establish the new call.</td>
</tr>
<tr>
<td>PeripheralID (optional)</td>
<td>INT</td>
<td>The ICM Peripheral ID of the ACD the agent is attached to.</td>
</tr>
<tr>
<td>AgentInstrument (optional)</td>
<td>STRING, maximum length 12</td>
<td>The agent’s instrument number.</td>
</tr>
<tr>
<td>CallPlacementType (optional)</td>
<td>STRING, maximum length 40</td>
<td>A value specifying how the call is to be placed identified in Table 9-7.</td>
</tr>
<tr>
<td>CallMannerType (optional)</td>
<td>INT</td>
<td>A value specifying additional call processing options identified in Table 9-8.</td>
</tr>
<tr>
<td>AlertRings (optional)</td>
<td>INT</td>
<td>The maximum amount of time that the call’s destination will remain alerting, specified as an approximate number of rings. A zero value indicates that the peripheral default (typically 10 rings) should be used.</td>
</tr>
<tr>
<td>CallOption (optional)</td>
<td>INT</td>
<td>A value from Table 9-9 specifying additional peripheral-specific call options.</td>
</tr>
<tr>
<td>FacilityType (optional)</td>
<td>INT</td>
<td>A value from Table 9-10 indicating the type of facility to be used.</td>
</tr>
<tr>
<td>AnsweringMachine (optional)</td>
<td>INT</td>
<td>A value from Table 9-11 specifying the action to be taken if the call is answered by an answering machine.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Priority (optional)</td>
<td>BOOL</td>
<td>This field should be set to TRUE if the call should receive priority handling.</td>
</tr>
<tr>
<td>PostRoute (optional)</td>
<td>BOOL</td>
<td>When this field is set to TRUE, the Post-Routing capabilities of the ICM are to be used to determine the new call destination.</td>
</tr>
<tr>
<td>UserToUserInfo (optional)</td>
<td>STRING, maximum length 40</td>
<td>The ISDN user-to-user information.</td>
</tr>
<tr>
<td>CallVariable1 (optional)</td>
<td>STRING, maximum length 40</td>
<td>Call variable data that should be set in the new call in place of the corresponding data in the active call.</td>
</tr>
<tr>
<td>CallVariable10 (optional)</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>ECC (optional)</td>
<td>ARGUMENTS</td>
<td>ECC data that should be set in the new call in place of the corresponding data in the active call.</td>
</tr>
<tr>
<td>CallWrapupData (optional)</td>
<td>STRING, maximum length 40</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>FacilityCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>A trunk access code, split extension, or other data needed to access the chosen facility.</td>
</tr>
<tr>
<td>AuthorizationCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>An authorization code needed to access the resources required to initiate the call.</td>
</tr>
<tr>
<td>AccountCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>A cost-accounting or client number used by the peripheral for charge-back purposes.</td>
</tr>
</tbody>
</table>
### Table 9-7  CallPlacementType Values

<table>
<thead>
<tr>
<th>CallPlacementType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPT_UNSPECIFIED</td>
<td>Use default call placement.</td>
<td>0</td>
</tr>
<tr>
<td>CPT_LINE_CALL</td>
<td>An inside line call.</td>
<td>1</td>
</tr>
<tr>
<td>CPT_OUTBOUND</td>
<td>An outbound call.</td>
<td>2</td>
</tr>
<tr>
<td>CPT_OUTBOUND_NO_ACCESS_CODE</td>
<td>An outbound call that will not require an access code.</td>
<td>3</td>
</tr>
<tr>
<td>CPT_DIRECT_POSITION</td>
<td>A call placed directly to a specific position.</td>
<td>4</td>
</tr>
<tr>
<td>CPT_DIRECT_AGENT</td>
<td>A call placed directly to a specific agent.</td>
<td>5</td>
</tr>
<tr>
<td>CPT_SUPERVISOR_ASSIST</td>
<td>A call placed to a supervisor for call handling assistance.</td>
<td>6</td>
</tr>
</tbody>
</table>

### Table 9-8  CallMannerType Values

<table>
<thead>
<tr>
<th>CallMannerType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMT_UNSPECIFIED</td>
<td>Use default call manner.</td>
<td>0</td>
</tr>
<tr>
<td>CMT_POLITE</td>
<td>Attempt the call only if the originating device is idle.</td>
<td>1</td>
</tr>
<tr>
<td>CMT_BELLIGERENT</td>
<td>The call should always be attempted, disconnecting any currently active call.</td>
<td>2</td>
</tr>
<tr>
<td>CMT_SEMI_POLITE</td>
<td>Attempt the call only if the originating device is idle or is receiving dial tone.</td>
<td>3</td>
</tr>
</tbody>
</table>

### Table 9-9  CallOption Values

<table>
<thead>
<tr>
<th>CallOption</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPT_UNSPECIFIED</td>
<td>No call options specified, use defaults.</td>
<td>0</td>
</tr>
<tr>
<td>COPT_CALLING_AGENT_ONLINE</td>
<td>Attempt the call only if the calling agent is “online” (available to interact with the destination party).</td>
<td>1</td>
</tr>
</tbody>
</table>
COPT_CALLING_AGENT_RESERVED
Attempt the call only if ACDNR on the calling agent’s set is activated (DMS-100).

COPT_CALLING_AGENT_NOT_RESERVED
Attempt the call only if ACDNR on the calling agent’s set is not activated (DMS-100).

COPT_CALLING_AGENT_BUZZ_BASE
Causes a buzz to be applied to the base of the telephone set as the call is initiated (DMS-100).

COPT_CALLING_AGENT_BEEP_HSET
Causes a tone to be applied to the agent headset as the call is initiated (DMS-100).

COPT_SERVICE_CIRCUIT_ON
Causes a call classifier to be applied to the call (DEFINITY ECS).

### Table 9-10 FacilityType Values

<table>
<thead>
<tr>
<th>FacilityType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT_UNSPECIFIED</td>
<td>Use default facility type.</td>
<td>0</td>
</tr>
<tr>
<td>FT_TRUNK_GROUP</td>
<td>Facility is a trunk group.</td>
<td>1</td>
</tr>
<tr>
<td>FT_SKILL_GROUP</td>
<td>Facility is a skill group or split.</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 9-11 AnsweringMachine Values

<table>
<thead>
<tr>
<th>AnsweringMachine</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM_UNSPECIFIED</td>
<td>Use default behavior.</td>
<td>0</td>
</tr>
<tr>
<td>AM_CONNECT</td>
<td>Connect call to agent when call is answered by an answering machine.</td>
<td>1</td>
</tr>
<tr>
<td>AM_DISCONNECT</td>
<td>Disconnect call when call is answered by an answering machine.</td>
<td>2</td>
</tr>
<tr>
<td>AM_NONE</td>
<td>Do not use answering machine detection.</td>
<td>3</td>
</tr>
</tbody>
</table>
errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value
Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks
If the request is successful, the client should receive one or more of the following call related events:

- OnCallBegin
- OnCallDelivered
- OnServiceInitiated
- OnCallOriginated
- OnCallReachedNetwork
If the request is unsuccessful, the client should receive an OnControlFailureConf event.

MakeEmergencyCall
The MakeEmergencyCall method makes an emergency call to the Agent’s supervisor.
Syntax

C++:
```cpp
int MakeEmergencyCall ()
int MakeEmergencyCall (Arguments& reserved_args)
```

COM:
```com
HRESULT MakeEmergencyCall (/*[in, optional]*/ IArguments reserved_args, /*[out, retval]*/ int * errorcode)
```

VB:
```vb
MakeEmergencyCall () As Long
MakeEmergencyCall (reserved_args As CTIOSCLIENTLib.IArguments) As Long
```

Java:
```java
int MakeEmergencyCall (Arguments args)
```

Parameters

reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

The MakeEmergencyCall request is very similar to the RequestSupervisorAssist request in the following two ways:

- Both requests place a call from the requesting agent to a supervisor and are routed employing the same script. A typical script might attempt to route the call to the primary supervisor first (if logged in and in available state) and, failing that, to route the call to a skillgroup that all supervisors belong to.

- Both call requests can be configured through ICM Agent Desk Settings to be performed via a single step conference or consult call. If the consult method is chosen, the agent can complete the established consult call as a transfer or conference.

However, these two requests have the following important differences:
- Only Emergency calls are able to be recorded, if so configured in the ICM Agent Desk Settings.
- The calls are reported separately in ICM reporting.

Having these two separate requests gives a site some flexibility in implementing supervisor help for its agents, instructing agents to use one for certain cases and the other for different situations. For example, agents can be trained to click the Emergency button if the customer has more than $1,000,000 in an account, and otherwise to click the Supervisor Assist button. The Supervisor will be able to differentiate the agent’s request by noting the CallType. In addition, customers can set up one type of call to employ a consult call, while using the other if the agent should be added to the conference immediately through the single step configuration.

The MakeEmergencyCall request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Cisco IPCC only). The client issuing the request receives an OnEmergencyCall event when the request reaches an available supervisor. If the request is unsuccessful the client receives an OnControlFailureConf event.

**QueryAgentState**

The QueryAgentState method lets a client retrieve the current state of the agent.

**Syntax**

**C++:**
```c++
int QueryAgentState (Arguments & args );
```

**COM:**
```c
HRESULT QueryAgentState ( /*[in]*/ IArguments * args,
/*[out,retval]*/ int * errorcode );
```

**VB:**
```vb
QueryAgentState (ByVal args as CTIOSCLIENTLIB.IArguments) As Long
```

**Java:**
```java
int QueryAgentState (Arguments args);```

**Input Parameters**

`args`

Arguments array that contains the parameters listed in Table 9-12.
Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

If the request is successful, the client should receive an OnQueryAgentStateConf event. If it is unsuccessful, the client should receive an OnControlFailureConf event.

ReportBadCallLine

The ReportBadCallLine method informs the CTI OS server of the poor quality of the agent’s line. A note of this is recorded in the database.

Syntax

```c++
int ReportBadCallLine ()
int ReportBadCallLine (Arguments& reserved_args)
```  
```com
HRESULT ReportBadCallLine ([in, optional]* IArguments reserved_args, /* [out, retval]* int * errorcode)
```  
```vb
ReportBadCallLine () As Long
```  
```java
int ReportBadCallLine (Arguments args)
```

Parameters

reserved_args

Not currently used, reserved for future use.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent ID</td>
<td>STRING</td>
<td>Agent’s login ID.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>Agent’s instrument number.</td>
</tr>
</tbody>
</table>
errorCode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values
Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

RequestAgentTeamList
The RequestAgentTeamList method is called by a supervisor to make a request to the CTI OS server for a list of agents in the supervisor’s team.

Syntax

C++:
int RequestAgentTeamList ()
int RequestAgentTeamList (Arguments& reserved_args)

COM:
HRESULT RequestAgentTeamList (/*[in, optional]*/ IArguments reserved_args, /*[out, retval]*/ int * errorcode)

VB:
RequestAgentTeamList () As Long

Java:
int RequestAgentTeamList ()
    int RequestAgentTeamList (Arguments args)

Parameters

reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value
Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
Supported for use with IPCC only.

If this request is successful, the CTIOS server sends a separate OnNewAgentTeamMember event for each agent in the supervisor’s team. If this request is unsuccessful, the client receives an OnControlFailureConf event.

**RequestSupervisorAssist**

The RequestSupervisorAssist method allows the agent to call an available supervisor for assistance.

**Syntax**

**C++:**

```cpp
virtual int RequestSupervisorAssist();
int RequestSupervisorAssist (Arguments& reserved_args)
```

**COM:**

```cpp
HRESULT RequestSupervisorAssist (/*[in, optional]*/ IArguments reserved_args, /* [out, retval]*/ int * errorcode)
```

**VB:**

```vb
RequestSupervisorAssist () As Long
```

**Java:**

```java
int RequestSupervisorAssist(Arguments args);
```

**Parameters**

- **reserved_args**
  
  Not currently used, reserved for future use.

- **errorcode**

  An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

**Return Values**

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
Remarks

Supported for use with IPCC only. See “MakeEmergencyCall” for more information.

SendChatMessage

The SendChatMessage method sends asynchronous chat-like messages between CTI clients. Users can specify a distribution of one or more clients, and attach a text message.

Syntax

C++: `int SendChatMessage (Arguments& args)`

COM: `HRESULT SendChatMessage (/*[in]*/ IArguments *args, /*[out,retval]*/ int * errorcode)`

VB: `SendChatMessage (args As CTIOSCLIENTLib.IArguments) As Long`

Java: `int SendChatMessage(Arguments args)`

Input Parameters

`args`

Input parameter in the form of an Arguments array that contains one or more of the SendChatMessage parameters listed in Table 9-13.

Table 9-13 SendChatMessage Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution (required)</td>
<td>STRING</td>
<td>Currently the only supported value is “agent”.</td>
</tr>
<tr>
<td>Target (optional)</td>
<td>STRING</td>
<td>When the Distribution is set to DistributeToAgent, this field must be included with the AgentID of the intended recipient.</td>
</tr>
<tr>
<td>Message (optional)</td>
<td>STRING</td>
<td>The text of the user message. Maximum message size is 255 bytes.</td>
</tr>
</tbody>
</table>
errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

The recipient receives the message via the OnChatMessage event.

SetAgentState

The SetAgentState method requests a new agent state. Login and Logout are valid agent states and can be set using the SetAgentState method as well as the Login and Logout methods.

Syntax

C++: int SetAgentState(Arguments& args)
COM: HRESULT SetAgentState (/*[in]*/ IArguments *args, /*[out,retval]*/ int * errorcode)
VB: SetAgentState (args As CTIOSCLIENTLib.IArguments) As Long
Java: int SetAgentState(Arguments args)

Input Parameters

args

Input parameter in the form of an Arguments array that contains one or more of the SetAgentState parameters listed in Table 9-14.
### Table 9-14  *SetAgentState Parameters*

<table>
<thead>
<tr>
<th><strong>Keyword</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentState (required)</td>
<td>INT</td>
<td>One of the values in Table 6-1 representing the current state of the associated agent.</td>
</tr>
<tr>
<td>AgentID (required)</td>
<td>STRING, maximum length 12</td>
<td>The agent’s login ID.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING, maximum length 12</td>
<td>The agent’s instrument number. Optional if Agent Extension is provided.</td>
</tr>
<tr>
<td>PositionID</td>
<td>STRING, maximum length 12</td>
<td>Required for Alcatel only.</td>
</tr>
<tr>
<td>AgentPassword (optional)</td>
<td>STRING, maximum length 12</td>
<td>The agent’s password.</td>
</tr>
<tr>
<td>AgentWorkMode (optional)</td>
<td>INT</td>
<td>A value representing the desired work mode of the agent. Used by Avaya DEFINITY ECS with default value of ManualIn.</td>
</tr>
<tr>
<td>NumSkillGroups (optional)</td>
<td>INT</td>
<td>The number of Skill Groups that the agent is currently associated with, up to a maximum of 20.</td>
</tr>
<tr>
<td>EventReasonCode (optional)</td>
<td>INT</td>
<td>Reason for logging out. Required for IPCC, optional for all other switches.</td>
</tr>
<tr>
<td>PeripheralID (optional)</td>
<td>INT</td>
<td>The ICM Peripheral ID of the ACD the agent is attached to.</td>
</tr>
<tr>
<td>SkillGroupNumber (optional)</td>
<td>INT</td>
<td>The number of an agent skill group associated with the agent.</td>
</tr>
<tr>
<td>SkillGroupPriority (optional)</td>
<td>INT</td>
<td>The priority of an agent skill group associated with the agent.</td>
</tr>
</tbody>
</table>
errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

A successful request will result in an OnAgentStateChanged event. It may also result in OnPreLogout, OnPostLogout, and/or OnLogoutFailed events. If this request is unsuccessful, the client receives an OnControlFailureConf event.

SetValue

The SetValue method sets the value of the specified Agent property

Syntax

C++:

```cpp
bool SetValue( string& key, string& value)
bool SetValue( string& keyValuePair)
bool SetValue( string& key, int value)
bool SetValue( const char * key, const char * value)
bool SetValue( const char * keyValuePair)
bool SetValue( const char * key, int value)
```

COM:

```cpp
HRESULT SetValue (/*[in]*/ VARIANT *key, /*[in]*/ VARIANT *value,
/*[out,retval]*/ VARIANT_BOOL * errorcode)
```

VB:

```vb
SetValue (key As Variant, value As Variant) As Long
```

Parameters

key

An input parameter that contains the name of the property whose value you want to set.
value

An input parameter containing the value to be used in setting the specified property.

keyValuePair

An input parameter containing a string in the format “key=value” where key is a property to set and value is the new value.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

COM: Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

All Others: A boolean indicating the success or failure of the method.

Remarks

This method should only be used when first creating a new Agent in preparation for logging in. Therefore, it should be used to set the AgentID, AgentInstrument, AgentPassword, PeripheralID, and AutoLogin only.

StartMonitoringAgent

The StartMonitoringAgent method allows the client, which must be a supervisor, to start monitoring the specified Agent object. This call will cause the supervisor to receive all of the monitored call events (See “IMonitoredCallEvents Interface” in Chapter 6, “Event Interfaces and Events”) for this agent until the supervisor calls StopMonitoringAgent.

Syntax

C++: int StartMonitoringAgent(Arguments& args)

COM: HRESULT StartMonitoringAgent ([in] IArguments * args, /*[out, retval]*/ int * errorcode)

VB: StartMonitoringAgent (args As CTIOSCLIENTLib.IArguments) As Long

Java: int StartMonitoringAgent(Arguments args)
Parameters

args
An input parameter in the form of a pointer to an Arguments array that contains a member with a string value that is the UniqueObjectID of the desired agent. This should be packaged with the keyword “AgentReference”.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Cisco IPCC only).

StartMonitoringAgentTeam

The StartMonitoringAgentTeam method allows the client, which must be a supervisor, to start monitoring the specified agent team. This will cause the supervisor to receive all of the OnMonitorAgentStateChange events for every agent on the specified team.

Syntax

C++: int StartMonitoringAgentTeam (Arguments& args)
COM: HRESULT StartMonitoringAgentTeam ([in] IArguments args,
[out, retval]/ int * errorcode)
VB: StartMonitoringAgentTeam (args as CTIOSCLIENTLib.IArguments) As Long
Java: int StartMonitoringAgentTeam (Arguments args)
Parameters

args
Input parameter in the form of a pointer to an Arguments array that contains a member whose value is an integer TeamID that needs to be monitored. This value should be associated with the keyword “TeamID”.

errorCode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value
Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks
This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Cisco IPCC only).

StartMonitoringAllAgentTeams
The StartMonitoringAllAgentTeams method allows the client, which must be a supervisor, to start monitoring all the agent teams. This will cause the supervisor to receive monitored agent events for all of the agents in the supervisor’s team (see “IMonitoredAgentEvents Interface” in Chapter 6, “Event Interfaces and Events”).

Syntax

C++: int StartMonitoringAllAgentTeams (Arguments& reserved_args)
COM: HRESULT StartMonitoringAllAgentTeams (/*[in, optional]*/
IArguments reserved_args, /*[out, retval]*/ int * errorCode)
VB: StartMonitoringAllAgentTeams ([reserved_args as
CTIOSCLIENTLib.IArguments]) As Long
Java: int StartMonitoringAllAgentTeams (Arguments args)
Parameters

reserved_args
   Not currently used, reserved for future use.
errorcode
   An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Cisco IPCC only).

StartMonitoringCall

Description

The StartMonitoringCall method allows the client, which must be a supervisor, to set the value of the currently monitored call that is used in the SuperviseCall method. Since there is no StopMonitoringCall, to clear the value of the currently monitored call, call this method with an empty args parameter.

Syntax

C++: int StartMonitoringCall(Arguments& args)
COM: HRESULT StartMonitoringCall (/*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode)
VB: StartMonitoringCall (args As CTIOSCLIENTLib.IArguments) As Long
Java: int StartMonitoringCall(Arguments args)
Parameters

args
An input parameter in the form of a pointer to an Arguments array that contains a member with a string value that is the UniqueObjectID of the desired call. This should be packaged with the keyword “CallReference”.

errorCode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value
Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks
This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Cisco IPCC only).

StopMonitoringAgent
The StopMonitoringAgent method allows the client, which must be a supervisor, to stop monitoring the specified Agent object. This will stop all Monitored Call events from being sent to the supervisor.

Syntax

C++: int StopMonitoringAgent (Arguments& args)
COM: HRESULT StopMonitoringAgent (/*[in]*/ IArguments * args,
/*[out,retval]*/ int * errorCode)
VB: StopMonitoringAgent (args As CTIOSCLIENTLib.IArguments) As Long
Java: int StopMonitoringAgent (Arguments args)
Parameters

agent

An input parameter in the form of a pointer to an Arguments array that contains a member with a string value that is the UniqueObjectID of the desired agent. This should be packaged with the keyword “AgentReference”.  

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Cisco IPCC only).

StopMonitoringAgentTeam

The StopMonitoringAgentTeam method allows the client, which must be a supervisor, to stop monitoring the specified agent team.

Syntax

C++:  int StopMonitoringAgentTeam (Arguments& args)
COM:  HRESULT StopMonitoringAgentTeam (/*[in]*/ IArguments args,  
      /*[out,retval]*/ int * errorcode)
VB: StopMonitoringAgentTeam (args as CTIOSCLIENTLib.IArguments) As Long
Java: int StopMonitoringAgentTeam(Arguments args)
Parameters

args
Input parameter in the form of a pointer to an Arguments array that contains a member with an integer value that is the TeamID for the team whose monitoring should stop. This value should be associated with the key “TeamID”.

errorCode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Cisco IPCC only).

StopMonitoringAllAgentTeams

The StopMonitoringAllAgentTeams method allows the client, which must be a supervisor, to stop monitoring all of the agent teams.

Syntax

C++: `int StopMonitoringAllAgentTeams (Arguments& reserved_args)`
COM: `HRESULT StopMonitoringAllAgentTeams (/*[in,optional]*/ IArguments reserved_args, /*[out,retval]*/ int * errorcode)`
VB: `StopMonitoringAllAgentTeams([reserved_args as CTIOSCLIENTLib.IArguments]) As Long`
Java: `int StopMonitoringAllAgentTeams(Arguments args)`
Parameters

reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Cisco IPCC only).

SuperviseCall

The SuperviseCall method allows the client, which must be a supervisor, to perform a supervisory action specified by the args parameter.

Syntax

C++: `int SuperviseCall(Arguments& args)`
COM: `HRESULT SuperviseCall (/*[in]*/ IArguments * args,
/*[out retval]*/  int errorCode)`
VB: `SuperviseCall (args As CTIOSCLIENTLib.IArguments ) As Long`
Java: `int SuperviseCall(Arguments args)`
Input Parameters

args

An input parameter in the form of a pointer to an Arguments array that contains members with string values that are the UniqueObjectIDs of the desired agent and call. These should be packaged with the keywords “AgentReference” and “CallReference” respectively. The third required parameter is one of the following integers representing the desired supervisory action:

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>eSupervisorBargeIn</td>
<td>BargeIn to the specified call of the specified agent.</td>
</tr>
<tr>
<td>4</td>
<td>eSupervisorIntercept</td>
<td>Intercept the specified call of the specified agent.</td>
</tr>
</tbody>
</table>

This is packaged with the keyword “SupervisoryAction”.

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Cisco IPCC only).

A BargeIn action is very similar to a Single Step Conference where the agent is the conference controller. As such, only this agent is able to add other parties to the conference; the supervisor will not be able to do this.

An Intercept can only be performed by a supervisor who has already performed a BargeIn. The Intercept simply hangs up the original agent, leaving only the customer and the supervisor talking.
Call Object

The Call object provides developers using the CTIOS Client Interface Library with an interface to Call behavior and control. The Call object exposes methods to perform all call behaviors, such as answering, hanging up, or transferring a call. The Call object represents one call connection of a call. For a call between two parties there are two call connections, and thus there would be two distinct CIL Call objects.

The object stores specific call information as properties, including the ICMEnterpriseUniqueID, ANI, DNIS, Call variables, and ExpandedCallContext variables. The Call object is created in response to call events received at the CIL. The Call object’s properties and state will be updated throughout the lifetime of the call connection.

See Chapter 3, “CIL Coding Conventions” for an explanation of accessing Call and ECC variables via the GetValue mechanism.

Current Call Concept

The Client Interface Library uses the concept of a Current Call. The Current Call concept is used by the CTIOS Toolkit as a way for the controls and the application to communicate with each other regarding which call is currently selected and should be the one to act upon. For example, if an agent has a call and receives a new Ringing call, he might select the Talking call on the grid. At this click, CallAppearanceMgr control calls SetCurrentCall() to make this call the Current Call. When the agent clicks the Hold control, this control would call GetCurrentCall() to obtain a call pointer through which to call the Hold() method. The agent might then select the Ringing call, which would again cause the
CallAppearanceMgr control to call SetCurrentCall() to make this new call the current call. Then, when the agent clicks the Answer control, this control would again call GetCurrentCall() to obtain a call pointer through which to call the Answer() method.

If your application uses Cisco’s out-of-the-box button controls (see Chapter 5, “CTI OS ActiveX Softphone Controls”), but not the CallAppearanceMgr grid control, you will need to use SetCurrentCall() and GetCurrentCall() in order for the button controls enable and disable correctly when switching between multiple calls.

Note
The CurrentCall concept does not place any limitations on call control of non-current calls. All of the call behaviors implemented by method calls on the Call object will work on any call object that is available at the CIL, even if it is not the CurrentCall.

Accessing ECC Variables

The Cisco ICM System provides a customer-defined data layout for sending call context data with a call. This mechanism is called Expanded Call Context, or ECC. ECC variables are defined in the ICM Configuration Manager, and are sent between ICM servers as a key-value structure. The mechanism for accessing ECC variables from CTI OS is similar to accessing all other call variables.

To simplify the organization of properties on the Call object, the ECC variables are stored in their own Arguments structure which is nested in the Call object Arguments structure.

Retrieving ECC Variable Values

To retrieve an ECC variable from the Call object, first retrieve the ECC (Arguments) structure from the Call object using GetValueArray with keyword ECC. Then, retrieve the specific ECC variable required by using its name as the keyword to GetValueInt, GetValueArray, or GetValueString, depending on its type. The following is some sample code for C++ without COM:

```cpp
Arguments * pECCData = NULL;
string sMyECCVariable;
```
int nMyECCArrayVariable;

if (pCall->IsValid(CTIOS_ECC))
{
    pCall->GetValueArray(CTIOS_ECC, &pECCData);
    if (pECCData)
    {
        if (pECCData->IsValid("user.MyECC"))
        {
            pECCData->GetValueString("user.MyECC", &sMyECCVariable);
        }
        if (pECCData->IsValid("user.MyArray[2]"))
        {
            pECCData->GetValueInt("user.MyArray[2]", &nMyECCArrayVariable);
        }
        pECCData->Release();
        pECCData = NULL;
    }
}

The same thing in VB would be as follows:

Dim MyECCData As CTIOSARGUMENTSLib.Arguments
Dim MyECCVariable As String
Dim MyECCArrayVariable As Integer

If MyCall.IsValid(CTIOS_ECC) = True Then
    Set MyECCData = MyCall.GetValueArray("CTIOS_ECC")
    If MyECCData.IsValid("user.MyECC") Then
        MyECCVariable = MyECCData.GetValueString("user.MyECC")
    End If
    If MyECCData.IsValid("user.MyArray[2]") Then
        MyECCArrayVariable = MyECCData.GetValueInt("user.MyArray[2]")
    End If
End If
End If
Adding ECC Values

If you want to add ECC values to a call without deleting ones that are already set in the call, retrieve the ECC variables and then add the new ones as shown in C++ without COM:

```
Arguments & RequestArgs = Arguments::CreateInstance();
Arguments * pECCData = NULL;
// presumes that we have a Call object pointer in pCall
if (pCall->IsValid(CTIOS_ECC))
    pCall->GetValueArray(CTIOS_ECC, &pECCData);
else
    Arguments::CreateInstance(&pECCData);

pECCData->AddItem("user.MyECC", "FirstECCVariable");
pECCData->AddItem("user.MyArray[2]", 2222);

RequestArgs.AddItem(CTIOS_ECC, *pECCData);
pCall->SetCallData(RequestArgs);
RequestArgs.Release();
pECCData->Release();
```
Chapter 10 Call Object

Properties

The same thing in VB would be as follows:

```vbnet
Dim MyRequestArgs As New CTIOSARGUMENTSLib.Arguments
Dim MyECCData As CTIOSARGUMENTSLib.Arguments

If MyCall.IsValid(CTIOS_ECC) Then
    Set MyECCData = MyCall.GetValueArray(CTIOS_ECC)
Else
    Set MyECCData = New CTIOSARGUMENTSLib.Arguments
End If

MyECCData.AddItem("user.MyECC", "FirstECCVariable")
MyECCData.AddItem("user.MyArray[2]", 2222)

MyRequestArgs.AddItem("ECC", MyECCData)

MyCall.SetCallData(MyRequestArgs)
```

The same thing in Java would be as follows:

```java
if(Call != null)
{
    Arguments  rArgEcc =  Call.GetValueArray(CTIOS_ECC);
    if(null == rArgEcc)
    {
        rArgEcc = new Arguments();
    }
    rArgEcc.SetValue("user.MyEcc", 22222);
    rArgEcc.SetValue("user.MyArray[3]", "new data");
    Call.SetValue(CTIOS_ECC, rArgEcc);
}
```

Table 10-1 lists the available call object properties.

Note

The data type listed for each keyword is the standardized data type discussed in the section “CTIOS CIL Data Types” in Chapter 3, “CIL Coding Conventions.” See Table 3-1 for the appropriate language specific types for these keywords.
### Table 10-1 Call Object Properties

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANI</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>CallerEnteredDigits</td>
<td>STRING</td>
<td>The digits entered by the caller in response to IVR prompting.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>CallVariable1</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable2</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable3</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable4</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable5</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable6</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable7</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable8</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable9</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable10</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallWrapupData</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>ClassIdentifier</td>
<td>INT</td>
<td>Private; for internal use only.</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>DNIS</td>
<td>STRING</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>ECC</td>
<td>ARGUMENTS</td>
<td>Arguments structure of key-value pairs of ECC variables.</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
</tbody>
</table>
### Table 10-1 Call Object Properties (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeasuredCallQTime</td>
<td>INT</td>
<td>Number of seconds this call was in a local queue before being delivered to the agent.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>INT</td>
<td>The call key created by the ICM. The ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>Router CallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
</tbody>
</table>
Methods

Table 10-2 lists the available call object methods.

Table 10-2 Call Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternate</td>
<td>Places the current call on hold and retrieves a previously held call.</td>
</tr>
<tr>
<td>Answer</td>
<td>Answers a call that is in the alerting or ringing state.</td>
</tr>
<tr>
<td>Clear</td>
<td>Clears a call, dropping all parties to the call.</td>
</tr>
<tr>
<td>ClearConnection</td>
<td>Hangs up a call, leaving other parties in a conference call. If there are</td>
</tr>
<tr>
<td></td>
<td>only two parties on the call it clears the call.</td>
</tr>
<tr>
<td>Conference</td>
<td>Either establishes a three party conference call or adds a new party to an</td>
</tr>
<tr>
<td></td>
<td>existing conference call.</td>
</tr>
<tr>
<td>DumpProperties</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetAllProperties</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetCallContext</td>
<td>Gets data associated with the call other than call and expanded call context (ECC) variables.</td>
</tr>
</tbody>
</table>
Table 10-2 Call Object Methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCallData</td>
<td>Obtains call and expanded call context (ECC) variables.</td>
</tr>
<tr>
<td>GetElement</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetLastError</td>
<td>Returns the last error that occurred on the calling thread.</td>
</tr>
<tr>
<td>GetNumProperties</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetPropertyName</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetPropertyType</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetValue methods</td>
<td>Retrieve a property from the Call object based on the property’s name key.</td>
</tr>
<tr>
<td>Hold</td>
<td>Places a current call on hold.</td>
</tr>
<tr>
<td>IsValid</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>MakeConsultCall</td>
<td>Places a current call on hold and makes a new call.</td>
</tr>
<tr>
<td>Reconnect</td>
<td>Clears the current call and then retrieves a held call.</td>
</tr>
<tr>
<td>Retrieve</td>
<td>Retrieves a held call.</td>
</tr>
<tr>
<td>SetCallData</td>
<td>Sets call and expanded call context (ECC) variables.</td>
</tr>
<tr>
<td>SendDTMFSignal</td>
<td>Requests the ACD to send a sequence of DTMF tones.</td>
</tr>
<tr>
<td>SingleStepConference</td>
<td>Performs a single step conference.</td>
</tr>
<tr>
<td>SingleStepTransfer</td>
<td>Performs a single step transfer.</td>
</tr>
<tr>
<td>Snapshot</td>
<td>Issues a server request to get the current call information, including call data and a list of associated devices and the connection state for the call of each device.</td>
</tr>
<tr>
<td>StartRecord</td>
<td>Starts recording of a call.</td>
</tr>
<tr>
<td>StopRecord</td>
<td>Stops recording of a call.</td>
</tr>
<tr>
<td>Transfer</td>
<td>Transfers a call to a third party.</td>
</tr>
</tbody>
</table>
Arguments Parameters

The following rules apply to the optional_args and reserved_args parameters in Call Object methods:

- In VB, you can ignore these parameters altogether. For example, you can treat the line:
  \[ \text{Answer([reserved_args As IArguments]) As Long} \]
  as follows:
  \[ \text{Answer()} \]

- To ignore these parameters in COM you must send a NULL, as shown:
  \[ \text{Answer (NULL)} \]

Alternate

The Alternate method combines the action of placing a talking call on hold and then retrieving a previously held call at the same device. If there are only two calls at the device, this method may be called via either the current or the held call.

Syntax

**C++:**

\[ \text{int Alternate() } \]
\[ \text{int Alternate(Arguments \\& reserved_args);} \]

**COM:**

\[ \text{HRESULT Alternate /*[in,optional]*/ IArguments *reserved_args,} \]
\[ /*[out, retval]*/ int * errorcode ]; \]

**VB:**

\[ \text{Alternate([reserved_args As IArguments]) As Long} \]

**Java:**

\[ \text{int Alternate(Arguments reserved_args);} \]

Parameters

reserved_args

Not currently used, reserved for future use.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”
Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

For switches that allow more than two calls at a device (for example G3), it is recommended that this request only be made through the desired held call, because of the ambiguity caused by multiple held calls at the device.

The Alternate request must be made via a call whose status is either LCS_CONNECT or LCS_HELD or it will fail.

The following events will be received if this request is successful.

For the call making the Alternate request:
- OnAlternateCallConf event

For the originally current call:
- OnCallHeld event

For the originally held call:
- OnCallRetrieved event

The following events will be received by the call making the Alternate request if this request fails:
- OnControlFailureConf event

Answer

The Answer method answers a call that is in the alerting or ringing state (i.e., call status of LCS_ALERTING).

Syntax

**C++**:

```cpp
int Answer()
int Answer(Arguments & reserved_args)
```

**COM**:

```cpp
HRESULT Answer (/*[in,optional]*/ IArguments *reserved_args,
             {/*[out, retval]*/ int * errorcode })
```

**VB**:

```vb
Answer([reserved_args As IArguments]) As Long
```

**Java**:

```java
int Answer(Arguments reserved_args)
```
Parameters

reserved_args
   Not currently used, reserved for future use.

errocode
   An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

A call may be answered after the OnCallDelivered event has been received. The Answer request must be made via a call whose call status LCS_ALERTING or it will fail.

The following events will be received if this request is successful:
   • OnAnswerCallConf event
   • OnCallEstablished event

The following events will be received if this request fails:
   • OnControlFailureConf event

Clear

The Clear method clears the call and drops all parties to the call.

Syntax

C++: int Clear()
     int Clear(Arguments & reserved_args);

COM: HRESULT Clear /*[in,optional]*/ IArguments *reserved_args,
     /*[out, retval]*/ int * errocode

VB: Clear([reserved_args As IArguments]) As Long

Java: int Clear(Arguments reserved_args);
Parameters

reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

In the case of a multi-party Conference call, calling Clear() will result in all of the parties to the call being hung up. (If this is not the desired behavior, see the ClearConnection method.) Under certain switches the Clear request will be made via a call whose status is LCS_CONNECT or LCS_INITIATE or it will fail. Many other switches will allow the Clear method to be called via a call whose status is LCS_ALERTING or LCS_HOLD. It may never be made via a call whose status is LCS_NULL indicating that it has already been cleared.

The following events will be received if this request is successful:

• OnClearCallConf event
• OnCallCleared event

The following events will be received if this request fails:

• OnControlFailureConf event

Note

The Clear method is not supported on IPCC. Use of the Clear method with IPCC will result in loss of third-party call control. To avoid this error, applications should use the ClearConnection method instead of Clear to hang up a call.
ClearConnection

The ClearConnection method clears a single connection from a call. If there are only two parties to the call, this effectively clears the call, however for a multi-party conference call, only the one connection is dropped.

Syntax

C++: int ClearConnection()
     int ClearConnection(Arguments & reserved_args);

COM: HRESULT ClearConnection(/*[in,optional]*/ IArguments *reserved_args,
                            /*[out, retval]*/ int * errorcode)

VB: ClearConnection([reserved_args As IArguments]) As Long

Java: int ClearConnection(Arguments reserved_args);

Parameters

reserved_args
    Not currently used, reserved for future use.

errorcode
    An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

As with the Clear method, under certain switches the ClearConnection request must be made via a call whose status is LCS_CONNECT or LCS_INITIATE or it will fail. Many other switches allow the Clear method to be called via a call whose status is LCS_ALERTING or LCS_HOLD. It may never be made via a call whose status is LCS_NULL indicating that it has already been cleared.

The following events will be received if this request is successful:

- OnClearConnectionConf event
• OnCallConnectionCleared event
If this is a two party call, these events will be followed by
• OnCallCleared event
The following events will be received if this request fails:
• OnControlFailureConf event

Conference
The Conference method either begins a new conference call or adds an additional call to an existing conference call. When it begins a new conference call, it combines an original two-party call with a two-party consult call (where the two calls have a common party) into a single three party call. Only the common party (which is called the “Conference Controller”) can call this method to make the new conference call. This method may be called on either of the Conference Controller’s calls.

Syntax

C++: int Conference();
   int Conference(Arguments& optional_args)
COM: HRESULT Conference( /*[in, optional]*/ IArguments
   *optional_args, /*[out, retval]*/ int * errorcode )
VB: Conference([optional_args As IArguments]) As Long
Java: int Conference(Arguments optional_args)

Parameters

optional_args
An optional input parameter, which is a pointer or reference to an Arguments array that contains a member with the string value that is the UniqueObjectID of the call to which this call should be conferenced. If this argument is used, it should be added to the Arguments parameter with the keyword of “CallReferenceObjectID”. This would only be necessary in an environment where there are multiple held calls and the request is being made through the talking call. If the request is being made through a specific held call in this scenario, or if there are only two calls at the device, this parameter is unnecessary.
errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

Before making this request, it is necessary for the original call to be in the held state and the consult call to be in the talking state or the request will fail. Therefore, if the calls are alternated (see Alternate), they must be alternated again to return the two calls to their appropriate states.

If there are only two calls at the device, this method may be called via either the current or the held call. For switches which allow more than two calls at a device (for example G3), it is recommended that this request only be made through the desired held call, because of the ambiguity caused by multiple held calls at the device unless the desired held call is indicated by the optional parameter.

The Conference request must be made via a call whose call status is LCS_CONNECT or LCS_HELD or it will fail.

On certain switches (notably IPCC), only the Conference Controller (the party that first initiated the conference call) may add additional parties to an existing conference call.

The following events will be received if this request is successful:

- OnConferenceCallConf event
- OnCallConferenced event

The following events will be received if this request fails:

- OnControlFailureConf event

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”
GetCallContext

The GetCallContext method returns an Arguments array containing the values for call properties other than CallVariables and ECC Variables, such as ANI, DNIS, and the other properties listed in Table 10-3.

Syntax

C++: int GetCallContext(Arguments& args)
COM: HRESULT GetCallContext(/*[out, retval]*/ IArguments ** args)
VB: GetCallContext (CTIOSCLIENTLib.IArguments args)
Java: int GetCallContext()

Parameters

args
An output parameter containing a reference or a pointer to an Arguments array containing any of the members in Table 10-3 that are present in the call.

Return Value

COM: Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”
All Others: A pointer to an Arguments array that, on return, holds name/value pairs from Table 10-3. Any of these parameters included may be accessed from the Arguments array using the associated keyword.

Table 10-3 GetCallContext Arguments Array Contents

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANI</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>CallerEnteredDigits</td>
<td>STRING</td>
<td>The digits entered by the caller in response to IVR prompting.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>CallWrapupData</td>
<td>STRING</td>
<td>Call-related wrapup data.</td>
</tr>
</tbody>
</table>
Table 10-3  GetCallContext Arguments Array Contents (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionCallID</td>
<td>INT</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>DNIS</td>
<td>STRING</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID</td>
<td>STRING</td>
<td>A unique identifier for this contact throughout the enterprise. This can track a single customer contact across multiple sites, e.g., when a call is transferred between agents.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The ICM identifier for the Service to which this call was routed.</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The ACD number of the Service to which this call was routed.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The ICM identifier for the SkillGroup to which this call was routed.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The number of the SkillGroup at the ACD to which this call was routed.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>A unique object ID for the call.</td>
</tr>
<tr>
<td>UserToUserInfo</td>
<td>STRING</td>
<td>The ISDN user-to-user information element.</td>
</tr>
</tbody>
</table>

Remarks

This is simply a convenience method to be called to get all of a call’s non-CallVariable data at one time. If only certain data members are desired, call the appropriate GetValue method for each instead.

GetCallData

The GetCallData method returns the values of CallVariable1 through CallVariable10 and all of the ECC (Extended CallContext) variables.
Syntax

C++:
```cpp
int GetCallData(Arguments& args)
```

COM:
```csharp
HRESULT GetCallData(/*[out,retval]*/ IArguments ** args)
```

VB:
```vbnet
GetCallData (CTIOSCLIENTLib.IArguments args)
```

Java:
```java
int GetCallData()
```

Parameters

`args`

An output parameter containing a reference or a pointer to an Arguments array containing the call data, as described under Remarks.

Return Value

COM: Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

All Others: A pointer to an Arguments array that, on return, holds parameters described under Remarks.

Remarks

This is simply a convenience method to be called to get all of a call’s CallVariables (1 through 10) and ECC Call Variables at one time. If only certain call variables are desired, call the appropriate GetValue method for each instead.

Access the data in the following way:

- To access the values for individual CallVariables from the arguments parameter, use GetValueString with either the keywords of “CallVariable1” through “CallVariable10”.

To access ECC call data, use the following procedure:

- First, get the ECC variables as a whole from the arguments parameter, using GetValueArray with the keyword “ECC”. This will return another Arguments array that is nested in the Arguments array returned from GetCallData.
- To access an individual ECC scalar variable from this Arguments array, use the appropriate GetValueString, GetValueInt, etc. depending on the variable’s type, using the string keyword “user.VariableName”.


• To access an individual ECC array variable from this Arguments array, use the appropriate GetValueString, GetValueInt, etc. depending on the variable’s type, using the string keyword “user.ArrayName[n] where n is a zero based integer that notes the offset in the array.

GetLastError (Java only)

The GetLastError method returns the last error that occurred on the calling thread.

Syntax

int GetLastError()

Returns

An integer object containing the last error that occurred on the calling thread, or null if no such error can be found. See Table 3-2 and Table 3-4 for a list of possible Java CIL error codes.

GetValue Methods


GetValueBoolObj (Java only)

The GetValueBool method retrieves a Boolean with the specified key from the array.

Syntax

Boolean GetValueBoolObj(int iKey)
Parameters

Key

Key ID for the object to be retrieved.

Returns

A Boolean object representation of the contained value or null if error.

GetValueIntObj (Java only)

Gets the contained value as an integer.

Syntax

Integer GetValueIntObj( int iKey)

Parameters

key

Key ID of the value to be retrieved.

Returns

An Integer object containing a 32 bit signed integer value or null if error.

GetValueShortObj (Java only)

Retrieves a 16 bit short with the specified key from the array.

Syntax

Short GetValueShortObj( int iKey)
Short GetValueShortObj( short sKey)
Methods

Parameters

key
   Key ID of the value to be retrieved.

Returns

A Short object containing a 16 bit short value or null if error.

GetValueUIntObj (Java only)

Retrieves a 32 bit unsigned integer with the specified key from the array.

Syntax

Long GetValueUIntObj( int iKey)

Parameters

key
   Key ID of the value to be retrieved.

Returns

A Long object containing the 32 bit unsigned integer value or null if error.

GetValueUShortObj (Java only)

Retrieves a 16 bit unsigned short with the specified key from the array.

Syntax

Integer GetValueUShortObj( int iKey)
Parameters

key

Key ID of the value to be retrieved.

Returns

An Integer object containing the 16 bit unsigned short value or null if error.

Hold

The Hold method holds a current call.

Syntax

C++: int Hold()
    int Hold(Arguments & reserved_args);

COM: HRESULT Hold (/*[in,optional]*/ IArguments *reserved_args,
    (/*[out, retval]*/ int * errorcode )

VB: Hold([reserved_args As IArguments]) As Long

Java: int Hold(Arguments reserved_args);

Parameters

reserved_args

Not currently used, reserved for future use.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
Remarks

The Hold request must be made via a call whose call status is LCS_CONNECT or it will fail.

The following events will be received if this request is successful:
- OnHoldCallConf event
- OnCallHeld event

The following events will be received if this request fails:
- OnControlFailureConf event

MakeConsultCall

The MakeConsultCall method initiates the combined action of placing the associated current call on hold and then making a new call. By default, the call context data (including call variables) of the current call is used to initialize the context data of the new consultation call. The application may override some or all of the original call context in the consultation call by providing the desired values in this request.

The simplest form of the request only requires a dialed number and a consult type. The request may also include optional parameters, as listed in Table 10-4.

Syntax

**C++:** int MakeConsultCall (Arguments& args)

**COM:** HRESULT MakeConsultCall (/*[in]*/ IArguments *args, /*[out, retval]*/ int * errorcode)

**VB:** MakeConsultCall (args As CTIOSCLIENTLib.IArguments, errorcode As Long)

**Java:** int MakeConsultCall(Arguments args)

Input Parameters

*args

An output parameter of either a reference or a pointer to an Arguments array that contains parameters from Table 10-4. Any of these parameters included should be added to the Arguments array using the associated key word.
### Table 10-4  MakeConsultCall Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DialedNumber (required)</td>
<td>STRING, maximum length 40</td>
<td>Dialed number; the number to be dialed to establish the new call.</td>
</tr>
<tr>
<td>ConsultType (required)</td>
<td>INT</td>
<td>A value specifying whether this consult call is in preparation for either a transfer or a conference, as specified in the ConsultType Table.</td>
</tr>
<tr>
<td>CallPlacementType (optional)</td>
<td>STRING, maximum length 40</td>
<td>A value specifying how the call is to be placed identified in Table 10-5.</td>
</tr>
<tr>
<td>CallMannerType (optional)</td>
<td>INT</td>
<td>A value specifying additional call processing options identified in Table 10-6.</td>
</tr>
<tr>
<td>AlertRings (optional)</td>
<td>INT</td>
<td>The maximum amount of time that the call’s destination will remain alerting, specified as an approximate number of rings. A zero value indicates that the peripheral default (typically 10 rings) should be used.</td>
</tr>
<tr>
<td>CallOption (optional)</td>
<td>INT</td>
<td>A value from Table 10-7 specifying additional peripheral-specific call options.</td>
</tr>
<tr>
<td>FacilityType (optional)</td>
<td>INT</td>
<td>A value from Table 10-8 indicating the type of facility to be used.</td>
</tr>
<tr>
<td>AnsweringMachine (optional)</td>
<td>INT</td>
<td>A value from Table 10-9 specifying the action to be taken if the call is answered by an answering machine.</td>
</tr>
</tbody>
</table>
### Table 10-4 MakeConsultCall Parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority (optional)</td>
<td>BOOL</td>
<td>This field should be set to TRUE if the call should receive priority handling.</td>
</tr>
<tr>
<td>PostRoute (optional)</td>
<td>BOOL</td>
<td>When this field is set to TRUE, the Post-Routing capabilities of the ICM determine the new call destination.</td>
</tr>
<tr>
<td>UserToUserInfo (optional)</td>
<td>STRING, maximum length 40</td>
<td>The ISDN user-to-user information.</td>
</tr>
<tr>
<td>CallVariable1 (optional)</td>
<td>STRING, maximum length 40</td>
<td>Call variable data that should be set in the new call in place of the corresponding data in the current call.</td>
</tr>
<tr>
<td>CallVariable10 (optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECC</td>
<td>ARGUMENTS</td>
<td>ECC data that should be set in the new call in place of the corresponding data in the current call.</td>
</tr>
<tr>
<td>CallWrapupData (optional)</td>
<td>STRING, maximum length 40</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>FacilityCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>A trunk access code, split extension, or other data needed to access the chosen facility.</td>
</tr>
<tr>
<td>AuthorizationCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>An authorization code needed to access the resources required to initiate the call.</td>
</tr>
<tr>
<td>AccountCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>A cost-accounting or client number used by the peripheral for charge-back purposes.</td>
</tr>
</tbody>
</table>
### Table 10-5 CallPlacementType Values

<table>
<thead>
<tr>
<th>CallPlacementType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPT_UNSPECIFIED</td>
<td>Use default call placement.</td>
<td>0</td>
</tr>
<tr>
<td>CPT_LINE_CALL</td>
<td>An inside line call.</td>
<td>1</td>
</tr>
<tr>
<td>CPT_OUTBOUND</td>
<td>An outbound call.</td>
<td>2</td>
</tr>
<tr>
<td>CPT_OUTBOUND_NO_ACCESS_CODE</td>
<td>An outbound call that will not require an access code.</td>
<td>3</td>
</tr>
<tr>
<td>CPT_DIRECT_POSITION</td>
<td>A call placed directly to a specific position.</td>
<td>4</td>
</tr>
<tr>
<td>CPT_DIRECT_AGENT</td>
<td>A call placed directly to a specific agent.</td>
<td>5</td>
</tr>
<tr>
<td>CPT_SUPERVISOR_ASSIST</td>
<td>A call placed to a supervisor for call handling assistance.</td>
<td>6</td>
</tr>
</tbody>
</table>

### Table 10-6 CallMannerType Values

<table>
<thead>
<tr>
<th>CallMannerType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMT_UNSPECIFIED</td>
<td>Use default call manner.</td>
<td>0</td>
</tr>
<tr>
<td>CMT_POLITE</td>
<td>Attempt the call only if the originating device is idle.</td>
<td>1</td>
</tr>
<tr>
<td>CMT_BELLIGERENT</td>
<td>The call should always be attempted, disconnecting any currently active call.</td>
<td>2</td>
</tr>
<tr>
<td>CMT_SEMI_POLITE</td>
<td>Attempt the call only if the originating device is idle or is receiving dial tone.</td>
<td>3</td>
</tr>
</tbody>
</table>

### Table 10-7 CallOption Values

<table>
<thead>
<tr>
<th>CallOption</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPT_UNSPECIFIED</td>
<td>No call options specified, use defaults.</td>
<td>0</td>
</tr>
<tr>
<td>COPT_CALLING_AGENT_ONLINE</td>
<td>Attempt the call only if the calling agent is “online” (available to interact with the destination party).</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 10-7  CallOption Values (continued)

<table>
<thead>
<tr>
<th>CallOption Values</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPT_CALLING_AGENT_RESERVED</td>
<td>Attempt the call only if ACDNR on the calling agent’s set is activated (DMS-100).</td>
<td>2</td>
</tr>
<tr>
<td>COPT_CALLING_AGENT_NOT_RESERVED</td>
<td>Attempt the call only if ACDNR on the calling agent’s set is not activated (DMS-100).</td>
<td>3</td>
</tr>
<tr>
<td>COPT_CALLING_AGENT_BUZZ_BASE</td>
<td>Causes a buzz to be applied to the base of the telephone set as the call is initiated (DMS-100).</td>
<td>4</td>
</tr>
<tr>
<td>COPT_CALLING_AGENT_BEEP_HSET</td>
<td>Causes a tone to be applied to the agent headset as the call is initiated (DMS-100).</td>
<td>5</td>
</tr>
<tr>
<td>COPT_SERVICE_CIRCUIT_ON</td>
<td>Causes a call classifier to be applied to the call (DEFINITY ECS)</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 10-8  FacilityType Values

<table>
<thead>
<tr>
<th>FacilityType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT_UNSPECIFIED</td>
<td>Use default facility type.</td>
<td>0</td>
</tr>
<tr>
<td>FT_TRUNK_GROUP</td>
<td>Facility is a trunk group.</td>
<td>1</td>
</tr>
<tr>
<td>FT_SKILL_GROUP</td>
<td>Facility is a skill group or split.</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 10-9  AnsweringMachine Values

<table>
<thead>
<tr>
<th>AnsweringMachine</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM_UNSPECIFIED</td>
<td>Use default behavior.</td>
<td>0</td>
</tr>
<tr>
<td>AM_CONNECT</td>
<td>Connect call to agent when call is answered by an answering machine.</td>
<td>1</td>
</tr>
<tr>
<td>AM_DISCONNECT</td>
<td>Disconnect call when call is answered by an answering machine.</td>
<td>2</td>
</tr>
<tr>
<td>AM_NONE</td>
<td>Do not use answering machine detection.</td>
<td>3</td>
</tr>
</tbody>
</table>
errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

The MakeConsultCall request must be made via a call whose call status is LCS_CONNECT or it will fail. Calling MakeConsultCall successfully will result in the same events as a successful MakeCall called on the agent.

The following events will be received if this request is successful.

For the call making the MakeConsultCallRequest:

- OnMakeConsultCallConf event
- OnCallHeld event

For the newly created outgoing consult call:

- OnBeginCall event
- OnServiceInitiated event
- OnCallOriginated event
- OnCallDelivered event

For the new connection that is ringing as a result of the consult call:

<table>
<thead>
<tr>
<th>AnsweringMachine</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM_NONE_NO_MODEM</td>
<td>Do not use answering machine detection, but disconnect call if answered by a modem.</td>
<td>4</td>
</tr>
<tr>
<td>AM_CONNECT_NO_MODEM</td>
<td>Connect call when call is answered by an answering machine, disconnect call if answered by a modem.</td>
<td>5</td>
</tr>
</tbody>
</table>
OnBeginCall event
OnCallDelivered event

The following events will be received if this request fails:
OnControlFailureConf event

Reconnect

The Reconnect method combines the action of releasing a current call and then retrieving a previously held call at the same device. If there are only two calls at the device, this method may be called via either the talking or the held call.

Syntax

C++:

```cpp
text
int Reconnect()
int Reconnect(Arguments & reserved_args)
```  

COM:

```csharp
HRESULT Reconnect (/*[in,optional]*/ IArguments * reserved_args,
                    /*[out, retval]*/ int * errorcode )
```  

VB:

```vb
Reconnect([reserved_args As IArguments]) As Long
```  

Java:

```java
int Reconnect(Arguments optional_args)
```  

Parameters

reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
Remarks

For switches which allow more than two calls at a device (for example G3), it is recommended that this request only be made through the desired held call, because of the ambiguity caused by multiple held calls at the device.

The Alternate request must be made via a call whose status is either LCS_CONNECT or LCS_HELD or it will fail.

The following events will be received if this request is successful.

For the call making the Reconnect request:

- OnReconnectCallConf event

For the originally current call:

- OnCallConnectionCleared event
- OnCallCleared event
- OnCallEnd event

For the originally held call:

- OnCallRetrieved event

The following events will be received by the call making the Alternate request if this request fails:

- OnControlFailureConf event

Retrieve

The Retrieve method unholds a held call.

Syntax

**C++**

```cpp
int Retrieve()
int Retrieve(Arguments & reserved_args)
```

**COM**

```cpp
HRESULT Retrieve(/*[in,optional]*/ IArguments *reserved_args,
/*[out, retval]*/ int * errorcode )
```

**VB**

```vb
Retrieve([reserved_args As IArguments]) As Long
```

**Java**

```java
int Retrieve(Arguments optional_args)
```
Parameters

reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

The Retrieve request must be made via a call whose call status is LCS_HELD or it will fail.
The following events will be received if this request is successful:
- OnRetrieveCallConf event
- OnCallRetrieved event
The following events will be received if this request fails:
- OnControlFailureConf event

SendDTMFSignal

The SendDTMFSignal method requests that the ACD to send a sequence of DTMF tones.

Syntax

**C++:** int SendDTMFSignal(Arguments& args)

**COM:** HRESULT SendDTMFSignal ([in] args *arguments, [out, retval]*/ int * errorcode)

**VB:** SendDTMFSignal (args As CTIOSCLIENTLib.IArguments, errorcode As Long)

**Java:** int SendDTMFSignal(Arguments args)
Parameters

args

An input parameter of either a reference or a pointer to an Arguments array containing parameters from Table 10-10. Any of these parameters included should be added to the Arguments array using the associated key word.
errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

**Return Values**

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

**Remarks**

The following event will be received if this request succeeds:

- OnSendDTMFSignalConf

The following event will be received if this request fails:

- OnControlFailureConf
SetCallData

The SetCallData method enables any or all of a call’s CallVariables (1 through 10) and ECC data to be set at one time.

Syntax

C++: int SetCallData(Arguments& args)
COM: HRESULT SetCallData(/*[in]*/ args *arguments, /*[out]*/ int * errorcode)
VB: SetCallData (args As CTIOSCLIENTLib.IArguments, errorcode As Long)
Java: int SetCallData(Arguments args)

Input Parameters

args
An input parameter of either a reference or a pointer to an Arguments array containing parameters described under Remarks for GetCallData.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

You must specify the data for all elements in the Arguments array, not just those elements that you want to change. Failure to do so will cause the unchanged elements to disappear.

The following events will be sent if this request succeeds:

- OnSetCallDataConf
- OnCallDataUpdate
The following event will be sent if this request fails:

- OnControlFailureConf

**SingleStepConference**

The SingleStepConference method initiates a one-step conference without the intermediate consultative call so that when the called party answers, he will be joined in the current call. This method requires a DialedNumber argument. This method is not supported under all switches.

**Syntax**

C++: `int SingleStepConference(Arguments& args)`  
COM: `HRESULT SingleStepConference (IArguments *args, int * errorcode)`  
VB: `SingleStepConference (args As CTIOSCLIENTLib.IArguments, errorcode As Long)`  
Java: `int SingleStepConference(Arguments args)`

**Parameters**

- `args`

  An output parameter of either a reference or a pointer to an Arguments array containing parameters from Table 10-11. Any of these parameters included should be added to the Arguments array using the associated keyword.

**Table 10-11 SingleStepConference Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DialedNumber (required)</td>
<td>STRING, maximum length 40</td>
<td>Dialed number; the number to be dialed to establish the new call.</td>
</tr>
<tr>
<td>CallPlacementType (optional)</td>
<td>STRING, maximum length 40</td>
<td>A value specifying how the call is to be placed identified in Table 10-5.</td>
</tr>
<tr>
<td>CallMannerType (optional)</td>
<td>INT</td>
<td>A value specifying additional call processing options identified in Table 10-6.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AlertRings (optional)</td>
<td>INT</td>
<td>The maximum amount of time that the call’s destination will remain alerting, specified as an approximate number of rings. A zero value indicates that the peripheral default (typically 10 rings) should be used.</td>
</tr>
<tr>
<td>CallOption (optional)</td>
<td>INT</td>
<td>A value from Table 10-7 specifying additional peripheral-specific call options.</td>
</tr>
<tr>
<td>FacilityType (optional)</td>
<td>INT</td>
<td>A value from Table 10-8 indicating the type of facility to be used.</td>
</tr>
<tr>
<td>AnsweringMachine (optional)</td>
<td>INT</td>
<td>A value from Table 10-9 specifying the action to be taken if the call is answered by an answering machine.</td>
</tr>
<tr>
<td>Priority (optional)</td>
<td>BOOL</td>
<td>This field should be set to TRUE if the call should receive priority handling.</td>
</tr>
<tr>
<td>PostRoute (optional)</td>
<td>BOOL</td>
<td>When this field is set to TRUE, the Post-Routing capabilities of the ICM determine the new call destination.</td>
</tr>
<tr>
<td>UserToUserInfo (optional)</td>
<td>STRING, maximum length 40</td>
<td>The ISDN user-to-user information.</td>
</tr>
<tr>
<td>CallVariable1 (optional)</td>
<td>STRING, maximum length 40</td>
<td>Call variable data that should be set in the new call in place of the corresponding data in the current call.</td>
</tr>
<tr>
<td>CallVariable2 (optional)</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>CallVariable10 (optional)</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

The DialedNumber is the only required member necessary in the Arguments parameter. A SingleStepConference request will fail if the call’s status is not LCS_CONNECT.

The following events will be received if this request is successful:

- OnAgentStateChanged event (Hold)
- OnCallHeld event
- OnAgentStateChanged event (Talking)
- OnBeginCall event
Chapter 10      Call Object

Methods

• OnServiceInitiated event
• OnCallOriginated event
• OnCallDelivered event
• OnCallConferenced event
• OnCallEnd event
• ConferenceCallConf event

The following events will be received if this request fails:
• OnControlFailureConf event

SingleStepTransfer

The SingleStepTransfer method initiates a one-step transfer without the intermediate consultative call. When the called party answers the call, the called party will be talking to the party to be transferred and the transferring party will drop out of the call. The method requires a DialedNumber argument.

Syntax

C++: int SingleStepTransfer(Arguments& args)
COM: HRESULT SingleStepTransfer (/*[in]*/ IArguments * args, /*[out, retval]*/ int * errorcode)
VB: SingleStepTransfer (args As CTIOSCLIENTLib.IArguments, errorcode As Long)
Java: int SingleStepTransfer(Arguments args)

Input Parameters

args

An output parameter of either a reference or a pointer to an Arguments array containing parameters from Table 10-11. Any of these parameters included should be added to the Arguments array using the associated keyword.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”
Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Snapshot

The Snapshot method issues a server request to retrieve the current call information. If values are passed in the optional args parameter, the snapshot request will return the server’s current call values for only the requested arguments. Otherwise all call information is returned, including the fields described under GetCallContext and GetCallData. See OnCallDataUpdate in Chapter 6, “Event Interfaces and Events” for more information.

Syntax

C++

```cpp
int Snapshot()
int Snapshot(Arguments & optional_args)
```

COM:

```c
HRESULT Snapshot (/*[in,optional]*/  IArguments * optional_args,    (/*[out, retval]*/ int * errorcode )
```

VB:

```vb
Snapshot([optional_args As IArguments]) As Long
```

Java:

```java
int Snapshot(Arguments rArgs)
```

Parameters

optional_args

An input parameter of either a pointer or a reference to an Arguments array.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
Remarks

The current information about the call will be received in the OnCallDataUpdate event.

The following events will be received if this request is successful:

- OnCallDataUpdate event

The following events will be received if this request fails:

- OnControlFailureConf event

StartRecord

The StartRecord method is used to start recording a call.

Syntax

C++:
```cpp
int StartRecord()
int StartRecord(Arguments & reserved_args);
```

COM:
```cpp
HRESULT StartRecord (/*[in,optional]*/  IArguments
*reserved_args,  (/*[out, retval]*/ int * errorcode )
```

VB:
```vb
StartRecord([reserved_args As IArguments]) As Long
```

Java:
```java
int StartRecord(Arguments reserved_args);
```

Parameters

reserved_args

Not currently used, reserved for future use.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
Remarks

Calling this method causes the CTI Server to forward the request to one or more server applications that have registered the “Cisco:CallRecording” service as described in the Cisco ICM Software CTI Server Message Reference. It will fail if there is no recording server available to CTIServer.

The following events will be received if this request is successful:
- OnStartRecordingConf event

The following events will be received if this request fails:
- OnControlFailureConf event

StopRecord

The StopRecord method is used to stop recording a call.

Syntax

C++:
```cpp
int StopRecord();
int StopRecord(Arguments & reserved_args);
```

COM:
```cpp
HRESULT StopRecord (/*[in,optional]*/  IArguments *reserved_args,
                     /*[out, retval]*/  int * errorcode )
```

VB:
```vb
StopRecord([reserved_args As IArguments]) As Long
```

Java:
```java
int StopRecord(Arguments args);
```

Parameters

reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
Remarks

Calling this method causes the CTIServer to forward the request to the server application with the SessionID received in the OnStartRecordingConf event if non-zero, or if that SessionID is zero, to one or more one or more server applications that have registered the “Cisco:CallRecording” service as described in the Cisco ICM Software CTI Server Message Reference. It will fail if there is no recording server available to CTIServer.

The following events will be received if this request is successful:

- OnStopRecordConf event

The following events will be received by the call making the Alternate request if this request fails:

- OnControlFailureConf event

Transfer

The Transfer method transfers a call to a third party. This method may be called on either the held original call or the current consult call. If the device has only these two calls, the optional parameter is not necessary. At the end of a successful transfer, both of these calls will be gone from the device. See the Conference method for more information.

Syntax

C++:

```cpp
int Transfer();
int Transfer(Arguments& optional_args)
```

COM:

```plaintext
HRESULT Transfer ( [in, optional]  IArguments *optional_args,
  /*[out, retval]*/ int * errorcode )
```

VB:

```vbnet
Transfer([optional_args As IArguments]) As Long
```

Java:

```java
int Transfer(Arguments args)
```

Input Parameters

optional_args

An optional input parameter containing a member with a string value that is the UniqueObjectID of the call that is participating in the transfer. If this argument is used, it should be added to the Arguments parameter with the
keyword of “CallReferenceObjectID”. This would only be necessary in an environment where there are multiple held calls and the request is being made through the current call. If the request is being made through a specific held call in this scenario, or if there are only two calls at the device, this parameter is unnecessary.

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

**Return Values**

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

**Remarks**

Before making this request, it is necessary for the original call to be in the held state and the consult call to be in the talking state or the request will fail. Therefore, if the calls are alternated (See Alternate), they must be alternated again to return the two calls into their appropriate states.

If there are only two calls at the device, this method may be called via either the talking or the held call. For switches which allow more than two calls at a device (for example G3), it is recommended that this request only be made through the desired held call, because of the ambiguity caused by multiple held calls at the device unless the desired held call is indicated by the optional parameter.

The Transfer request must be made via a call whose call status is LCS_CONNECT or LCS_HELD or it will fail.

The following events are received by the transfer initiator if this request is successful:

- OnCallTransferred event
- OnCallEnd event
- OnCallEnd event
- OnAgentStateChange event
- OnTransferCallConf event
The following events will be received if this request fails:

- OnControlFailureConf event
SkillGroup Object

The SkillGroup object provides developers using the CTI OS Client Interface Library with an interface to Skill Group properties and data. The SkillGroup is mainly a representation used for accessing statistics, which can be enabled or disabled via method calls to the SkillGroup object. The SkillGroups are accessible directly from the Session object or the Agent Object.

The SkillGroup object methods can be accessed as follows:

- Via the Agent object inside the Session in Agent mode
- Via the Agent object inside the Session in Monitor mode
- In C++ and Java, via the session object inside the session in Monitor mode when a special filter is set,

See the section xxx in Chapter 4 ccc for related code examples.

Properties

Table 11-1 lists the available SkillGroup properties.

Note

The data type listed for each keyword is the standardized data type discussed in the section “CTIOS CIL Data Types” in Chapter 3, “CIL Coding Conventions.” See Table 3-1 for the appropriate language specific types for these keywords.
To access statistics, first use GetValue on the Skill Group object to obtain the Statistics arguments array, then use GetValue to obtain the desired value.

**Table 11-1  Skill Group Properties**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The number of the skill group from the Peripheral.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>STRING</td>
<td>The ICM SkillGroupID of the SkillGroup, if available.</td>
</tr>
<tr>
<td>SkillGroupName</td>
<td>STRING</td>
<td>The ICM SkillGroupName of the SkillGroup, if available.</td>
</tr>
<tr>
<td>SkillGroupState</td>
<td>INT</td>
<td>Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.</td>
</tr>
<tr>
<td>ClassIdentifier</td>
<td>INT</td>
<td>Value represents skillgroup class.</td>
</tr>
</tbody>
</table>

Not all the statistics values listed in Table 11-1 are present in every system configuration. Whether a particular statistic value is available depends both on the protocol version of CTIServer with which CTI OS connects and on the peripheral on which the agent resides. The statistics listed in Table 11-2 are available in Protocol Version 8 of CTI Server.

One very important real-time skillgroup statistic is the number of calls currently in queue. Previously, this value was typically provided in CallsQNow. However, the number of calls currently in queue is now stored in RouterCallsQNow.

**Statistics**

Table 11-2 lists the available SkillGroup statistics.
### Table 11-2 Skill Group Statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentsLoggedOn</td>
<td>Number of agents that are currently logged on to the skill group.</td>
</tr>
<tr>
<td>AgentsAvail</td>
<td>Number of agents for the skill group in Available state.</td>
</tr>
<tr>
<td>AgentsNotReady</td>
<td>Number of agents in the Not Ready state for the skill group.</td>
</tr>
<tr>
<td>AgentsReady</td>
<td>Number of agents in the Ready state for the skill group.</td>
</tr>
<tr>
<td>AgentsTalkingIn</td>
<td>Number of agents in the skill group currently talking on inbound calls.</td>
</tr>
<tr>
<td>AgentsTalkingOut</td>
<td>Number of agents in the skill group currently talking on outbound calls.</td>
</tr>
<tr>
<td>AgentsTalkingOther</td>
<td>Number of agents in the skill group currently talking on internal (not inbound or outbound) calls.</td>
</tr>
<tr>
<td>AgentsWorkNot Ready</td>
<td>Number of agents in the skill group in the Work Not Ready state.</td>
</tr>
<tr>
<td>AgentsWorkReady</td>
<td>Number of agents in the skill group in the Work Ready state.</td>
</tr>
<tr>
<td>AgentsBusyOther</td>
<td>Number of agents currently busy with calls assigned to other skill groups.</td>
</tr>
<tr>
<td>AgentsReserved</td>
<td>Number of agents for the skill group currently in the Reserved state.</td>
</tr>
<tr>
<td>AgentsHold</td>
<td>Number of calls to the skill group currently on hold.</td>
</tr>
<tr>
<td>AgentsICM Available</td>
<td>Number of agents in the skill group currently in the ICMAvailable state.</td>
</tr>
<tr>
<td>AgentsApplication Available</td>
<td>Number of agents in the skill group currently in the Application Available state.</td>
</tr>
</tbody>
</table>
### Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentsTalkingAutoOut</td>
<td>Number of calls to the skill group currently talking on AutoOut (predictive) calls.</td>
</tr>
<tr>
<td>AgentsTalking Preview</td>
<td>Number of calls to the skill group currently talking on outbound Preview calls.</td>
</tr>
<tr>
<td>AgentsTalking Reservation</td>
<td>Number of calls to the skill group currently talking on agent reservation calls.</td>
</tr>
<tr>
<td>RouterCallsQNow</td>
<td>The number of calls currently queued by the ICM call router for this skill group. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>LongestRouterCallQNow</td>
<td>The queue time, in seconds, of the currently ICM call router queued call that has been queued to the skill group the longest. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>CallsQNow</td>
<td>The number of calls currently queued to the skill group. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>CallsQTimeNow</td>
<td>The total queue time, in seconds, of calls currently queued to the skill group. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>LongestCallQNow</td>
<td>The queue time, in seconds, of the currently queued call that has been queued to the skill group the longest. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
</tbody>
</table>
Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailTimeTo5</td>
<td>Total seconds agents in the skill group were in the Available state.</td>
</tr>
<tr>
<td>LoggedOnTimeTo5</td>
<td>Total time, in seconds, agents in the skill group were logged on.</td>
</tr>
<tr>
<td>NotReadyTimeTo5</td>
<td>Total seconds agents in the skill group were in the Not Ready state.</td>
</tr>
<tr>
<td>AgentOutCallsTo5</td>
<td>Total number of completed outbound ACD calls made by agents in the skill group.</td>
</tr>
<tr>
<td>AgentOutCallsTalk TimeTo5</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCallsTimeTo5</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCallsHeldTo5</td>
<td>The total number of completed outbound ACD calls agents in the skill group have placed on hold at least once.</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeTo5</td>
<td>Total number of seconds outbound ACD calls were placed on hold by agents in the skill group.</td>
</tr>
</tbody>
</table>
### Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandledCallsTo5</td>
<td>The number of inbound ACD calls handled by agents in the skill group.</td>
</tr>
<tr>
<td>HandledCallsTalk TimeTo5</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by agents in the skill group. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>HandledCallsAfter CallTimeTo5</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the skill group.</td>
</tr>
<tr>
<td>HandledCallsTimeTo5</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the skill group. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>IncomingCallsHeldTo5</td>
<td>The total number of completed inbound ACD calls agents in the skill group placed on hold at least once.</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeTo5</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsRcvdTo5</td>
<td>Number of internal calls received by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeTo5</td>
<td>Number of seconds spent on internal calls received by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsHeldTo5</td>
<td>The total number of internal calls agents in the skill group placed on hold at least once.</td>
</tr>
</tbody>
</table>
### Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>InternalCallsHeld TimeTo5</td>
<td>Total number of seconds completed internal calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>AutoOutCallsTo5</td>
<td>Total number of AutoOut (predictive) calls completed by agents in the skill group.</td>
</tr>
<tr>
<td>AutoOutCallsTalk TimeTo5</td>
<td>Total talk time, in seconds, for completed AutoOut (predictive) calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCallsTimeTo5</td>
<td>Total handle time, in seconds, for completed AutoOut (predictive) calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCallsHeld To5</td>
<td>The total number of completed AutoOut (predictive) calls that agents in the skill group have placed on hold at least once.</td>
</tr>
<tr>
<td>AutoOutCallsHeld TimeTo5</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>PreviewCallsTo5</td>
<td>Total number of outbound Preview calls completed by agents in the skill group.</td>
</tr>
</tbody>
</table>
### Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviewCallsTalk TimeTo5</td>
<td>Total talk time, in seconds, for completed outbound Preview calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCallsTime To5</td>
<td>Total handle time, in seconds, for completed outbound Preview calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCallsHeld To5</td>
<td>The total number of completed outbound Preview calls that agents in the skill group have placed on hold at least once.</td>
</tr>
<tr>
<td>PreviewCallsHeld TimeTo5</td>
<td>Total number of seconds outbound Preview calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>ReservationCallsTo5</td>
<td>Total number of agent reservation calls completed by agents in the skill group.</td>
</tr>
<tr>
<td>ReservationCalls TalkTimeTo5</td>
<td>Total talk time, in seconds, for completed agent reservation calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationCalls TimeTo5</td>
<td>Total handle time, in seconds, for completed agent reservation calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>ReservationCalls HeldTo5</td>
<td>The total number of agent reservation calls that agents in the skill group have placed on hold at least once.</td>
</tr>
<tr>
<td>ReservationCalls HeldTimeTo5</td>
<td>Total number of seconds agent reservation calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>BargeInCallsTo5</td>
<td>Total number of supervisor call barge-ins completed in the skill group.</td>
</tr>
<tr>
<td>InterceptCallsTo5</td>
<td>Total number of supervisor call intercepts completed in the skill group.</td>
</tr>
<tr>
<td>MonitorCallsTo5</td>
<td>Total number of supervisor call monitors completed in the skill group.</td>
</tr>
<tr>
<td>WhisperCallsTo5</td>
<td>Total number of supervisor call whispers completed by agents in the skill group.</td>
</tr>
<tr>
<td>EmergencyCallsTo5</td>
<td>Total number of emergency calls completed by agents in the skill group.</td>
</tr>
<tr>
<td>CallsQ5</td>
<td>The number of calls queued to the skill group during the current five-minute. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
</tbody>
</table>
### Table 11-2 Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallsQTime5</td>
<td>The total queue time, in seconds, of calls queued to the skill group during the current five-minute. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>LongestCallQ5</td>
<td>The longest queue time, in seconds, of all calls queued to the skill group during the current five-minute. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>AvailTimeToHalf</td>
<td>Total seconds agents in the skill group were in the Available state.</td>
</tr>
<tr>
<td>LoggedOnTime ToHalf</td>
<td>Total time, in seconds, agents in the skill group were logged on.</td>
</tr>
<tr>
<td>NotReadyTime ToHalf</td>
<td>Total seconds agents in the skill group were in the Not Ready state.</td>
</tr>
<tr>
<td>AgentOutCallsTo Half</td>
<td>Total number of completed outbound ACD calls made by agents in the skill group.</td>
</tr>
<tr>
<td>AgentOutCallsTalk TimeToHalf</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
### Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentOutCallsTimeToHalf</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCallsHeldToHalf</td>
<td>The total number of completed outbound ACD calls agents in the skill group have placed on hold at least once.</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeToHalf</td>
<td>Total number of seconds outbound ACD calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>HandledCallsToHalf</td>
<td>The number of inbound ACD calls handled by agents in the skill group.</td>
</tr>
<tr>
<td>HandledCallsTalk TimeToHalf</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by agents in the skill group. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>HandledCallsAfter CallTimeToHalf</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the skill group.</td>
</tr>
<tr>
<td>HandledCallsTimeToHalf</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the skill group. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
### Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>IncomingCallsHeldToHalf</td>
<td>The total number of completed inbound ACD calls agents in the skill group placed on hold at least once.</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeToHalf</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsRcvdToHalf</td>
<td>Number of internal calls received by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsRcvd TimeToHalf</td>
<td>Number of seconds spent on internal calls received by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsHeldToHalf</td>
<td>The total number of internal calls agents in the skill group placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCallsHeld TimeToHalf</td>
<td>Total number of seconds completed internal calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>AutoOutCallsToHalf</td>
<td>Total number of AutoOut (predictive) calls completed by agents in the skill group.</td>
</tr>
<tr>
<td>AutoOutCallsTalk TimeToHalf</td>
<td>Total talk time, in seconds, for completed AutoOut (predictive) calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
### Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoOutCallsTime ToHalf</td>
<td>Total handle time, in seconds, for completed AutoOut (predictive) calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCallsHeld ToHalf</td>
<td>The total number of completed AutoOut (predictive) calls that agents in the skill group have placed on hold at least once.</td>
</tr>
<tr>
<td>AutoOutCallsHeld TimeToHalf</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>PreviewCallsToHalf</td>
<td>Total number of outbound Preview calls completed by agents in the skill group.</td>
</tr>
<tr>
<td>PreviewCallsTalk TimeToHalf</td>
<td>Total talk time, in seconds, for completed outbound Preview calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCallsTime ToHalf</td>
<td>Total handle time, in seconds, for completed outbound Preview calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviewCallsHeldToHalf</td>
<td>The total number of completed outbound Preview calls that agents in the</td>
</tr>
<tr>
<td></td>
<td>skill group have placed on hold at least once.</td>
</tr>
<tr>
<td>PreviewCallsHeld TimeToHalf</td>
<td>Total number of seconds outbound Preview calls were placed on hold by</td>
</tr>
<tr>
<td></td>
<td>agents in the skill group.</td>
</tr>
<tr>
<td>ReservationCallsToHalf</td>
<td>Total number of agent reservation calls completed by agents in the skill</td>
</tr>
<tr>
<td></td>
<td>group.</td>
</tr>
<tr>
<td>ReservationCalls TalkTimeToHalf</td>
<td>Total talk time, in seconds, for completed agent reservation calls</td>
</tr>
<tr>
<td></td>
<td>handled by agents in the skill group. The value includes the time spent</td>
</tr>
<tr>
<td></td>
<td>from the call being initiated to the time the agent begins after call</td>
</tr>
<tr>
<td></td>
<td>work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>ReservationCalls TimeToHalf</td>
<td>Total handle time, in seconds, for completed agent reservation calls</td>
</tr>
<tr>
<td></td>
<td>handled by agents in the skill group. The value includes the time spent</td>
</tr>
<tr>
<td></td>
<td>from the call being initiated to the time the agent completes after call</td>
</tr>
<tr>
<td></td>
<td>work time for the call. The time includes hold time associated with the</td>
</tr>
<tr>
<td></td>
<td>call.</td>
</tr>
<tr>
<td>ReservationCalls HeldToHalf</td>
<td>The total number of agent reservation calls that agents in the skill</td>
</tr>
<tr>
<td></td>
<td>group have placed on hold at least once.</td>
</tr>
<tr>
<td>ReservationCalls HeldTimeToHalf</td>
<td>Total number of seconds agent reservation calls were placed on hold by</td>
</tr>
<tr>
<td></td>
<td>agents in the skill group.</td>
</tr>
<tr>
<td>BargeInCallsToHalf</td>
<td>Total number of supervisor call barge-ins completed in the skill group.</td>
</tr>
</tbody>
</table>
### Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>InterceptCallsToHalf</td>
<td>Total number of supervisor call intercepts completed in the skill group.</td>
</tr>
<tr>
<td>MonitorCallsToHalf</td>
<td>Total number of supervisor call monitors completed in the skill group.</td>
</tr>
<tr>
<td>WhisperCallsToHalf</td>
<td>Total number of supervisor call whispers completed by agents in the skill group.</td>
</tr>
<tr>
<td>EmergencyCallsToHalf</td>
<td>Total number of emergency calls completed by agents in the skill group.</td>
</tr>
<tr>
<td>CallsQHalf</td>
<td>The number of calls queued to the skill group during the current half hour. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>CallsQTimeHalf</td>
<td>The total queue time, in seconds, of calls queued to the skill group during the current half hour. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>LongestCallQHalf</td>
<td>The longest queue time, in seconds, of all calls queued to the skill group during the current half hour. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>AvailTimeToday</td>
<td>Total seconds agents in the skill group were in the Available state.</td>
</tr>
<tr>
<td>LoggedOnTime Today</td>
<td>Total time, in seconds, agents in the skill group were logged on.</td>
</tr>
<tr>
<td>NotReadyTime Today</td>
<td>Total seconds agents in the skill group were in the Not Ready state.</td>
</tr>
<tr>
<td>AgentOutCalls Today</td>
<td>Total number of completed outbound ACD calls made by agents in the skill group.</td>
</tr>
</tbody>
</table>
### Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentOutCallsTalkTimeToday</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCallsTimeToday</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCallsHeldToday</td>
<td>The total number of completed outbound ACD calls agents in the skill group have placed on hold at least once.</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeToday</td>
<td>Total number of seconds outbound ACD calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>HandledCallsToday</td>
<td>The number of inbound ACD calls handled by agents in the skill group.</td>
</tr>
<tr>
<td>HandledCallsTalkTimeToday</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by agents in the skill group. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>HandledCallsAfterCallTimeToday</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the skill group.</td>
</tr>
</tbody>
</table>
### Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandledCallsTime Today</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the skill group. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>IncomingCallsHeldToday</td>
<td>The total number of completed inbound ACD calls agents in the skill group placed on hold at least once.</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeToday</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsRcvd Today</td>
<td>Number of internal calls received by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsRcvd TimeToday</td>
<td>Number of seconds spent on internal calls received by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsHeld Today</td>
<td>The total number of internal calls agents in the skill group placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCallsHeld TimeToday</td>
<td>Total number of seconds completed internal calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>AutoOutCallsToday</td>
<td>Total number of AutoOut (predictive) calls completed by agents in the skill group.</td>
</tr>
</tbody>
</table>
Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoOutCallsTalk TimeToday</td>
<td>Total talk time, in seconds, for completed AutoOut (predictive) calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCallsTime Today</td>
<td>Total handle time, in seconds, for completed AutoOut (predictive) calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCallsHeld Today</td>
<td>The total number of completed AutoOut (predictive) calls that agents in the skill group have placed on hold at least once.</td>
</tr>
<tr>
<td>AutoOutCallsHeld TimeToday</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>PreviewCallsToday</td>
<td>Total number of outbound Preview calls completed by agents in the skill group.</td>
</tr>
<tr>
<td>PreviewCallsTalk TimeToday</td>
<td>Total talk time, in seconds, for completed outbound Preview calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
### Table 11-2 Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviewCallsTime Today</td>
<td>Total handle time, in seconds, for completed outbound Preview calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCallsHeld Today</td>
<td>The total number of completed outbound Preview calls that agents in the skill group have placed on hold at least once.</td>
</tr>
<tr>
<td>PreviewCallsHeld TimeToday</td>
<td>Total number of seconds outbound Preview calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>ReservationCalls Today</td>
<td>Total number of agent reservation calls completed by agents in the skill group.</td>
</tr>
<tr>
<td>ReservationCalls TalkTimeToday</td>
<td>Total talk time, in seconds, for completed agent reservation calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>ReservationCalls TimeToday</td>
<td>Total handle time, in seconds, for completed agent reservation calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationCalls HeldToday</td>
<td>The total number of agent reservation calls that agents in the skill group have placed on hold at least once.</td>
</tr>
<tr>
<td>ReservationCalls HeldTimeToday</td>
<td>Total number of seconds agent reservation calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>BargeInCallsToday</td>
<td>Total number of supervisor call barge-ins completed in the skill group.</td>
</tr>
<tr>
<td>InterceptCallsToday</td>
<td>Total number of supervisor call intercepts completed in the skill group.</td>
</tr>
<tr>
<td>MonitorCallsToday</td>
<td>Total number of supervisor call monitors completed in the skill group.</td>
</tr>
<tr>
<td>WhisperCallsToday</td>
<td>Total number of supervisor call whispers completed by agents in the skill group.</td>
</tr>
<tr>
<td>EmergencyCalls Today</td>
<td>Total number of emergency calls completed by agents in the skill group.</td>
</tr>
<tr>
<td>CallsQToday</td>
<td>The number of calls queued to the skill. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>CallsQTimeToday</td>
<td>The total queue time, in seconds, of calls queued to the skill group. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>LongestCallQToday</td>
<td>The longest queue time, in seconds, of all calls queued to the skill group. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>AgentsLoggedOn</td>
<td>Number of agents that are currently logged on to the skill group.</td>
</tr>
<tr>
<td>AgentsAvail</td>
<td>Number of agents for the skill group in Available state.</td>
</tr>
</tbody>
</table>
Table 11-2  Skill Group Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentsNotReady</td>
<td>Number of agents in the Not Ready state for the skill group.</td>
</tr>
<tr>
<td>AgentsReady</td>
<td>Number of agents in the Ready state for the skill group.</td>
</tr>
<tr>
<td>AgentsTalkingIn</td>
<td>Number of agents in the skill group currently talking on inbound calls.</td>
</tr>
<tr>
<td>AgentsTalkingOut</td>
<td>Number of agents in the skill group currently talking on outbound calls.</td>
</tr>
<tr>
<td>AgentsTalkingOther</td>
<td>Number of agents in the skill group currently talking on internal (not inbound or outbound) calls.</td>
</tr>
<tr>
<td>AgentsWorkNot Ready</td>
<td>Number of agents in the skill group in the Work Not Ready state.</td>
</tr>
<tr>
<td>AgentsWorkReady</td>
<td>Number of agents in the skill group in the Work Ready state.</td>
</tr>
<tr>
<td>AgentsBusyOther</td>
<td>Number of agents currently busy with calls assigned to other skill groups.</td>
</tr>
<tr>
<td>AgentsReserved</td>
<td>Number of agents for the skill group currently in the Reserved state.</td>
</tr>
<tr>
<td>AgentsHold</td>
<td>Number of calls to the skill group currently on hold.</td>
</tr>
<tr>
<td>AgentsICM Available</td>
<td>Number of agents in the skill group currently in the ICMAvailable state.</td>
</tr>
<tr>
<td>AgentsApplication Available</td>
<td>Number of agents in the skill group currently in the Application Available state.</td>
</tr>
<tr>
<td>AgentsTalkingAutoOut</td>
<td>Number of calls to the skill group currently talking on AutoOut (predictive) calls.</td>
</tr>
</tbody>
</table>
Methods

Table 11-3 lists the SkillGroup object methods.

**Table 11-3 SkillGroup Object Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisableSkillGroupStatistics</td>
<td>Disables skill group statistic messages.</td>
</tr>
<tr>
<td>DumpProperties</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>EnableSkillGroupStatistics</td>
<td>Enables skill group statistic messages.</td>
</tr>
<tr>
<td>GetElement</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetNumProperties</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetPropertyName</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetValue</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetValueInt (C++), GetValueIntObj (Java)</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetValueString</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>IsValid</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>SetValue</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
</tbody>
</table>
Chapter 11      SkillGroup Object

Methods

DisableSkillGroupStatistics

The DisableSkillGroupStatistics method requests that sending real-time statistics to the skillgroup object be stopped.

Syntax

C++:    int DisableSkillGroupStatistics(Arguments & args)
COM:    HRESULT DisableSkillGroupStatistics (IArguments * args, int * errorCode)
VB:     DisableSkillGroupStatistics (args As CTIOSCLIENTLib.IArguments, errorCode As Long)
Java:   int DisableSkillGroupStatistics(Arguments args)

Parameters

args

If this method is called in C++ or Java via the session object in monitor mode with the special filter, the args parameter has two required values for PeripheralId and SkillGroupNumber. See the Remarks section for a code example. Otherwise, this parameter is not used.

errorCode

An output parameter (return parameter in VB) that contains an error code, if any.

Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

The CTI OS server sends skillgroup statistics in an OnSkillGroupStatisticsUpdated event. If this request is successful, the OnNewSkillGroupStatistics event is no longer received.
The following is a C++ code example where the args parameter contains values for PeripheralID and SkillGroupNumber.

```cpp
Arguments & argsStatBroadcast = Arguments::CreateInstance();
argsStatBroadcast.AddItem(CTIOS_SKILLGROUPNUMBER, intSG);
argsStatBroadcast.AddItem(CTIOS_PERIPHERALID, m_periphID);
m_pSkGrStatSession->DisableSkillGroupStatistics { argsStatBroadcast };
argsStatBroadcast.Release();
```

### DumpProperties


### EnableSkillGroupStatistics

The EnableSkillGroupStatistics method requests that real-time statistics be sent to the skillgroup object. In an agent mode application, this request is usually made through the agent object (see Chapter 10, “Call Object”).

#### Syntax

**C++:**  
```cpp
int EnableSkillGroupStatistics(Arguments & args)
```

**COM:**  
```cpp
HRESULT EnableSkillgroupStatistics (IArguments * args, int * errorCode)
```

**VB:**  
```vbnet
EnableSkillgroupStatistics (args As CTIOSCLIENTLib.IArguments, errorCode As Long)
```

**Java:**  
```java
int EnableSkillGroupStatistics(Arguments args)
```

#### Parameters

- **args**
  
  If this method is called via the session object in monitor mode with the special filter, the args parameter has two required values for PeripheralID and SkillGroupNumber. See the Remarks section for a code example. Otherwise, this parameter is not used.

- **errorCode**
  
  An output parameter (return parameter in VB) that contains an error code, if any.
Return Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

Remarks

CTI OS Server sends skillgroup statistics in an OnSkillGroupStatisticsUpdated event.
The following is a C++ code example where the args parameter contains values for PeripherID and SkillGroupNumber.

```cpp
Arguments & argsStatBroadcast = Arguments::CreateInstance();
argsStatBroadcast.AddItem(CTIOS_SKILLGROUPNUMBER, intSG);
argsStatBroadcast.AddItem(CTIOS_PERIPHERALID, m_periphID);
m_pSkGrStatSession->EnableSkillGroupStatistics ( argsStatBroadcast );
argsStatBroadcast.Release();
```

GetElement


GetValue Methods


IsValid


SetValue

Helper Classes

The CTI OS Client Interface Library make extensive use of several custom data structures. This chapter describes the CTI OS Helper Classes (data structures). The following helper classes are distributed with the Client Interface Library:

- **Arg.** The Arg structure is the basic data type used in the CIL for any parameter included in methods or events. Objects of this type allow the CIL to be fully extensible and reusable. Arg supports many useful types including string, integer, Boolean, and Arguments array. Arg is the base class for the Arguments class. In most programming scenarios, programmers will not use Arg directly, but indirectly through the Arguments class.

- **Arguments.** The Arguments structure is used to maintain and send a set of key-value pairs between the CIL and CTI OS Server for events and requests. The Arguments array elements must all be of type Arg. The Arguments structure enables future growth of the CTI OS feature set, without requiring changes to the method call signature.

- **CilRefArg.** The CilRefArg class is a specialized subclass of Arg. It is used to store a reference to an object derived from CCtiOsObject (C++ only). For instance, it can hold reference to a CAgent, CCall, CSkillGroup, CCtiOsSession, or CWaitObject.

- **CCtiosException.** The CCtiosException class is used by CTIOS to provide detailed information when an exception occurs (C++ and Java only). When an exception is caught as CCtiosException, the programmer can query it for details such as error codes and error messages.
• **CWaitObject.** CWaitObject is a CIL object that derives from CtiOsObject. It is a utility class (C++ only) that enables a thread to wait for one or more CTI events. The user can provide a list of events along with a wait timeout. Wait objects are created with the CreateWaitObject Session Object method and destroyed with the DestroyWaitObject Session Object method.

### Arg Class

The `Arg` is a generic class used in parameters or return values in CIL methods. Information sent by CTI OS server to the CIL in an event is packed in an Arguments object where each element of the array is an object of type `Arg`. An `Arg` object’s absolute data type is determined by the type of data it stores. The basic types an object can store are identified by the enumerated constants in Table 12-2.

Arg class methods will do conversion between types whenever possible. For example, you can do a `SetValue(25)` and then do a `GetValueString()` which will return the string “25”. You can also do a `SetValue(“25”)` and then do a `GetValueIntObj` which will return an Integer object containing the numeric value 25. However, if you call `SetValue “abc”` and try to retrieve it as an int, it will fail.

Table 12-1 lists the available Arg class methods.

#### Table 12-1  Arg Class Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the data item.</td>
</tr>
<tr>
<td>Clone</td>
<td>Creates an exact copy of the Arg object.</td>
</tr>
<tr>
<td>CreateInstance</td>
<td>Creates an Arg object.</td>
</tr>
<tr>
<td>DumpArg</td>
<td>Builds a string containing the value stored in the Arg.</td>
</tr>
<tr>
<td>GetType</td>
<td>Returns the type of the data stored in the argument (one of the values in Table 12-2).</td>
</tr>
</tbody>
</table>
Arg Class

In many scenarios, programmers will stick to Arguments (see the preceding section), which wraps many Arg methods and encapsulates a collection of Arg objects.

AddRef

The AddRef method increments the reference count for the data item. It is necessary to call this if you are storing a pointer to the item for some time (for example, if you plan to store and use Arguments received in an event handler after the event thread has returned to the calling code). When you are finished with the item, you must call the Release method or a memory leak will result.

Syntax

C++:       unsigned long AddRef()
COM:        HRESULT AddRef()
VB, Java:   Not used

Parameters

None.

Return Values

COM: Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”
C++: The current reference count after the AddRef() call.

**Clone**

The Clone method allocates a new Arg in memory and copies its key, value, and type to the new instance.

**Syntax**

C++: `Arg & Clone()`  
COM: `HRESULT Clone(/*[out, retval]*)*/ IArg** arg);`  
VB: `Clone() as CTIOSCLIENTLib.IArg`  
Java: `Arg Clone()`

**Output Parameters**

arg  
Pointer to an IArg instance that is a copy of the object.

**Return Values**

COM: Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

Others: If successful, will return a reference to a new Arg object. If unsuccessful in C++ or VB, it will throw a CCtiosException with iCode set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED. If unsuccessful in Java, it returns null but does not throw an exception.

**CreateInstance**

The CreateInstance method creates an object of type Arg class and sets the reference count of the object to 1. It is important to release the object when it is no longer in use in the program.
Chapter 12      Helper Classes

Arg Class

Syntax

C++:
static Arg& CreateInstance(); // static creation mechanism.
    static Arg& CreateInstance(Arg& arg); // static creation
    mechanism.
    static bool CreateInstance(Arg ** arg); // static creation
    mechanism, alternate version
COM: Wrapped by CoCreateInstance
VB: Wrapped by New
Java: Not available

Parameters

arg
    (output) Pointer to the newly created Arg.

Return Values

COM: Default HRESULT return values. See Chapter 3, “CIL Coding
Conventions.”

Others: Either a reference to the newly created Arg or a boolean indicating
method success. If the methods not returning bool are unsuccessful, they will raise
a CCtiosException with iCode set to
E_CTIOS_ARGUMENT_ALLOCATION_FAILED.

Remarks

Internally this method increments the Arg’s reference count, so do not call
AddRef(). However, you must call Release() after you are finished with the Arg.

Dump Arg

The DumpArg method builds a string containing the value stored in the Arg. This
involves doing any type conversion required to display the data as a string. For
example, it will automatically convert an INTEGER type to a string that can be
logged for debugging. In the event that a Arg object is actually an Arguments
object, the string returned is the one built by Arguments.DumpArg, and thus
enabled printing of nested Arguments structures.
Chapter 12  Helper Classes

Arg Class

Syntax

C++: `string DumpArg()`
COM: `HRESULT DumpArg([out, retval] BSTR* arg_string)`
VB: `DumpArg() as String`
Java: Not available

Parameters

`arg_string`

The pointer to the string into which the contents of the Arg object will be written.

Return Values

COM: Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”
Others: A string containing the contents of the structure.

GetType

The GetType method returns the type of the data stored by the Arg. See Table 12-3 for a list of possible types.

Syntax

C++: `enumArgTypes GetType()`
COM: `HRESULT GetType(/*[out, retval]*/ int* type)`
VB: `GetType () as Integer`
Java: `int GetType()`

Output Parameters

`type`

Integer that receive the enumerated constant that identifies data type stored in IArg.
Return Values

**COM:** Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

**Others:** Returns the enumerated value that identifies the data type stored in the Arg (see Table 12-2).

<table>
<thead>
<tr>
<th>Argument Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARG_NOTSET</td>
<td>Argument type not determined</td>
</tr>
<tr>
<td>ARG_INT</td>
<td>Signed integer</td>
</tr>
<tr>
<td>ARG.Uint</td>
<td>Unsigned integer</td>
</tr>
<tr>
<td>ARG_USHORT</td>
<td>2 bytes unsigned integer</td>
</tr>
<tr>
<td>ARG_SHORT</td>
<td>2 bytes signed integer</td>
</tr>
<tr>
<td>ARG_BOOL</td>
<td>1 byte integer</td>
</tr>
<tr>
<td>ARG_STRING</td>
<td>STL character string</td>
</tr>
<tr>
<td>ARG_ARGARRAY</td>
<td>Variable length Arguments array</td>
</tr>
</tbody>
</table>

GetValue Methods

The GetValue method returns the value stored in the object. To extract a specific type of data you invoke the method designated for it. For more detail on GetValueArray, GetValueInt, and GetValueString, see the corresponding methods described in Chapter 7, “CtiOs Object.”

**Syntax**

```c++
int GetValueInt();
unsigned int GetValueUInt();
unsigned short GetValueUShort();
short GetValueShort();
string& GetValueString();
bool GetValueBool();
bool GetValueInt(int * value);
bool GetValueUInt(unsigned int * value);
```
bool GetValueUShort(unsigned short * value);
bool GetValueShort( short * psValuale);
bool GetValueBool( bool * value);
bool GetValueString(string* value);

COM: HRESULT GetValue(/*[out, retval]*/ VARIANT* value);
VB: GetValue() as Variant
VB: GetValue (key as String, value as Variant) as Boolean
Java: Integer GetValueIntObj()
    Long GetValueUINTObj()
    Short GetValueShortObj()
    Integer GetValueUShortObj()
    Boolean GetValueBoolObj()
    String GetValueString()

Parameters

Value
DropOutput parameter of the specified type containing the value of the Arg. For
COM, this value is of type VARIANT * whose type is one of the types listed
in Table 12-3.

Table 12-3 Variant types supported by GetValue (COM)

<table>
<thead>
<tr>
<th>Variant Type</th>
<th>Standard C++ Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT_INT</td>
<td>Int</td>
</tr>
<tr>
<td>VT_UINT</td>
<td>Unsigned int</td>
</tr>
<tr>
<td>VT_I2</td>
<td>Short</td>
</tr>
<tr>
<td>VT_UI2</td>
<td>Unsigned short</td>
</tr>
<tr>
<td>VT_BOOL</td>
<td>Bool</td>
</tr>
<tr>
<td>VT_BSTR</td>
<td>string, const string and char *</td>
</tr>
</tbody>
</table>

Return Values

C++
Methods taking no parameters, if successful, will return the value in the object;
otherwise, they will raise a CCtiosException with iCode set to
E_CTIOS_INVALID_ARGUMENT.
The methods taking a pointer to the variable receiving the result will return true, if the method was able to get the value, otherwise, false.

**COM**

If the method was able to set the variant type of the value (i.e., value->vt) to any of the types listed in Table 12-3, it returns the value in the appropriate field of the variant. Otherwise it returns VT_EMPTY.

### Release

The Release method decrements the reference count for the data item. It is necessary to call Release when you are finished with a data item that has had its reference count incremented via CreateInstance or AddRef; otherwise, a memory leak will result.

### Syntax

**C++:** unsigned long Release()

**COM:** HRESULT Release()

**VB, Java:** Not used

### Parameters

None.

### Return Values

**COM:** Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

**C++:** The current reference count after the Release() call.

### SetValue

The SetValue method sets the value in the Arg object.
Syntax

C++:

```cpp
bool SetValue( int value );
bool SetValue( unsigned int value );
bool SetValue( unsigned short value );
bool SetValue( short value );
bool SetValue( bool value );
bool SetValue( char * value );
bool SetValue( string& value );
bool SetValue( const string& value );
bool SetValue( Arg & value );
```

COM:

```cpp
HRESULT SetValue(/*[in]*/ VARIANT * pVariant, /*[out,retval]*/ VARIANT_BOOL * errorcode );
```

VB:

```vbnet
SetValue (key as String, value as Variant) as Boolean
```

Java:

```java
boolean SetValue(Arg rArg)
boolean SetValue(int iVal)
boolean SetValue(short nValue)
boolean SetValue(String sValue)
boolean SetValue(UInt(long lValue)
boolean SetValueUShort(int iValue
```

Parameters

`value`

The value of the specified type to assign to the Arg. For COM, this value is of type VARIANT * whose type is one of the types listed in Table 12-4.

**Table 12-4  Supported Variant Types**

<table>
<thead>
<tr>
<th>Variant Type</th>
<th>Standard C++ Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT_INT</td>
<td>Int</td>
</tr>
<tr>
<td>VT_UINT</td>
<td>Unsigned int</td>
</tr>
<tr>
<td>VT_I2</td>
<td>Short</td>
</tr>
<tr>
<td>VT_UI2</td>
<td>Unsigned short</td>
</tr>
<tr>
<td>VT_BOOL</td>
<td>Bool</td>
</tr>
<tr>
<td>VT_BSTR</td>
<td>string, const string and char *</td>
</tr>
<tr>
<td>VT_DISPATCH</td>
<td>Pointer to an IArg interface</td>
</tr>
</tbody>
</table>
errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

C++

If the method was able to set the value it returns true, otherwise it returns false.

COM, VB

If the method was able to set the value it returns VARIANT_TRUE. Otherwise, it returns VARIANT_FALSE.

Arguments Class

The Arguments structure (class) provides key/value support to form a collection of values. Each value stored in an Arguments structure is associated with a key. To add an item, use the AddItem method and pass a key and a value. The key must be a string or an enumerated value, and the value can be almost any type (i.e. all types supported by Arg). To retrieve the item, use the appropriate GetValue method with a key, and the value is returned. Keys are not case sensitive, and leading and trailing spaces are always removed from the key.

Arguments also supports access by index. The index is useful for retrieving items sequentially, but may not be as fast as retrieval by key. The Arguments structure’s index is 1-based, to provide easier support for Visual Basic programmers. Internally, the Arguments structure uses a binary tree and other techniques to provide fast access to any item. Arguments can support a virtually unlimited number of key-value pairs, and supports nested Arguments structure as well.

Table 12-5 lists the Arguments class methods.
### Arguments Class

#### Table 12-5 Arguments Class Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddItem</td>
<td>Adds an item to an Arguments array.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the data item.</td>
</tr>
<tr>
<td>Clear</td>
<td>Deletes all elements from an Arguments array.</td>
</tr>
<tr>
<td>Clone</td>
<td>Creates a copy of an Arguments array.</td>
</tr>
<tr>
<td>CreateInstance</td>
<td>Creates an Arguments array.</td>
</tr>
<tr>
<td>DumpArgs</td>
<td>Returns Arguments object as a string</td>
</tr>
<tr>
<td>GetElement</td>
<td>Returns the value stored under a specified index.</td>
</tr>
<tr>
<td>GetElementInt</td>
<td>Returns the value stored under a specified index.</td>
</tr>
<tr>
<td>GetElementUInt</td>
<td>Returns the value stored under a specified index.</td>
</tr>
<tr>
<td>GetElementUShort</td>
<td>Returns the value stored under a specified index.</td>
</tr>
<tr>
<td>GetElementShort</td>
<td>Returns the value stored under a specified index.</td>
</tr>
<tr>
<td>GetElementBool</td>
<td>Returns the value stored under a specified index.</td>
</tr>
<tr>
<td>GetElementString</td>
<td>Returns the value stored under a specified index.</td>
</tr>
<tr>
<td>GetElementArg</td>
<td>Returns the value stored under a specified index.</td>
</tr>
<tr>
<td>GetElementArgType</td>
<td>Returns the value stored under a specified index.</td>
</tr>
<tr>
<td>IsValid</td>
<td>Tests if a key is present in the current Arguments array.</td>
</tr>
<tr>
<td>NumElements</td>
<td>Returns the number of arguments in the current Arguments array.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count for the data item.</td>
</tr>
<tr>
<td>RemoveItem</td>
<td>Removes an item from an Arguments array.</td>
</tr>
<tr>
<td>SetElement</td>
<td>Sets the value of an index.</td>
</tr>
<tr>
<td>SetValue</td>
<td>Sets the value of a key.</td>
</tr>
</tbody>
</table>
Usage Notes

When writing an application using the CTI OS SDK, the following sequence of steps in the program may produce a problem:

- Programmer passes an Arguments array into a CTI OS SDK method (methodA)
- MethodA returns
- Programmer modifies the same Arguments array
- Programmer passes the modified Arguments array into another CTI OS SDK method (methodB)

When running the application, the call to methodA may behave as if it was passed the modified Arguments array. This is because many CTI OS methods simply place a pointer to the Arguments array on a queue of items to be sent to CTI OS server. When the same Arguments array is later modified, as in the preceding example, the pointer on the queue now points to the modified array and the modified array is sent to CTI OS server. A problem may occur depending on timing, as there are multiple threads involved: the thread pulling items off the queue and the thread modifying the Arguments array. If the queued message is sent to CTI OS before the Arguments array is modified, the problem will not occur.

To avoid this problem, call the Clone method on the initial Arguments array and modify the copy rather than modifying the original. For example, the preceding example would change as follows:

- Programmer passes an Arguments array (initialArray) into a CTI OS SDK method (methodA)
- MethodA returns
- modifiedArray = initialArray.Clone()
- Programmer modifies modifiedArray
- Programmer passes the modifiedArray into another CTI OS SDK method (methodB)
AddItem

The AddItem method expects a key/value pair. The key value may be a string or an integer. The value may be a string, an integer, or an object reference. If there is an entry with the same key, it will be replaced with this entry, otherwise the new key/value pair will be added to the arguments array. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Syntax

C++:

```cpp
bool AddItem( std::string& key, int value );
bool AddItem( std::string& key, unsigned int value );
bool AddItem( std::string& key, unsigned short value );
bool AddItem( std::string& key, short value );
bool AddItem( std::string& key, bool value );
bool AddItem( std::string& key, char * pchar );
bool AddItem( std::string& key, std::string& value );
bool AddItem( std::string& key, Arg& value );
bool AddItem( std::string& key, const Arg& value );
bool AddItem( std::string& key, Arguments& value );
bool AddItem( std::string& key, const Arguments& value );
bool AddItem( char * key, int value );
bool AddItem( char * key, unsigned int value );
bool AddItem( char * key, unsigned short value );
bool AddItem( char * key, short value );
bool AddItem( char * key, bool value );
bool AddItem( char * key, char * value );
bool AddItem( char * key, std::string& value );
bool AddItem( char * key, Arg& cArg );
bool AddItem( char * key, const Arg& value );
bool AddItem( char * key, Arguments& value );
bool AddItem( char * key, const Arguments& value );
bool AddItem( enum_Keywords key, int value );
bool AddItem( enum_Keywords key, unsigned int value );
bool AddItem( enum_Keywords key, unsigned short value );
bool AddItem( enum_Keywords key, short value );
bool AddItem( enum_Keywords key, bool value );
bool AddItem( enum_Keywords key, char * value );
bool AddItem( enum_Keywords key, std::string& value );
bool AddItem( enum_Keywords key, Arg& cArg );
```
bool AddItem( enum_Keywords key, const Arg& value );
bool AddItem( enum_Keywords key, Arguments& value );
bool AddItem( enum_Keywords key, const Arguments& value );

COM: HRESULT AddItem(/*[in]*/ VARIANT *key, /*[in]*/ VARIANT *value,
/*[out,retval]*/ VARIANT_BOOL *success) As Boolean;

VB: AddItem( Key as Variant, Value as Variant)

Java: boolean AddItem( String key, String value )
     boolean AddItem( String key, int value )
     boolean AddItem( String key, Arguments value )
     boolean AddItem( int key, String value )
     boolean AddItem( int key, int value )
     boolean AddItem( int key, Arguments value )

Parameters

key
Key name for the item to be added.

value
Value of the item to be added.

success
An output parameter (return parameter in C++ and VB) that contains a boolean indicating success or lack thereof.

Return Value

C++: Returns True in if the entry was successfully added, otherwise False.

COM and VB: Standard return values are valid; see Chapter 3, “CIL Coding Conventions.”

AddRef

The AddRef method increments the reference count for the data item. It is necessary to call this if you are storing a pointer to the item for some time. When you are finished with the item, you must call the Release method or a memory leak will result.
Arguments Class

Syntax

**C++:**

`unsigned long AddRef()`

**COM:**

`HRESULT AddRef()`

**VB, Java:**

Not used

**Parameters**

None.

**Return Values**

**COM:** Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

**Others:** None.

**Clear**

The Clear method deletes all the elements from Arguments object.

**Syntax**

**C++:**

`void Clear()`

**COM:**

`HRESULT Clear()`

**VB:**

`Clear()`

**Java:**

`void Clear()`

**Parameters**

None.

**Return Value**

None.
Clone

The Clone method creates a copy of the Arguments structure. Because in C++ this method is implemented in the base class (Arg), it returns a reference to an Arg, but this is actually a reference to an Arguments array. Therefore, it is necessary to cast the return value of this method. The following C++ code sample shows this casting:

```cpp
Arguments & argsCopy = (Arguments&) argsOrig.Clone();
```

To cast in VB, do the following:

```vbnet
Dim Args As CTIOSCLIENTLib.IArguments
Set Args = Orig.Clone()
```

Syntax

- **C++**: `Arg & Clone()`
- **COM**: `HRESULT Clone(/*[out, retval]*/ IArguments ** args);`
- **VB**: `Clone() as CTIOSCLIENTLib.IArguments`
- **Java**: `Arguments Clone()`

Parameters

- `args`
  
  An output parameter containing a pointer to an Arguments array that is a copy of the object.

Return Value

- **COM**: Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”
- **Others**: A reference to the Arg structure that is a copy of the object.

CreateInstance

The CreateInstance method creates an object of type Arguments class and sets the reference count of the object to 1. It is important to release the object when it is no longer in use in the program.
Arguments Class

Syntax

C++:

```cpp
static Arguments & CreateInstance()
static bool CreateInstance(Arguments ** args)
```

COM:
Not exposed, called by CoCreateInstance.

VB:
Not exposed, called by New.

Java:
Not exposed called by Arg.

Parameters

args

A pointer to the newly created Arguments structure.

Return Value

COM: Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

Others: Either a reference to the newly created Arguments structure or a boolean indicating method success.

Remarks

Internally this method increments the Arg’s reference count, so do not call AddRef(). However, you must call Release() after you are finished with the Arg.

DumpArgs

The DumpArgs method builds a string showing all of the members of the Arguments structure in the form “key1 = value1; key2 = value2;...”. It is primarily used for debugging.

Syntax

```cpp
C++:

string DumpArgs()
```

COM:

```cpp
HRESULT DumpArgs([out,retval] BSTR* arg_string);
```

VB:

```cpp
DumpArgs() as String
```

Java:

```cpp
string DumpArgs()
```
Parameters

arg_string

The pointer to the string containing the contents of the Arguments array listing all of the key/value pairs in the format of “key1 = value1; key2 = value2;...”.

Return Values

COM: Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

Others: A string containing the contents of the Arguments array listing all key/value pairs

GetElement Methods

The GetElement method is similar to GetValue, except that it uses an index value instead of a key. The index value is not related to the order in which items are added or removed. The order of items in Arguments is never guaranteed. This method is useful for sequentially iterating over all items in Arguments. The Index is 1-based. The Index should never be less than one or greater than NumElements. see also NumElements method. The GetElementKey returns the key of a given index.

Syntax

C++:

Arg& GetElement( int index );
bool GetElement( int index, Arg ** value);
int GetElementInt( int index );
bool GetElementInt( int index, int * value);
unsigned int GetElementUInt( int index );
bool GetElementUInt( int index, unsigned int * value);
unsigned short GetElementUShort( int index );
bool GetElementUShort( int index, unsigned short * value );
short GetElementShort( int index );
bool GetElementShort( int index, short * value );
bool GetElementBool( int index );
bool GetElementBool( int index, bool * value);
std::string GetElementString( int index );
bool GetElementString( int index, std::string * value);
Arguments& GetElementArg( int index );
bool GetElementArg( int index, Arguments ** key);
std::string GetElementKey( int index );
bool GetElementKey( int nIndex, std::string * key);
bool GetElementKey( int nIndex, int * key);

COM: HRESULT GetElementKey(/*[in]*/ int index, /*[out]*/ BSTR * key);
HRESULT GetElement(/*[in]*/ int index, /*[out]*/ VARIANT * value);

VB: GetElement (Integer index, Variant value)
    GetElement (Integer index, String key)

Java: Arg
    GetElement (int iIndex)
Arguments GetElementArguments (int iIndex)
    Integer GetElementIntObj (int iIndex)
    Long GetElementUIntObj (int iIndex)
    Short GetElementShortObj (int iIndex)
    Integer GetElementUShortObj (int iIndex)
    Boolean GetElementBoolObj (int iIndex)
    String GetElementString (int iIndex)
    String GetElementKey (int iIndex)

Parameters

value
    An output parameter containing the value of the member at the specified index.

key
    An output parameter containing the key of the member at the specified index.

index
    An input parameter containing an index into the Arguments array.

Return Value

COM: Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

Others: Returns either the value at the index specified independently from its key, or a boolean indicating success or failure.
GetValue Methods

The GetValue method returns the value stored under a key. This method will return a blank string if the key is invalid. The existence of a key can be tested using IsValid. Keys are not case sensitive. Leading and trailing spaces are always removed from the key. For more detail on GetValueArray, GetValueInt, and GetValueString, see the corresponding methods described in Chapter 7, “CtiOs Object.”

Syntax

C++:

```
Arg& GetValue( enum_Keywords eKey );
bool GetValue( enum_Keywords key, Arg ** value );
Arg& GetValue( std::string& key );
bool GetValue( std::string& key, Arg ** value );
Arg& GetValueArg( std::string& key );
bool GetValueArg( std::string& key, Arg ** value );
int GetValueInt( enum_Keywords key); /*throws exception*/
bool GetValueInt( enum_Keywords key, int * value );
unsigned int GetValueUInt( enum_Keywords key );
bool GetValue UInt( enum_Keywords key, unsigned int * value );
unsigned short GetValueUShort( enum_Keywords key );
bool GetValueUShort( enum_Keywords key, unsigned short * value );
short GetValueShort( enum_Keywords key );
bool GetValueShort( enum_Keywords key, short * value );
bool GetValueBool( enum_Keywords key );
bool GetValueBool( enum_Keywords key, bool * value );
std::string GetValueString( enum_Keywords key );
bool GetValueString( enum_Keywords key, std::string * value );
int GetValueInt( std::string& key ); /*throws exception*/
bool GetValueInt( std::string& key, int * value );
unsigned int GetValueUInt( std::string& key );
bool GetValue UInt( std::string& key, unsigned int * value );
unsigned short GetValueUShort( std::string& key );
bool GetValueUShort( std::string& key, unsigned short * value );
short GetValueShort( std::string& key );
bool GetValueShort( std::string& key, short * value );
bool GetValueBool( std::string& key );
bool GetValueBool( std::string& key, bool * value );
std::string GetValueString( std::string& key );
Arguments& GetValueArray( std::string& key );
bool GetValueArray( std::string& key, Arguments ** value );
Arguments& GetValueArray( enum_Keywords key );
bool GetValueArray( enum_Keywords key, Arguments ** value );
```
Chapter 12  Helper Classes

Arguments Class

Arg& GetValue( char * key );
bool GetValue( char * key, Arg ** value);
Arguments& GetValueArray( char * key );
bool GetValueArray( char * key, Arguments ** value);
int GetValueInt( char * key );
bool GetValueInt( char * key, int * value);
unsigned int GetValueUInt( char * key );
bool GetValueUInt( char * key, unsigned int * value);
unsigned short GetValueUShort( char * key );
bool GetValueUShort( char * key, unsigned short * value);
short GetValueShort( char * key );
bool GetValueShort( char * key, short * value);
bool GetValueBool( char * key );
bool GetValueBool( char * key, bool * value);
std::string GetValueString( char * key );
bool GetValueString( char * key, std::string * value);
Arg& GetValueArg( char * key );
bool GetValueArg( char * key, Arg ** value);

COM:
HRESULT GetValue(/*[in]*/ BSTR key, /*[out, retval]*/ VARIANT * pValue);
HRESULT GetValueInt(/*[in]*/ VARIANT *key, /*[out, retval]*/ int *value);
HRESULT GetValueString(/*[in]*/ VARIANT *key, /*[out, retval]*/ BSTR *value);
HRESULT GetValueArray(/*[in]*/ VARIANT *key, /*[out, retval]*/ IArguments **pArguments);
HRESULT GetValueBool(/*[in]*/ VARIANT *key, /*[out, retval]*/ VARIANT_BOOL * value);

VB:
GetValue (Key as String) as Variant
GetValue(key As Variant) As Arg
GetValueArray(key As Variant) As Arguments
GetValueBool(key As Variant) As Boolean
GetValueInt(key As Variant) As Long
GetValueString(key As Variant) As String

Java:
Arg GetValue(int iKey)
Arg GetValue(String sKey)
Arguments GetValueArray(int iKey)
Arguments GetValueArray(String sKey)
Integer GetValueIntObj(int iKey)
Integer GetValueIntObj(String sKey)
Long GetValueUIntObj(int iKey)
Long GetValueUIntObj(String sKey)
Short GetValueShortObj(int iKey)
Short GetValueShortObj(String sKey)
Integer GetValueUShortObj(int iKey)
Integer GetValueUShortObj(String sKey)
Boolean GetValueBoolObj(int iKey)
Boolean GetValueBoolObj(String sKey)
Arguments Class

String GetValueString(int iKey)
String GetValueString(String sKey)

Parameters

An enumerated keyword (see Appendix A, “CTI OS Keywords and Enumerated Types”) or a string specifying the keyword of the value to be retrieved.

Return Values

C++ returns a Bool or the a reference to the value, COM returns an HRESULT. See also Chapter 3, “CIL Coding Conventions.”

Remarks

Visual Basic’s Integer type is a 16-bit integer. However, the GetValueInt method returns a 32-bit integer. Thus, in Visual Basic the return type for GetValueInt is actually a Visual Basic type Long. Visual Basic Programmers can use the GetValueInt method and receive the return value as an Integer, and Visual Basic will perform an implicit cast. However, if the value retrieved is a 32-bit integer, an overflow error will occur. To resolve this error, it is recommended that you use a 32-bit integer (Long).

Those methods that do not return a bool indicating success or failure will throw a CtiosException if the method fails. The most common reasons for failure are NULL key or element with specified key not found.

IsValid

The IsValid method returns True if the specified key exists in the current Arguments array, otherwise it returns False.

Syntax

C++:   bool IsValid( std::string& key );
       bool IsValid( char * key );
       bool IsValid( Arg& arg );
       bool IsValid( enum_Keywords key );
### Arguments Class

**COM:**
```
HRESULT IsValid( /*[in]*/ VARIANT* key, /*[out, retval]*/
    VARIANT_BOOL* bIsvalid);
```

**VB:**
```
IsValid (key as string) as Boolean
```

**Java:**
```
boolean IsValid(int key)
boolean IsValid(String key)
boolean   IsValid(Arg rArg)
```

### Parameters

**key/arg**

Either the key of the desired Arguments member or an Arg containing a string key.

C++ and COM allow you to specify the key as string or enumerated (see Appendix A, “CTI OS Keywords and Enumerated Types”); all others expect the key as a string.

### Return Values

**COM:** Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

**Others:** True if key exists in the current Arguments array, otherwise False.

### NumElements

The NumElements method returns number of elements stored in the current arguments array. This method is useful in combination with GetElement to implement a “for” loop to iterate over all values of an arguments array without knowing the keywords (those can be retrieved at the same time using GetElementKey).

### Syntax

**C++:**
```
int  NumElements();
```

**COM:**
```
HRESULT NumElements( /*[out, retval]*/ int * num_elements);
```

**VB:**
```
NumElements as Integer
```

**Java:**
```
int NumElements()
```
Chapter 12  Helper Classes

Arguments Class

Parameters

num_elements

Pointer to an integer value containing the number of elements in the Arguments array.

Return Value

**COM:** Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

**Others:** Number of elements in arguments array.

Release

The Release method decrements the reference count for the data item. It is necessary to call Release when you are finished with a data item that has had its reference count incremented via CreateInstance or AddRef; otherwise, a memory leak will result.

Syntax

**C++:** unsigned long Release()

**COM:** HRESULT Release()

**VB, Java:** Not used

Parameters

None.

Return Values

**COM:** Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

**C++:** The current reference count after the Release() call.
Chapter 12 Helper Classes

Arguments Class

RemovalItem

The RemovalItem method removes a value and its associated key from an arguments array. Subsequent attempts to access a value that was removed using RemovalItem will fail.

Syntax

C++:
bool RemoveItem( std::string& key );
bool RemoveItem( char * key );
bool RemoveItem( enum_Keywords key );

COM:
HRESULT RemoveItem(/*[in]*/ VARIANT* key, /*[out, retval]*/ VARIANT_BOOL* bSuccess);

VB:
RemoveItem ( key as Variant) as Boolean

Java:
boolean RemoveItem(int key)
    boolean RemoveItem(String key)

Parameters

key

The key to use to locate and remove the item in the Arguments array. Leading and trailing spaces are always removed from the key.

Return Values

A True return value means the entry was located and removed.

SetElement

The SetElement method is identical to SetValue (which is similar to AddItem), except that it uses an index value instead of a key.

Syntax

C++:
bool SetElement( int index, int value );
    bool SetElement( int index, unsigned int value );
    bool SetElement( int index, unsigned short value );
    bool SetElement( int index, short value );
bool SetElement( int index, bool value );

bool SetElement( int index, std::string& value );
bool SetElement( int index, char * pchar );
bool SetElement( int index, Arg& value );
bool SetElement( int index, Arguments& value );

COM: HRESULT SetElement(/*[in]*/ int index, /*[in]*/ VARIANT * value, /*[out,retval]*/ boolean * success);

VB: SetElement (index as Integer, value as Variant) as Boolean

Java: Not available

Parameters

index

The index at which the value is to be set. This index value is not related to the order in which items are added or removed. The order of items in Arguments is never guaranteed. This method is useful for sequentially iterating over all items in Arguments. Index is 1-based. Index should never be less than 1 or greater than NumElements (see above). C++ implements several overloaded methods for different value types, while COM and VB use Variants.

value

The associated value to be set in the element at the designated index.

success

Output parameter (return parameter in C++ and VB) containing a boolean indicating success or failure.

Return Values

COM: Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

Others: A boolean indicating success or failure.
SetValue

The SetValue method sets a value for a key. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Syntax

C++:

```cpp
bool SetValue( std::string& key, int value );
bool SetValue( std::string& key, unsigned int value );
bool SetValue( std::string& key, unsigned short value );
bool SetValue( std::string& key, short value );
bool SetValue( std::string& key, bool value );
bool SetValue( std::string& key, std::string& value );
bool SetValue( std::string& key, char * pchar );
bool SetValue( std::string& key, Arg& value );
bool SetValue( std::string& key, Arguments& value );
bool SetValue( std::string& key, const Arguments& value );
bool SetValue( char * key, int value );
bool SetValue( char * key, unsigned int value );
bool SetValue( char * key, unsigned short value );
bool SetValue( char * key, short value );
bool SetValue( char * key, bool value );
bool SetValue( char * key, std::string& value );
bool SetValue( char * key, char * value );
bool SetValue( char * key, Arg& value );
bool SetValue( char * key, Arguments& value );
bool SetValue( char * key, const Arguments& value );
bool SetValue( enum_Keywords key, int value );
bool SetValue( enum_Keywords key, unsigned int value );
bool SetValue( enum_Keywords key, unsigned short value );
bool SetValue( enum_Keywords key, short value );
bool SetValue( enum_Keywords key, bool value );
bool SetValue( enum_Keywords key, std::string& value );
bool SetValue( enum_Keywords key, Arg& value );
bool SetValue( enum_Keywords key, const Arg& value );
bool SetValue( enum_Keywords key, const Arguments& cArguments );
bool SetValue( enum_Keywords key, char * value );
```

COM:

```c
HRESULT SetValue(/*[in]*/ VARIANT* key, /*[in]*/ VARIANT* value, /*[out, retval]*/ VARIANT_BOOL* success);
```

VB:

```vb
SetValue (key as String, value as Variant) as Boolean
```
Java: boolean SetValue(Arguments rArguments)
      boolean SetValue(int iKey, Arg rArg)
      boolean SetValue(String sKey, Arg rArg)
      boolean SetValue(int iKey, int iVal)
      boolean SetValue(String sKey, int iVal)
      boolean SetValue(int iKey, short nValue)
      boolean SetValue(String sKey, short nValue)
      boolean SetValue(int iKey, String sValue)
      boolean SetValue(String sKey, String sValue)
      boolean SetValueUInt(int iKey, long lValue)
      boolean SetValueUInt(String sKey, long lValue)
      boolean SetValueUShort(int iKey, int iValue)
      boolean SetValueUShort(String sKey, int iValue)

Parameters

key
The key whose value is to be set.

value
The value to use in setting the element with the designated key.

success
Output parameter (return parameter in C++ and VB) containing a boolean indicating success or failure.

Return Values

Bool return value that indicates success or failure of the operation.

Remarks

The C++ methods overload several implementations for different value types and allow to specify a key via enumerated keywords (see Appendix A, “CTI OS Keywords and Enumerated Types”) as well as string. COM and VB use String keywords and Variants as values.
CILRefArg Class (C++ and Java only)

The CILRefArg class is a subclass of the Arg class. Its main responsibility is to store a reference of a CCTiOsObject object (see Chapter 7, “CtiOs Object”). This class allows object references to be included in argument structure. The object types that can be used are any of the following: CAgent, CCall, CSkillGroup, CWaitObject or CCTiOsSession.

In addition to the methods inherited from the Arg class, the CILRefArg class contains the methods listed in Table 12-6.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetType</td>
<td>Returns the ARG_REFERENCE.</td>
</tr>
<tr>
<td>GetUniqueObjectID</td>
<td>Returns the UID of the contained CtiOsObject</td>
</tr>
<tr>
<td>GetValue</td>
<td>Returns the encapsulated pointer in the object.</td>
</tr>
<tr>
<td>SetValue</td>
<td>Encapsulates the pointer to CTI OS object into the CILRefArg object.</td>
</tr>
</tbody>
</table>

GetType

The CILRefArg class GetType method returns the ARG_REFERENCE. It is defined an enumerated type to represent CTI OS Objects references. It has the same syntax as the Arg class GetType method.

GetUniqueObjectID (Java only)

The GetUniqueObjectID method returns the unique objectID of the contained CtiOsObject.

Syntax

```java
String GetUniqueObjectID()
```
CILRefArg Class (C++ and Java only)

Chapter 12      Helper Classes

**CILRefArg Class (C++ and Java only)**

**Parameters**

None.

**Return Values**

If successful, it returns the unique objectID of the contained CtiOsObject. If no object is contained in the CILRefArg, it returns null.

**Remarks**

To obtain a unique object ID in C++, use `bool GetValueString(string* pString)`.

**GetValue**

The GetValue method returns the reference to CTI OS object encapsulated in the CILRefArg.

**Syntax**

```cpp
CCtiOsObject * GetValue();
```

**Parameters**

None.

**Return Values**

If successful, it returns the value in the object. Otherwise, it will throw a CCtiosException with `iCode` set to E_CTIOS_INVALID_ARGUMENT.

**SetValue**

Sets the reference to the CTI OS Object in the CILRefArg.
## Syntax

```cpp
bool SetValue(CCtiOsObject * pObject);
```

### Input Parameters

- **pObject**
  A pointer to a CtiOsObject to encapsulate (e.g. CCall, CAgent, etc.)

### Return Values

If the method was able to set the reference it returns true. Otherwise, it returns false.

## CCtiOsException Class (C++ and Java only)

The CCtiOsException class is normally used within the Arguments class. It provides access to additional information when exceptions are thrown, such as what parameter is in error, memory allocation failed, and so on.

Table 12-7 lists the available CCtiOsException class methods.

### Table 12-7  CCtiOsException Class Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCtiOsException</td>
<td>Class constructor.</td>
</tr>
<tr>
<td>getCode</td>
<td>Returns the error code that generated the exception.</td>
</tr>
<tr>
<td>getStatus</td>
<td>Returns the error status that generated the exception.</td>
</tr>
<tr>
<td>getString</td>
<td>Returns a text string containing the description of the exception.</td>
</tr>
<tr>
<td>What</td>
<td>Returns a text string containing the description of the exception, the code of an error and the status.</td>
</tr>
</tbody>
</table>
CCtiosException Constructor

The CCtiosException constructor initializes an object of type CCtiosException.

Syntax

CCtiosException(const char *pMsg, int iCode, int iStatus);
CCtiosException(const string& rstrMsg, int iCode, int iStatus);

Input Parameters

pMsg
Pointer to string that holds a description of an error.
iCode
Number that identifies an error.
iStatus
Status of an error.
rstrMsg
An STL string that holds a description of an error.

Return Values

None.

GetCode

The GetCode method returns the error code that generated the exception.

Syntax

C++ and Java: int GetCode();
Parameters

None.

Return Values

Returns an integer error code that generated the exception. The errors are described in the Cilerror.h include file – see also Appendix A, “CTI OS Keywords and Enumerated Types.”

getStatus

The getStatus method returns the error status that generated the exception.

Syntax

C++ and Java: int getStatus ();

Parameters

None.

Return Values

Returns an integer error status that generated the exception.

getString

The getString method returns a text string containing the description of the exception.

Syntax

C++: const char* getString ();
Java: String getString ();
CWaitObject Class (C++ and Java only)

CWaitObject is a CIL object that derives from CtiOsObject. It is a utility class that enables a thread to wait for one or more CTI events. The user can provide a list of events along with a wait timeout. Wait objects are created with the CreateWaitObject Session Object method and destroyed with the DestroyWaitObject Session Object method.
Warning

You must not use a WaitObject instance within an event handler. Events are sent to desktop applications by a single thread in the CIL. If that thread is blocked while waiting for a specific event, the thread will deadlock and the event handler will not receive any more events.

Methods

Table 12-8 list the CWaitObject class methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateWaitObject</td>
<td>See Chapter 8, “Session Object.”</td>
</tr>
<tr>
<td>DestroyWaitObject</td>
<td>See Chapter 8, “Session Object.”</td>
</tr>
<tr>
<td>DumpEventMask</td>
<td>Returns a printable string listing the events in the CWaitObject’s mask</td>
</tr>
<tr>
<td>GetMask</td>
<td>Sets a user provided pointer to an Arguments object that contains the list of events that the object will wait for.</td>
</tr>
<tr>
<td>GetTriggerEvent</td>
<td>Gets the ID of the event that triggered the WaitOnMultipleEvents method to wake.</td>
</tr>
<tr>
<td>InMask</td>
<td>Returns true if the specified event ID is in the list of events that the object will wait for.</td>
</tr>
<tr>
<td>SetMask</td>
<td>Set the list of events that the object will wait for.</td>
</tr>
<tr>
<td>WaitOnMultipleEvents</td>
<td>Waits for the events in the object’s event mask for the specified time period or until one of the events occurs.</td>
</tr>
</tbody>
</table>

CreateWaitObject

See Chapter 8, “Session Object.”
DestroyWaitObject

See Chapter 8, “Session Object.”

DumpEventMask

The DumpEventMask method returns a printable string listing the events in the CWaitObject’s mask.

Syntax

C++ and Java: string DumpEventMask();

Parameters

None.

Return Values

A printable string object listing the events in the wait mask.

GetMask

The GetMask method gets the list of events that the CWaitObject will wait for.

Syntax

C++: bool GetMask(Arguments ** pMask);
Java: Arguments GetMask();

Parameters

pMask
A pointer to an Arguments object pointer. GetMask will set the value of pMask to a pointer to an Arguments object that contains the event mask.
Return Values

If the method was able to get the mask it returns true; otherwise, it returns false.

GetTriggerEvent

The GetTriggerEvent method returns the ID of the last event in the CWaitObject’s mask that triggered the WaitOnMultipleEvents method to wake.

Syntax

C++: EnumCTIOS_EventID GetTriggerEvent();
Java: int GetTriggerEvent();

Parameters

None.

Return Values

The ID of the event or eUnknownEvent if no event triggered a wakeup.

InMask

The InMask method checks to see if the specified event is in the mask of events that the CWaitObject will wait for.

Syntax

C++: bool InMask(int iEventId);
Java: boolean InMask(int iEventId);
Parameters

iEventId

The enumerated event ID of the event to check for.

Return Values

If the event is in the mask it returns true. Otherwise, it returns false.

SetMask

The SetMask method sets the list of events that the CWaitObject will wait for.

Syntax

C++: bool SetMask(Arguments & args);
Java: boolean SetMask(Arguments rArgs);

Parameters

args

A reference to an Arguments object containing the list of events to wait for. The Arguments should contain values where the keys are “Event1” through “EventN” and the values are the enumerated event IDs.

Return Values

If the method was able to set the mask it returns true. Otherwise it returns false.

WaitOnMultipleEvents

The WaitOnMultipleEvents method waits for the events in the CWaitObject’s wait mask or returns if one of the events has not occurred after the specified timeout period. This is a “one of” method which returns after one of the specified events occurs.
CWaitObject Class (C++ and Java only)

Syntax

C++: int WaitOnMultipleEvents(DWORD dwMilliseconds = INFINITE);
Java: int WaitOnMultipleEvents(long iMilliseconds);

Parameters

Milliseconds

The maximum length of time in milliseconds to wait before timing out. The default is INFINITE if called without arguments.

Return Values

The WaitOnMultipleEvents method returns one of the values listed in Table 12-9.

<table>
<thead>
<tr>
<th>Value</th>
<th>When Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_SIGNALED</td>
<td>If one of the events in the mask occurred.</td>
</tr>
<tr>
<td>EVENT_WAIT_TIMEDOUT</td>
<td>If the timeout period elapsed.</td>
</tr>
<tr>
<td>WAIT_FAILED</td>
<td>If unable to wait on the events in the mask.</td>
</tr>
</tbody>
</table>
SilentMonitorManager Object

The SilentMonitorManager Object provides developers with an interface to silent monitor behavior. The SilentMonitorManager object exposes methods to perform all silent monitor tasks, such as starting, stopping and managing silent monitor sessions. The SilentMonitorManager object stores specific silent monitor session information as properties.

The SilentMonitorManager object can be used in two different modes:

- In Monitoring Mode, an application that wants to silent monitor conversation without being noticed by the calling parties must create a SilentMonitorManager object and set it mode to eSMMonitoringMode using the StartSMMonitoringMode method.

- In Monitored Mode, an application will accept requests to initiate silent monitor sessions to forward the voice conversations to the remote monitoring application. The application will create a SilentMonitorManager object and set the mode to eSMMonitoredMode using the StartSMMonitoredMode method.

For more information on these modes see the section “Initiating and Ending a Silent Monitor Session” in Chapter 4, “Building Your Application.”

---

**Note**

SilentMonitorManager Object methods and properties are not available in the Java CIL. SilentMonitorManager Object methods and properties are supported for use with Cisco IPCC only.
Properties

Table 13-1 lists the SilentMonitorManager object properties.

Table 13-1  SilentMonitorManager Object Properties

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Heartbeat interval for the silent monitor session.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Timeout for no activity.</td>
</tr>
<tr>
<td>IPPhoneInformation</td>
<td>ARGUMENTS</td>
<td>This property is only accessible via the GetIPPhoneInfo method. It contains all the information related to the IP Phone used by the application.</td>
</tr>
<tr>
<td>MediaTerminationPort</td>
<td>INT</td>
<td>TCP/IP port where monitored conversation will be sent for playback on system sound card.</td>
</tr>
<tr>
<td>SessionInformation</td>
<td>ARGUMENTS</td>
<td>This property is only accessible via the GetSessionInfo method. It contains all the information related to the current active silent monitor session.</td>
</tr>
<tr>
<td>SMManagerMode</td>
<td>SHORT</td>
<td>Mode in which the manager object will operate (Table 13-2).</td>
</tr>
</tbody>
</table>
Table 13-2  SMManagerMode Values

<table>
<thead>
<tr>
<th>Enum Value</th>
<th>Description</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eSMModeNotSet</td>
<td>Mode not set.</td>
<td>-1</td>
</tr>
<tr>
<td>eSMMonitoredMode</td>
<td>The manager accept request for silent monitor sessions and forward voice to the monitoring application.</td>
<td>0</td>
</tr>
<tr>
<td>eSMMonitoringMode</td>
<td>The manager can make requests to remote client to start a silent monitor session and send voice.</td>
<td>1</td>
</tr>
</tbody>
</table>

Methods

Table 13-3 lists the SilentMonitorManager object methods.

Table 13-3  SilentMonitorManager Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcceptSilentMonitoring</td>
<td>Establishes a silent monitor session and immediately starts sending audio.</td>
</tr>
<tr>
<td>GetIPPhoneInfo</td>
<td>Retrieves the information of the IP Phone used by the client application.</td>
</tr>
<tr>
<td>GetSessionInfo</td>
<td>Retrieves the information related to the current silent monitor session.</td>
</tr>
<tr>
<td>GetSMSessionList</td>
<td>Retrieves a list of all active Silent Monitor sessions.</td>
</tr>
<tr>
<td>IsMonitoredTarget</td>
<td>Determines if the device/agent is a target being monitored.</td>
</tr>
<tr>
<td>SetIPPhoneInfo</td>
<td>Saves the information of the IP Phone used by the client application.</td>
</tr>
<tr>
<td>StartSilentMonitorRequest</td>
<td>Sends a silent monitor session start request to a targeted client.</td>
</tr>
</tbody>
</table>
Methods

Chapter 13 SilentMonitorManager Object

Table 13-3 SilentMonitorManager Object Methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartSMMonitoredMode</td>
<td>Puts the SilentMonitorManager in Monitored mode.</td>
</tr>
<tr>
<td>StartSMMonitoringMode</td>
<td>Puts the SilentMonitorManager in Monitoring mode.</td>
</tr>
<tr>
<td>StopSilentMonitorMode</td>
<td>Sets the SilentMonitorManager mode to eSMModeNotSet. If a silent monitor session is active at this time, the session is terminated.</td>
</tr>
<tr>
<td>StopSilentMonitorRequest</td>
<td>Stops the active silent monitor session.</td>
</tr>
</tbody>
</table>

Argument Parameter Rules

The following rules apply to the optional_args and reserved_args parameters in SilentMonitorManager Object methods:

- In VB, you can ignore these parameters altogether. For example, you can treat the line:

  \[
  \text{StartSMMonitoringMode}([\text{reserved_args As IArguments}]) \text{ As Long}
  \]

  as follows:

  \[
  \text{StartSMMonitoringMode}()
  \]

- To ignore these parameters in COM you must send a NULL, as shown:

  \[
  \text{StartSMMonitoringMode}(\text{NULL})
  \]

AcceptSilentMonitoring

The AcceptSilentMonitoring method establishes the silent monitor session requested by the OnSilentMonitorRequestedEvent and immediately starts sending audio to the monitoring client. This method should only be used if the parameter DoDefaultMessageHandling was set to False when the subscriber handled the OnSilentMonitorRequestedEvent event.
Syntax

C++: int AcceptSilentMonitoring(Arguments & args);
COM: HRESULT AcceptSilentMonitoring ( /*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode);
VB: AcceptSilentMonitoring (ByVal args as CTIOSCLIENTLIB.IArguments) As Long

Parameters

args

Arguments array that contains the parameters listed in Table 13-4:

Table 13-4 AcceptSilentMonitoring Arguments Array Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the monitoring application.</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port of the monitoring application.</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Heartbeat interval for the silent monitor session.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Timeout for no activity.</td>
</tr>
<tr>
<td>OriginatingServerID</td>
<td>STRING</td>
<td>TCP/IP Address:Port of the CTIOS server from which the request originated.</td>
</tr>
</tbody>
</table>
errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

GetIPPhoneInfo

The GetIPPhoneInfo method gets the information about the client application IP Phone.

Syntax

C++: Arguments * GetIPPhoneInfo(void);
COM: HRESULT GetIPPhoneInfo ( /*[out,retval]*/ IArguments ** IPPhoneInfo);
VB: GetIPPhoneInfo () as CTIosclientlib.IArguments

Table 13-4 AcceptSilentMonitoring Arguments Array Parameters (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OriginatingClientID</td>
<td>STRING</td>
<td>Client Identification of the monitoring application.</td>
</tr>
<tr>
<td>DoDefaultMessageHandling</td>
<td>BOOLEAN</td>
<td>When this parameter is set to True, it instructs the SilentMonitorManager to immediately start sending audio and establish the silent monitor session. If this value is set to False, it instructs the SilentMonitorManager to not send voice and not to establish the silent monitor session. It is then the responsibility of the subscriber to report this status accordingly.</td>
</tr>
</tbody>
</table>
Parameters

None.

Return Value

This method returns an arguments array that contain the parameters listed in Table 13-5.

Table 13-5 GetIPPhoneInfo Return Arguments Array

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientAddress</td>
<td>STRING</td>
<td>IP Address of the IP Phone to be used by the client application.</td>
</tr>
<tr>
<td>BitRate</td>
<td>INT</td>
<td>Audio transmission bit rate</td>
</tr>
<tr>
<td>PacketSize</td>
<td>INT</td>
<td>Number of milliseconds of audio stored in a packet.</td>
</tr>
<tr>
<td>Direction</td>
<td>SHORT</td>
<td>One of the following values that indicates the direction of voice flow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>between the calling party and the called party:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: Bidirectional</td>
</tr>
<tr>
<td>RTPType</td>
<td>SHORT</td>
<td>One of the following values that indicates the type of RTP messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>between the calling party and the called party:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: audio</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: video</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: data</td>
</tr>
</tbody>
</table>
Methods

Chapter 13  SilentMonitorManager Object

Table 13-5  GetIPPhonelnfo Return Arguments Array (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EchoCancelation</td>
<td>SHORT</td>
<td>One of the following values that indicates whether the echo cancellation feature is enabled on this IP Phone: 0: Off 1: On</td>
</tr>
<tr>
<td>PayLoadType</td>
<td>SHORT</td>
<td>Audio codec type.</td>
</tr>
</tbody>
</table>

GetSessionInfo

The GetSessionInfo method retrieves the information related to the current silent monitor session.

Syntax

C++: Arguments * GetSessionInfo(Arguments & args) ;
COM: HRESULT GetSessionInfo ( /*[in]*/ IArguments * args,
 /*[out, retval]*/ IArguments * SMSessionInfo );
VB: GetSessionInfo (ByVal args as CTIOSCLIENTLIB.IArguments) As
CTIOSCLIENTLIB.IArguments

Parameters

args

Arguments array that contains one, but not both, of the parameters listed in Table 13-6:
Table 13-6  GetSessionInfo Arguments Array Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED</td>
<td>Unique silent monitor session Object ID of the target object that is being monitored.</td>
</tr>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the target object that is being monitored.</td>
</tr>
</tbody>
</table>

Return Values

This method returns an arguments array containing the key/value pairs listed in Table 13-7:

Table 13-7  GetSessionInfo Return Arguments Array Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED</td>
<td>Unique silent monitor session Object ID of the target object that is being monitored.</td>
</tr>
<tr>
<td>SMSessionStatus</td>
<td>SHORT</td>
<td>One of the ISilentMonitorEvent status codes in Table 6-4.</td>
</tr>
<tr>
<td>AudioMode</td>
<td>INT</td>
<td>Reserved. Specifies the audio mode bitmask.</td>
</tr>
<tr>
<td>AgentID/DeviceID</td>
<td>STRING</td>
<td>Agent ID or DeviceID of the target being monitored.</td>
</tr>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the target object being monitored.</td>
</tr>
<tr>
<td>MonitoredDeviceIPAddress</td>
<td>STRING</td>
<td>TCP/IP Address of the monitored IP Phone being monitored.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>ID of the peripheral associated with the agent and IP phone.</td>
</tr>
</tbody>
</table>
Methods

13-10
Cisco ICM Software CTI OS Developer's Guide Release 6.0(0)

Chapter 13  SilentMonitorManager Object

GetSMSessionList

The GetSMSessionList method retrieves a list of all active SilentMonitor Sessions in an arguments array. If the number of elements returned is greater than 0, this indicates that the SilentMonitorManager currently is in an active Silent Monitor session.

Syntax

C++: Arguments * CIL_API GetSMSessionList(void)
COM: HRESULT GetSMSessionList([out,retval] IArguments **pIArguments );
VB: GetSMSessionList () as CTIOSCLIENTLIB.IArguments

Parameters

None.

Return Values

Arguments array that contains a list of all Silent Monitor sessions. The current version only allows one active session, so the main use for this function is to use the NumElements method on the returned arguments array to determine if the current SilentMonitorManager is in an active Silent Monitor session.

Table 13-7  GetSessionInfo Return Arguments Array Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP Address of the system receiving audio.</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port on which receiving system is listening for audio.</td>
</tr>
</tbody>
</table>

GetSMSessionList

The GetSMSessionList method retrieves a list of all active SilentMonitor Sessions in an arguments array. If the number of elements returned is greater than 0, this indicates that the SilentMonitorManager currently is in an active Silent Monitor session.

Syntax

C++: Arguments * CIL_API GetSMSessionList(void)
COM: HRESULT GetSMSessionList([out,retval] IArguments **pIArguments );
VB: GetSMSessionList () as CTIOSCLIENTLIB.IArguments

Parameters

None.

Return Values

Arguments array that contains a list of all Silent Monitor sessions. The current version only allows one active session, so the main use for this function is to use the NumElements method on the returned arguments array to determine if the current SilentMonitorManager is in an active Silent Monitor session.
IsMonitoredTarget

The IsMonitoredTarget method determines if the specified device or agent is a target that is being monitored.

Syntax

C++: bool IsMonitoredTarget (Arguments & args);
COM: HRESULT IsMonitoredTarget ( /*[in]*/ IArguments * args,
    /*[out,retval]*/ VARIANT_BOOL * bMonitored );
VB: IsMonitoredTarget () As Boolean

Parameters

args

Arguments array that contains the parameter listed in Table 13-8:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the target object being monitored.</td>
</tr>
</tbody>
</table>

Return Value

True if the specified MonitoredUniqueObjectID corresponds to the monitored agent or device; False otherwise.

SetIPPhoneInfo

The SetIPPhoneInfo method saves the information of the IP Phone used by the client application.

Syntax

C++: int SetIPPhoneInfo (Arguments & args);
Methods

Chapter 13  SilentMonitorManager Object

COM: HRESULT SetIPPhoneInfo ( /*[in]*/ IArguments * args,
/*[out,retval]*/ int * errorcode );

VB: SetIPPhoneInfo (ByVal args as CTIOSCLIENTLIB.IArguments ) As Long

Parameters

args

Arguments array that can contain the parameters listed in Table 13-9:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientAddress</td>
<td>STRING</td>
<td>IP Address of the IP Phone to be used by the client application.</td>
</tr>
<tr>
<td>(required)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BitRate</td>
<td>INT</td>
<td>Audio transmission bit rate.</td>
</tr>
<tr>
<td>(optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PacketSize</td>
<td>INT</td>
<td>Number of milliseconds of audio stored in a packet.</td>
</tr>
<tr>
<td>(optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direction</td>
<td>SHORT</td>
<td>One of the following values that indicates the direction of voice flow</td>
</tr>
<tr>
<td>(optional)</td>
<td></td>
<td>between the calling party and the called party:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: Bidirectional</td>
</tr>
<tr>
<td>RTPType</td>
<td>SHORT</td>
<td>One of the following values that indicates the type of RTP messages</td>
</tr>
<tr>
<td>(optional)</td>
<td></td>
<td>between the calling party and the called party:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: audio</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: video</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: data</td>
</tr>
</tbody>
</table>
Methods

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

StartSilentMonitorRequest

The StartSilentMonitorRequest method sends a silent monitor session start request to a targeted client.

Syntax

C++: int StartSilentMonitorRequest (Arguments & args, unsigned short * SMSessionKey );
COM: HRESULT StartSilentMonitorRequest ( /*[in]*/ IArguments * args, /*[out]*/ unsigned short * SMSessionKey, /*[out,retval]*/ int * errorcode );
VB: StartSilentMonitorRequest int (ByVal args as CTIOSCLIENTLIB.IArguments, ByRef SMSessionKey AsLong) As Long

Table 13-9 SetIPPhoneInfo Arguments Array Parameters (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EchoCancelation</td>
<td>SHORT</td>
<td>One of the following values that indicates whether the echo cancellation feature is enabled on this IP Phone: 0: Off 1: On</td>
</tr>
<tr>
<td>(optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PayLoadType</td>
<td>SHORT</td>
<td>Audio codec type.</td>
</tr>
<tr>
<td>(optional)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parameters

args
Arguments array that can contain the parameters listed in Table 13-10: All parameters are required.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentID or DeviceID</td>
<td>STRING</td>
<td>AgentID or DeviceID of the target to monitor. Specify either an AgentID or a DeviceID, not both.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>ID of the peripheral associated with the agent or device.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the system receiving audio.</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port of the system where is listening for audio.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Interval in seconds between heartbeats.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Seconds elapsing before a Silent Monitor session is aborted because of no heartbeats received from the monitored peer.</td>
</tr>
</tbody>
</table>

SMSessionKey
An output parameter that contains the unique key to the started silent monitor session. This key must be used to perform any action on the currently active silent monitor session.

errocde
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values
Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
Remarks

If the DeviceID is used, there must be an agent associated with the device. The session will timeout if there is no agent logged into the device. An established silent monitor session will end if the associated agent logs out of the device.

StartSMMonitoredMode

The StartSMMonitoredMode method puts the SilentMonitorManager in Monitored mode.

Syntax

C++: int StartSMMonitoredMode (Arguments & args );
COM: HRESULT StartSMMonitoredMode ( /*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode );
VB: StartSMMonitoredMode (ByVal args as CTIOSCLIENTLIB.IArguments) As Long

Parameters

args

Arguments array that contains the following parameters listed in Table 13-11:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredDeviceID</td>
<td>STRING</td>
<td>DeviceID of the device to monitor (for example, the extension number of the IP phone to monitor.</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port of the system where that is listening for audio.</td>
</tr>
</tbody>
</table>

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”
Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

StartSMMonitoringMode

The StartSMMonitoringMode method puts the SilentMonitorManager in Monitoring mode.

Syntax

C++: int StartSMMonitoringMode (Arguments & reserved_args );  
COM: HRESULT StartSMMonitoringMode ( /*[in]*/ IArguments * reserved_args, /*[out,retval]*/ int * errorcode );  
VB: StartSMMonitoringMode (ByVal reserved_args as CTIOSCLIENTLIB.IArguments) As Long

Parameters

reserved_args

Not currently used, reserved for future use.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”.

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

StopSilentMonitorMode

The StopSilentMonitorMode method sets the SilentMonitorManager mode to eSMModeNotSet. If a silent monitor session is active at the time, the session will be terminated.
Chapter 13 SilentMonitorManager Object

Methods

Syntax

C++: int StopSilentMonitorMode (Arguments & reserved_args);
COM: HRESULT StopSilentMonitorMode ( /*[in]*/ IArguments * reserved_args, /*[out,retval]*/ int * errorcode );
VB: StopSilentMonitorMode (ByVal reserved_args as CTIOSCLIENTLIB.IArguments) As Long

Parameters

reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

StopSilentMonitorRequest

The StopSilentMonitorRequest method stops the Active silent monitor session.

Syntax

C++: int StopSilentMonitorRequest (Arguments & args);
COM: HRESULT StartSilentMonitorRequest ( /*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode );
VB: StopSilentMonitorRequest (ByVal args as CTIOSCLIENTLIB.IArguments) As Long

Parameters

args
Arguments array that contains the parameter listed in Table 13-12:
Chapter 13  SilentMonitorManager Object

Table 13-12 StopSilentMonitorRequest Arguments Array Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED</td>
<td>Unique key of the current active silent monitor session</td>
</tr>
<tr>
<td></td>
<td>SHORT</td>
<td></td>
</tr>
</tbody>
</table>

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
CTI OS Keywords and Enumerated Types

Keywords

The CTIOS Client Interface Library uses the Arguments structure to pass key-value pairs between the client and the server (see Chapter 12, “Helper Classes” for a detailed explanation of Arguments). Throughout this document all event and method parameter lists, as well as object properties, are listed with the keywords and the types associated with those keywords.

The expected (required and optional) keywords are referred to in this document by string name. For example, the Agent’s property for agent identifier is referred to as AgentID.

In addition to being able to use the string name for a keyword, programmers can take advantage of an enumeration of keywords as well.

Note

The enumeration of keywords is presently only available in the C++ CIL.

For each string keyword, a corresponding enumerated keyword exists. The enumerated keyword is the same name, preceded by the prefix ‘ekw’. For example, the AgentID string keyword is mapped to the enumerated keyword ekwAgentID.
Usage Example in C++:

```cpp
Arguments& args = Arguments::CreateInstance();
args.AddItem(ekwAgentID, "22866");
args.AddItem(ekwAgentInstrument, "23901");
pAgent->Login(args);
args.Release();
```

The complete set of standard keywords used in CTIOS can be found in the C++ header file “ctioskeywords.h”, located in the `\Distribution\cpp\Include` directory on the CTI OS toolkit media.

### Java CIL Keywords

For Java CIL, the CtiOs_IKeywordIDs interface contains a list of known Java CIL CTI OS keywords. See the Java CIL Javadoc file for more information.

### Enumerated Types

CTIOS employs enumerated types to provide symbolic names for commonly recurring values.

- In C++, Visual Basic, and COM, these are presented as enumerated types.

The complete set of enumerated types and their values can be found in the following locations:

- For C++ CIL using static libraries: the complete set of enumerated types is located in the C++ header file “cilmessages.h”, located in the `\Distribution\cpp\Include` directory on the CTIOS toolkit media.
- For COM (Visual Basic and Visual C++): the complete set of enumerated types is located in the CTIOSClient Type Library, which is compiled into the “CTIOSClient.dll” file, located in the `\Distribution\COM` directory on the CTIOS toolkit media.

In the Java CIL, the CTIOS_Enums interface contains the Java CIL enumerated types. See the Java CIL Javadoc file for more information.
CTI OS Logging

This appendix discusses a few issues related to CTI OS logging.

Taking CTI OS Client Logs

The trace log name and location for client processes can be found under the following registry keys:

HKEY_LOCAL_MACHINE\Software\Cisco Systems\CTIOS\Logging\TraceFileName

The default filename is CtiosClientLog. Logfiles will be created using the convention <TraceFileName>.mmdd.hhmmss.log. The files will be created in the current directory of the executing program, such as the directory into which the AgentDesktop is installed. You can provide a fully qualified path for the TraceFileName if you wish to store the files in a different location. For example, setting the value to

C:\Temp\CtiosClientLog

would put the logfiles in the directory C:\Temp using the naming convention CtiosClientLog.mmdd.hhmmss.log. Client trace files are simple ASCII text and can be opened with a conventional text editor such as Notepad.
How to Set Trace Levels

Trace levels for client processes, such as the Agent Desktop phone, can be found in the registry under:

HKEY_LOCAL_MACHINE\Software\Cisco Systems\Ctios\Logging\TraceMask

Warning

The default value for the trace masks is 0x03. Changing this value can have a serious impact on client performance. It should only be modified by experienced field personnel or at the request of Cisco support personnel.

Java CIL Logging and Tracing

In the Java CIL, the LogWrapper class implements the ILogListener interface and acts as a default logging mechanism. If a LogWrapper is created using the constructor that takes a string for a filename, the Java CIL performs logging and tracing to the specified file. If a LogWrapper is created using the constructor that takes no arguments, it traces to the System.out file.

Note

You must create a class that implements ILogListener, register it with the LogManager, and write the trace events to a logfile, or you must use the provided LogWrapper utility class to create logfiles. You will also need to provide a way to adjust the trace mask at runtime. If you encounter problems, Cisco personnel will need to see this log file in order to assist you with your problem.

See the Java CIL Javadoc file for information on the LogWrapper class and its associated methods.
INDEX

A

AcceptSilentMonitoring
SilentMonitorManager object method 13-4

Accessing properties and parameters 3-12

ActiveX Softphone Controls 5-1

AddItem
Arguments class method 12-14

Agent mode 2-6
agent object methods 9-12
  GetAgentState 9-18
  GetMonitoredAgent 9-19
  GetMonitoredCall 9-20
  GetSkillGroups 9-21
  IsAgent 9-24
  IsSupervisor 9-24
  Login 9-25
  Logout 9-27
  MakeCall 9-29
  QueryAgentState 9-36
  ReportBadCallLine 9-37
  RequestAgentTeamList 9-38
  RequestSupervisorAssist 9-39
  SendChatMessage 9-40
  SetAgentState 9-41

StartMonitoringAgent 9-44
StartMonitoringAgentTeam 9-45
StartMonitoringAllAgentTeams 9-46
StartMonitoringCall 9-47
StopMonitoringAgent 9-48
StopMonitoringAgentTeam 9-49
StopMonitoringAllAgentTeam 9-50
SuperviseCall 9-51

AgentSelectCtl softphone control 5-9
agent state
CTI application control 1-2
AgentStateCtl softphone control 5-5
AgentStatisticsCtl softphone control 5-10
Alternate call object method 10-10
AlternateCtl softphone control 5-10
Answer call object method 10-11
AnswerCtl softphone control 5-10
Architecture 2-1

Arg class 12-2
Arg class methods
  Clone 12-4
  CreateInstance 12-4
  DumpArg 12-5
  GetType 12-6
  GetValue 12-7
SetValue 12-9
Arguments class methods
  AddItem 12-14
  Clear 12-16
  Clone 12-17
  DumpArgs 12-18
  GetElement 12-19
  GetValue 12-21
  IsValid 12-23
  NumElements 12-24
  RemoveItem 12-26
  SetElement 12-26
  SetValue 12-28
Arguments structure 3-11
asynchronous events 1-3

B
BadLineCtl softphone control 5-11
Button controls 5-4
ButtonCtl softphone control 5-11
Button enablement masks 3-17

C
CallAppearanceCtl softphone control 5-11
call object methods 10-8
  Alternate 10-10
  Answer 10-11
  Clear 10-12
  ClearConnection 10-14
  Conference 10-15
  GetCallContext 10-17
  GetCallData 10-18
  Hold 10-23
  MakeConsultCall 10-24
  Reconnect 10-30
  Retrieve 10-31
  SendDTMFSignal 10-32
  SetCallData 10-35
  SingleStepConference 10-36
  SingleStepTransfer 10-39
  Snapshot 10-40
  StartRecord 10-41
  StopRecord 10-42
  Transfer 10-43
CCtiosException class 12-32
ChatCtl softphone control 5-13
CIL error codes 3-4
CIL object model 2-4
CILRefArg class methods
  GetType 12-30
  GetValue 12-31
  SetValue 12-31
Clear
  Arguments class method 12-16
  Clear call object method 10-12
  ClearConnection call object method 10-14
Client Interface Library (CIL) 2-2
Clone
Arg class method 12-4
Arguments class method 12-17
Coding conventions 3-1
COM CIL
using 4-13
COM CIL (C++) 4-16
Conference call object method 10-15
ConferenceCtl softphone control 5-13
Connect session object method 8-8
CreateInstance
Arg class method 12-4
Arguments class method 12-17
CreateSilentMonitorManager session object method 8-10
CreateWaitObject session object method 8-11
CTI-enabled applications 1-2
CTI OS
advantages 1-6
application architecture 1-6
architecture 2-1
benefits 1-7
CWaitObject class methods
DumpEventMask 12-37
GetMask 12-37
GetTriggerEvent 12-38
SetMask 12-39
WaitOnMultipleEvents 12-39
CWaitObject methods
InMask 12-38

D
DestroySilentMonitorManager session object method 8-12
DestroyWaitObject session object method 8-13
DisableSkillGroupStatistics Session object method 8-14
DisableSkillGroupStatistics SkillGroup object method 11-23
Disconnect session object method 8-15
DumpArg Arg class method 12-5
DumpArgs
Arguments class method 12-18
DumpEventMask CWaitObject class method 12-37

E
EmergencyAssistCtl softphone control 5-15
EnableSkillGroupStatistics Session object method 8-16
EnableSkillGroupStatistics SkillGroup object method 11-24
Event flow 1-3

G
GetAgentState agent object method 9-18
GetAllAgents session object method 8-17
Index

GetAllCalls session object method 8-17
GetAllSkillGroups session object method 8-18
GetCallContext call object method 10-17
GetCallData call object method 10-18
GetCurrentSilentMonitor session object method 8-21
GetElement Arguments class method 12-19
GetIPPhoneInfo SilentMonitorManager object method 13-6
GetMask CWaitObject class method 12-37
GetMonitoredAgent agent object method 9-19
GetMonitoredCall agent object method 9-20
GetObjectFromObjectID session object method 8-21
GetSessionInfo SilentMonitorManager object method 13-8
GetSkillGroups agent object method 9-21
GetSMSessionList SilentMonitorManager object method 13-10
GetTriggerEvent CWaitObject class method 12-38
GetType
   Arg class method 12-6
   CILRefArg class method 12-30
GetValue
   Arguments class method 12-21
   CILRefArg class method 12-31
GetValueType
   Arg class method 12-7
Grid controls 5-4
GridControl softphone control 5-16

H

Hold call object method 10-23
HoldCtl softphone control 5-16

I

IAgentEvents interface 6-63
ICallEvents interface 6-19
InMask CWaitObject method 12-38
Integrating an application 4-2
IsAgent agent object method 9-24
IsAgent session object method 8-23
ISessionEvents interface 6-2
ISkillGroupEvents interface. 6-102
IsMonitoredTarget SilentMonitorManager object method 13-11
IsSupervisor agent object method 9-24
IsSupervisor session object method 8-24
IsValid
   Arguments class method 12-23

L

Login agent object method 9-25
Login button 5-5
Logout agent object method 9-27
Logout button 5-6
LogToServer session object method 8-25
Index

OnMonitorModeEstablished event 6-16
OnReconnectCallConf event 6-57
OnRetrieveCallConf event 6-58
OnRTPStreamTimedoutEvent event 6-118
OnSendDTMFCconf event 6-59
OnSetAgentModeEvent event 6-89
OnSetAgentStateConf event 6-90
OnSilentMonitorSessionDisconnected event 6-117
OnSilentMonitorStartedEvent event 6-109
OnSilentMonitorStartRequestedEvent event 6-107
OnSilentMonitorStatusReportEvent event 6-112
OnSilentMonitorStopRequestedEvent event 6-111
OnSkillGroupStatisticsUpdated event 6-102
OnSnapshotCallConf event 6-59
OnSnapshotDeviceConf event 6-17
OnStartSilentMonitorConf event 6-114
OnStopSilentMonitorConf event 6-116
OnTransferCallConf event 6-62
Reconnect call object method 10-30
ReconnectCtl softphone control 5-18
RecordCtl softphone control 5-22
Reference counting 2-7
Referencing COM components 4-14
Registering for events in Visual Basic 4-15
RemoveItem
  Arguments class method 12-26
ReportBadCallLine agent object method 9-37
RequestAgentTeamList agent object method 9-38
RequestDesktopSettings session object method 8-27
Request parameters
  setting 3-13
Request-response mechanism 1-4
RequestSupervisorAssist agent object method 9-39
Retrieve call object method 10-31

S
screen pop 1-2
SendChatMessage agent object method 9-40
SendDTMFSignal call object method 10-32
Session object methods
  DisableSkillGroupStatistics 8-14
  EnableSkillGroupStatistics 8-16
  SetSupervisorSilentMonitorMode 8-32
session object methods 8-4

Q
QueryAgentState agent object method 9-36

R
Ready button 5-7
Connect 8-8
CreateSilentMonitorManager 8-10
CreateWaitObject 8-11
DestroySilentMonitorManager 8-12
DestroyWaitObject 8-13
Disconnect 8-15
GetAllAgents 8-17
GetAllCalls 8-17
GetAllSkillGroups 8-18
GetObjectFromObjectID 8-21
IsAgent 8-23
IsSupervisor 8-24
LogToServer 8-25
RequestDesktopSettings 8-27
SetAgent 8-28
SetCurrentCall 8-30
SetCurrentSilentMonitor 8-30
SetMessageFilter 8-31
session object methods
GetCurrentSilentMonitor 8-21
SetAgent session object method 8-28
SetAgentState agent object method 9-41
SetCallData call object method 10-35
SetCurrentCall session object method 8-30
SetCurrentSilentMonitor session object method 8-30
SetElement
Arguments class method 12-26
SetIPPhoneInfo SilentMonitorManager object method 13-11
SetMask CWaitObject class method 12-39
SetMessageFilter session object method 8-31
SetSupervisorSilentMonitorMode session object method 8-32
Setting object properties 3-13
Setting request parameters 3-13
SetValue
Arg class method 12-9
Arguments class method 12-28
SilentMonitorManager object methods
AcceptSilentMonitoring 13-4
GetIPPhoneInfo 13-6
GetSessionInfo 13-8
GetSMSessionList 13-10
IsMonitoredTarget 13-11
SetIPPhoneInfo 13-11
StartSilentMonitorRequest 13-13
StartSMMonitorMode 13-15
StartSMMonitoringMode 13-16
StopSilentMonitorMode 13-16
StopSilentMonitorRequest 13-17
SingleStepConference call object method 10-36
SingleStepTransfer call object method 10-39
SkillGroup object methods 11-22
DisableSkillGroupStatistics 11-23
EnableSkillGroupStatistics 11-24
SkillgroupStatisticsCtl softphone control 5-19
Snapshot call object method 10-40
softphone controls
AgentSelectCtl 5-9
AgentStateCtl       5-5
AgentStatisticsCtl  5-10
AlternateCtl        5-10
AnswerCtl           5-10
BadLineCtl          5-11
ButtonCtl           5-11
CallAppearanceCtl   5-11
ChatCtl             5-13
ConferenceCtl       5-13
EmergencyAssistCtl  5-15
GridControl         5-16
HoldCtl             5-16
MakeCallCtl         5-17
RecordCtl           5-22
SkillgroupStatisticsCtl  5-19
StatusBarCtl        5-19
SupervisorOnlyCtl   5-20
TransferCtl         5-22
StartMonitoringAgent agent object method 9-44
StartMonitoringAgentTeam agent object method 9-45
StartMonitoringAllAgentTeams agent object method 9-46
StartMonitoringCall agent object method 9-47
StartRecord call object method 10-41
StartSilentMonitorRequest SilentMonitorManager object method 13-13
StartSMMonitoredMode SilentMonitorManager object method 13-15
StartSMMonitoringMode SilentMonitorManager object method 13-16
Static libraries    4-19
StatusBarCtl softphone control 5-19
STLport             4-24
StopMonitoringAgent agent object method 9-48
StopMonitoringAgentTeam agent object method 9-49
StopMonitoringAllAgentTeams agent object method 9-50
StopRecord call object method 10-42
StopSilentMonitorMode SilentMonitorManager object method 13-16
StopSilentMonitorRequest SilentMonitorManager object method 13-17
Subscribing for events in C++ 4-24
SuperviseCall agent object method 9-51
SupervisorOnlyCtl softphone control 5-20

third-party call control 1-3
Transfer call object method 10-43
TransferCtl softphone control 5-22
Index

U

UniqueObjectID  3-14

V

Visual Basic  4-13

W

WaitOnMultipleEvents CWaitObject class
   method  12-39

Work Not Ready button  5-8

Work Ready button  5-8