THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT
NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT
ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR
THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE
INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU
ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A
COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as
part of UCB’s public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE
PROVIDED “AS IS” WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED
OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL
DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR
INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH
DAMAGES.

CCIP, the Cisco Powered Network mark, the Cisco Systems Verified logo, Cisco Unity, Follow Me Browsing, FormShare, Internet Quotient, iQ
Breakthrough, iQ Expertise, iQ FastTrack, the iQ Logo, iQ Net Readiness Scorecard, Networking Academy, ScriptShare, SMARTnet, TransPath,
and Voice LAN are trademarks of Cisco Systems, Inc.; Changing the Way We Work, Live, Play, and Learn. Discover All That’s Possible, The Fastest
Way to Increase Your Internet Quotient, and iQuick Study are service marks of Cisco Systems, Inc.; and Aironet, ASIST, BPX, Catalyst, CCDA,
CCDP, CCIE, CCNA, CCNP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, the Cisco IOS logo, Cisco Press, Cisco Systems, Cisco
Systems Capital, the Cisco Systems logo, Empowering the Internet Generation, Enterprise/Solver, EtherChannel, EtherSwitch, Fast Step, GigaStack,
IOS, IP/TV, LightStream, MGX, MICA, the Networkers logo, Network Registrar, Packet, PIX, Post-Routing, Pre-Routing, RateMUX, Registrar,
SlideCast, StrataView Plus, Stratm, SwitchProbe, TeleRouter, and VCO are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the
U.S. and certain other countries.

All other trademarks mentioned in this document or Web site are the property of their respective owners. The use of the word partner does not imply
a partnership relationship between Cisco and any other company. (0203R)

Cisco ICM Software CTI OS Developer’s Guide
Copyright © 2001-2002, Cisco Systems, Inc.
All rights reserved.
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

Request-Response Paradigm 1-4

Obtaining Documentation xx

World Wide Web xx

Documentation CD-ROM xx

Ordering Documentation xx

Documentation Feedback xxi

Obtaining Technical Assistance xxi

Cisco.com xxii

Technical Assistance Center xxii

About This Guide xvii

Purpose xvii

Audience xvii

Conventions xvii

Organization xviii

Other Publications xx
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  3-7
  Generic Interfaces  3-8
    Arguments  3-8
    Accessing Properties and Parameters with GetValue  3-9
    Setting Request Parameters with AddItem  3-10
Setting Object Properties with SetValue  3-10
UniqueObjectID  3-11
Obtaining Objects from UniqueObjectIDs  3-13
Using Button Enablement Masks  3-14

CHAPTER 4

Building Your Application  4-1
Integrating your Application with CTI OS via the CIL  4-1
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  4-6
  Building a Simple Softphone with ActiveX Controls  4-6
  Adding a Hook for Screenpops  4-9
Using the COM CIL in Visual Basic  4-12
  Referencing COM Components in Visual Basic  4-13
  Registering for Events in Visual Basic  4-14
  Next Steps  4-14
Using the COM CIL in C++  4-15
  Adding COM Support to your Application  4-15
  Using the CIL Dynamic Link Libraries  4-16
  Creating an Instance of a COM Object  4-16
  Subscribing and Unsubscribing to COM Events in C++  4-17
  Next Steps  4-18
Using the C++ CIL and static libraries  4-18
  Header Files and Libraries  4-18
  Project Settings for Compiling and Linking  4-19
Contents

Subscribing for Events in C++ 4-22
STLport 4-22
Next Steps 4-23
Getting Connected to CTI OS Server 4-24
  Creating a Session 4-24
  Connecting to CTI OS 4-24
  Setting the Connection Mode 4-25
  Disconnecting from CTI OS Server 4-26
Deployment of Custom CTI OS Applications 4-27
  Applications Using the ActiveX Controls 4-27
  Applications Using COM (But Not ActiveX Controls) 4-30
  Deployment of C++ Applications 4-31

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 5-8
    AgentStatisticsCtl 5-9
    AlternateCtl 5-9
    AnswerCtl 5-10
    BadLineCtl 5-10
    CallAppearanceCtl 5-10
    ChatCtl 5-12
    ConferenceCtl 5-12
    EmergencyAssistCtl 5-14
HoldCtl  5-15
MakeCallCtl  5-16
ReconnectCtl  5-17
SkillgroupStatisticsCtl  5-18
StatusBarCtl  5-18
SupervisorOnlyCtl  5-19
RecordCtl  5-20
TransferCtl  5-21

CHAPTER 6
Event Interfaces and Events  6-1
Event Publication Model  6-1
ISessionEvents Interface  6-2
   OnConnection  6-2
   OnConnectionClosed  6-3
   OnConnectionFailure  6-3
   OnConnectionRejected  6-4
   OnCTIOSFailure  6-4
   OnCurrentAgentReset  6-6
   OnCurrentCallChanged  6-7
   OnGlobalSettingsDownloadConf  6-7
   OnHeartbeat  6-7
   OnMissingHeartbeat  6-8
   OnMonitorModeEstablished  6-9
   OnSetAgentMode  6-10
ICallEvents Interface  6-10
   OnAlternateCallConf  6-11
   OnAnswerCallConf  6-11
   OnCallBegin  6-12
   OnCallEnd  6-15
   OnCallDataUpdate  6-15
OnCallDelivered 6-18
OnCallEstablished 6-20
OnCallHeld 6-22
OnCallRetrieved 6-22
OnCallCleared 6-23
OnCallConnectionCleared 6-24
OnCallOriginated 6-25
OnCallFailed 6-26
OnCallTransferred 6-27
OnCallConferenced 6-30
OnCallDiverted 6-33
OnAgentPrecallEvent 6-34
OnAgentPrecallAbortEvent 6-37
OnCallServiceInitiatedEvent 6-38
OnCallQueuedEvent 6-40
OnCallDequeuedEvent 6-42
OnCallReachedNetworkEvent 6-43
OnCallStartRecordingConf 6-45
OnCallStopRecordingConf 6-45
OnClearCallConf 6-46
OnClearConnectionConf 6-46
OnConferenceCallConf 6-47
OnControlFailureConf 6-47
OnConsultationCallConf 6-48
OnDeflectCallConf 6-49
OnHoldCallConf 6-49
OnReconnectCallConf 6-50
OnRetrieveCallConf 6-50
OnSendDTM FConf 6-51
OnSetCallDataConf 6-51
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnSnapshotCallConf</td>
<td>6-52</td>
</tr>
<tr>
<td>OnTransferCallConf</td>
<td>6-54</td>
</tr>
<tr>
<td>OnCallTranslationRouteEvent</td>
<td>6-55</td>
</tr>
<tr>
<td>IAgentEvents Interface</td>
<td>6-57</td>
</tr>
<tr>
<td>OnAgentStateChange</td>
<td>6-57</td>
</tr>
<tr>
<td>OnAgentStatistics</td>
<td>6-59</td>
</tr>
<tr>
<td>OnChatMessage</td>
<td>6-69</td>
</tr>
<tr>
<td>OnControlFailureConf</td>
<td>6-70</td>
</tr>
<tr>
<td>OnMakeCallConf</td>
<td>6-71</td>
</tr>
<tr>
<td>OnPostLogout</td>
<td>6-72</td>
</tr>
<tr>
<td>OnPreLogout</td>
<td>6-73</td>
</tr>
<tr>
<td>OnSetAgentStateConf</td>
<td>6-75</td>
</tr>
<tr>
<td>OnSnapshotDeviceConf</td>
<td>6-76</td>
</tr>
<tr>
<td>OnLogoutFailed</td>
<td>6-77</td>
</tr>
<tr>
<td>OnNewAgentTeamMember</td>
<td>6-78</td>
</tr>
<tr>
<td>OnEmergencyCall</td>
<td>6-79</td>
</tr>
<tr>
<td>OnStartMonitoringAgent</td>
<td>6-80</td>
</tr>
<tr>
<td>OnStopMonitoringAgent</td>
<td>6-81</td>
</tr>
<tr>
<td>OnQueryAgentStateConf</td>
<td>6-81</td>
</tr>
<tr>
<td>OnAgentInfoEvent</td>
<td>6-83</td>
</tr>
<tr>
<td>OnSilentMonitorStartRequest</td>
<td>6-84</td>
</tr>
<tr>
<td>OnSilentMonitorStopRequest</td>
<td>6-84</td>
</tr>
<tr>
<td>SkillGroupEvents Interface</td>
<td>6-85</td>
</tr>
<tr>
<td>OnSkillGroupStatisticsUpdated</td>
<td>6-85</td>
</tr>
<tr>
<td>IButtonEnablementEvents</td>
<td>6-86</td>
</tr>
<tr>
<td>OnButtonEnablementChange</td>
<td>6-86</td>
</tr>
<tr>
<td>OnSupervisorButtonChange</td>
<td>6-89</td>
</tr>
<tr>
<td>IMonitoredAgentEvents Interface</td>
<td>6-90</td>
</tr>
<tr>
<td>IMonitoredCallEvents Interface</td>
<td>6-90</td>
</tr>
</tbody>
</table>
CHAPTER 7

CtiOs Object 7-1

Methods 7-1

DumpProperties 7-2
GetAllProperties 7-3
GetElement 7-4
GetNumProperties 7-4
GetProperty 7-5
GetPropertyType 7-6
GetValue 7-7
GetValueArray 7-8
GetValueInt 7-9
GetValueString 7-10
IsValid 7-11

CHAPTER 8

Session Object 8-1

Session Object Properties 8-2

Methods 8-4

Connect 8-6
Disconnect 8-7
DumpProperties 8-7
GetAllAgents 8-8
GetAllCalls 8-8
GetAllProperties 8-9
GetAllSkillGroups 8-9
GetCurrentAgent 8-10
GetCurrentCall 8-11
GetElement 8-11
GetNumProperties 8-11
GetObjectFromObjectID 8-12
GetProperty 8-13
GetPropertyType  8-13
GetValue Methods  8-13
IsAgent  8-13
IsSupervisor  8-14
IsValid  8-14
RequestDesktopSettings  8-14
SetAgent  8-15
SetCurrentCall  8-16
SetMessageFilter  8-17
Notes On Message Filters  8-18
Message Filter Syntax  8-18
A Simple Example  8-19
General Form of Filter Syntax  8-19
Combining Filters  8-19
Filtering for Specific Events  8-20

CHAPTER 9

Agent Object  9-1
Agent Object Properties  9-1
Agent Statistics  9-2
Methods  9-11
Arguments Parameters  9-13
DisableAgentStatistics  9-13
DisableSkillGroupStatistics  9-14
EnableAgentStatistics  9-15
EnableSkillGroupStatistics  9-16
GetAgentState  9-17
GetAllProperties  9-17
GetElement  9-17
GetMonitoredAgent  9-17
GetMonitoredCall  9-18
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 10-7
  Arguments Parameters 10-9
Alternate 10-10
Answer 10-11
Clear 10-12
ClearConnection 10-13
Conference 10-14
Deflect 10-16
GetCallContext 10-17
GetCallData 10-19
GetValue Methods 10-20
Hold 10-20
MakeConsultCall 10-22
Reconnect 10-28
Retrieve 10-29
SendDTMFSignal 10-30
SetCallData 10-32
SingleStepConference 10-33
SingleStepTransfer 10-36
Snapshot 10-37
StartRecord 10-38
StopRecord 10-39
Transfer 10-40

CHAPTER 11  SkillGroup Object 11-1
  Properties 11-2
  Statistics 11-3
  Methods 11-24
    DisableSkillGroupStatistics 11-25
DumpProperties  11-26
EnableSkillGroupStatistics  11-26
GetElement  11-27
GetValue Methods  11-27
IsValid  11-27
SetValue  11-27

CHAPTER  12  Helper Classes  12-1
Arg Class  12-2
  AddRef  12-3
  Clone  12-3
 .CreateInstance  12-4
  Dump Arg  12-5
  GetType  12-6
  GetValue Methods  12-7
  Release  12-8
  SetValue  12-9
Arguments Class  12-10
  Usage Notes  12-12
  AddItem  12-13
  AddRef  12-14
  Clear  12-15
  Clone  12-15
  CreateInstance  12-16
  DumpArgs  12-17
  GetElement Methods  12-18
  GetValue Methods  12-19
  IsValid  12-22
  NumElements  12-22
  Release  12-23
Contents

RemoveItem  12-24
SetElement   12-24
SetValue     12-26

CILRefArg Class (C++ only)  12-27
GetType      12-28
GetValue     12-28
SetValue     12-29

CCtiOsException Class  12-29
CCtiosException Constructor  12-30
getCode       12-31
getStatus     12-31
getString     12-32
what          12-32

APPENDIX A

CTI OS Keywords and Enumerated Types  A-1
Keywords    A-1
Enumerated Types  A-2

APPENDIX B

Sample C++ Application  B-1

APPENDIX C

Sample Visual Basic Application  C-1

APPENDIX D

Sample COM for C++ Application  D-1

APPENDIX E

CTI OS Logging  E-1
Taking CTI OS Client Logs  E-1
How to Set Trace Levels  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 usages 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.
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>

**About This Guide**

**Format**

- **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**.
### About This Guide

**Organization**

<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 and events associated with the CTI OS Session object.</td>
</tr>
<tr>
<td>Chapter 9, “Agent Object”</td>
<td>Describes the methods and events associated with the CTI OS Agent object.</td>
</tr>
<tr>
<td>Chapter 10, “Call Object”</td>
<td>Describes the methods and events associated with the CTI OS Call object.</td>
</tr>
<tr>
<td>Chapter 11, “SkillGroup Object”</td>
<td>Describes the methods and events associated with the CTI OS SkillGroup object.</td>
</tr>
<tr>
<td>Chapter 12, “Helper Classes”</td>
<td>Describes the methods and events associated with the CTI OS Arguments classes.</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, “Sample C++ Application”</td>
<td>Lists a sample CTI OS C++ application.</td>
</tr>
<tr>
<td>Appendix D, “Sample COM for C++ Application”</td>
<td>Lists a sample CTI OS COM for C++ application.</td>
</tr>
<tr>
<td>Appendix E, “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

The following sections provide sources for obtaining documentation from Cisco Systems.

World Wide Web

You can access the most current Cisco documentation on the World Wide Web at the following sites:

- http://www.cisco.com
- http://www-china.cisco.com
- http://www-europe.cisco.com

Documentation CD-ROM

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

Ordering Documentation

Cisco documentation is available in the following ways:

- Registered Cisco Direct Customers can order Cisco Product documentation from the Networking Products MarketPlace:
  http://www.cisco.com/cgi-bin/order/order_root.pl
About This Guide

Obtaining Technical Assistance

- Registered Cisco.com users can order the Documentation CD-ROM through the online Subscription Store:
  http://www.cisco.com/go/subscription
- Nonregistered Cisco.com users can order documentation through a local account representative by calling Cisco corporate headquarters (California, USA) at 408 526-7208 or, in North America, by calling 800 553-NETS(6387).

Documentation Feedback

If you are reading Cisco product documentation on the World Wide Web, you can submit technical comments electronically. Click Feedback in the toolbar and select Documentation. After you complete the form, click Submit to send it to Cisco.

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

To submit your comments by mail, use the response card behind the front cover of your document, or write to the following address:

Attn Document Resource Connection
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-9883

We appreciate your comments.

Obtaining Technical Assistance

Cisco provides Cisco.com as a starting point for all technical assistance. Customers and partners can obtain documentation, troubleshooting tips, and sample configurations from online tools. For Cisco.com registered users, additional troubleshooting tools are available from the TAC website.
Cisco.com

Cisco.com is the foundation of a suite of interactive, networked services that provides immediate, open access to Cisco information and resources at anytime, from anywhere in the world. This highly integrated Internet application is a powerful, easy-to-use tool for doing business with Cisco.

Cisco.com provides a broad range of features and services to help customers and partners streamline business processes and improve productivity. Through Cisco.com, you can find information about Cisco and our networking solutions, services, and programs. In addition, you can resolve technical issues with online technical support, download and test software packages, and order Cisco learning materials and merchandise. Valuable online skill assessment, training, and certification programs are also available.

Customers and partners can self-register on Cisco.com to obtain additional personalized information and services. Registered users can order products, check on the status of an order, access technical support, and view benefits specific to their relationships with Cisco.

To access Cisco.com, go to the following website:

http://www.cisco.com

Technical Assistance Center

The Cisco TAC website is available to all customers who need technical assistance with a Cisco product or technology that is under warranty or covered by a maintenance contract.

Contacting TAC by Using the Cisco TAC Website

If you have a priority level 3 (P3) or priority level 4 (P4) problem, contact TAC by going to the TAC website:

http://www.cisco.com/tac
P3 and P4 level problems are defined as follows:

- **P3**—Your network performance is degraded. Network functionality is noticeably impaired, but most business operations continue.
- **P4**—You need information or assistance on Cisco product capabilities, product installation, or basic product configuration.

In each of the above cases, use the Cisco TAC website to quickly find answers to your questions.

To register for Cisco.com, go to the following website:

http://www.cisco.com/register/

If you cannot resolve your technical issue by using the TAC online resources, Cisco.com registered users can open a case online by using the TAC Case Open tool at the following website:

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

**Contacting TAC by Telephone**

If you have a priority level 1 (P1) or priority level 2 (P2) problem, contact TAC by telephone and immediately open a case. To obtain a directory of toll-free numbers for your country, go to the following website:


P1 and P2 level problems are defined as follows:

- **P1**—Your production network is down, causing a critical impact to business operations if service is not restored quickly. No workaround is available.
- **P2**—Your production network is severely degraded, affecting significant aspects of your business operations. No workaround is available.
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.

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, 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. For advanced integrations, programmers can subclass CTI OS objects to develop their own customized behavior.

- **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. Subclassable event adapter classes enable programmers to subscribe to event interfaces and 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 2-1 CTIOS Object Server and Client Object Sharing](image-url)

![Figure 2-2 Client Interface Library Three-Tiered Architecture](image-url)
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. It also employs a watchdog thread to recover from these failures automatically.

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

CIL Object Model

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.”
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 an error will be logged, but 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.

In C++ only, 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 12.
- For sample CTI OS applications, see Appendixes B through D.
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). 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.
- **SHORT**: This is a 16-bit wide 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>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td>std::string OR const char *</td>
<td>String</td>
<td>BSTR</td>
</tr>
<tr>
<td>INT</td>
<td>long OR int</td>
<td>Long</td>
<td>long OR int</td>
</tr>
<tr>
<td>SHORT</td>
<td>short</td>
<td>Integer</td>
<td>short</td>
</tr>
<tr>
<td>BOOL</td>
<td>bool</td>
<td>Boolean</td>
<td>VARIANT_BOOL</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>Arguments</td>
<td>Arguments</td>
<td>IArguments *</td>
</tr>
<tr>
<td>ARG</td>
<td>Arg</td>
<td>Arg</td>
<td>IArg *</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 above. The possible value of the error code returned from CIL methods are defined in Table 3-2.

<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>
</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_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 CCtiOsObject 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 CCtiOsObject 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, CSupervisor or CWaitObject.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_AGENT</td>
<td>-112</td>
<td>No valid agent.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_CALL</td>
<td>-113</td>
<td>No valid call.</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_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_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>
</tbody>
</table>
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:

```c++
// 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
' 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 3-3:
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:

```vbnet
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
```

The Arguments structure can store and retrieve all native C/C++/Visual Basic types, as well as nested Arguments structures.

### Accessing Properties and Parameters with GetValue

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:

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

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

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.
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++
// C++
string sAgentID = “22866”;
args.AddItem(“AgentID”, sAgentID);
args.AddItem(CTIOSAGENTID, sAgentID); // alternative
args.AddItem(ekwAgentID, sAgentID); // alternative
```

```vb
'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++
// C++
Agent a;
a.SetValue(“AgentID”, “22866”);
a.SetValue(CTIOSAGENTID, “22866”); // alternative
a.SetValue(ekwAgentID, “22866”); // alternative
```

```vb
'Visual Basic
Dim a As Agent
a.SetValue “AgentID”, “22866”
```
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 3-4 explains the composition of the UniqueObjectID.

<table>
<thead>
<tr>
<th>Table 3-4 UniqueObjectID Components</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object Type</strong></td>
</tr>
<tr>
<td>Call Object</td>
</tr>
<tr>
<td>Agent Object</td>
</tr>
<tr>
<td>Device Object (for events only; no CIL object)</td>
</tr>
</tbody>
</table>
Note

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) 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:**

```c++
string sUniqueObjectID = "call.5000.101.23901";
Ccall * pCall = NULL;
m_pSession->GetObjectFromObjectID(sUniqueObjectID,
    (CCtiosObject**)&pCall);

pCall->Clear();
pCall->Release(); // release our reference to this object
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
        nBitMask = pArguments.GetValueInt "EnablementMask"
    End If
    ' do bitwise comparison
    If nBitMask And ENABLE_ANSWER Then
        m_AnswerButton.Enable
    End If
End Sub
```
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 bitmasks 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.

Integrating your Application with CTI OS via the CIL

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 control 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.

- **C++.** The CTI OS Client Interface Library for C++ is the appropriate choice for building a high-performance application running on a Windows platform in a C++ development environment. The C++ CIL is distributed as a set of
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. These samples, located in the Toolkit\Samples directory, include:
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).
Chapter 4  Building Your Application

Using The CTI OS ActiveX Controls

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.

Figure 4-1  Components Dialog in Visual Basic Listing CTIOS ActiveX Controls

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).
Chapter 4 Building Your Application

Using The CTI OS ActiveX Controls

Figure 4-4 CTI OS References Needed for Visual Basic COM Programming

Available References:

- ComPlus 1.0 Cатаoog Replication Type Library
- Control Wizard 1.0 Type Library
- cooldoc 1.0 Type Library
- cryptext 1.0 Type Library
- CSScd
- CSSed Utilities
- CTIOS Session Resolver 1.0 Type Library
- CTCCommonLigs 1.0 Type Library
- CTIOS Arguments 1.0 Type Library
- CTIOS _ad 1.0 Type Library
- CTIOS _ad 1.0 Type Library
- CTIOSClient 1.0 Type Library
- CusReg 1.0 Type Library
- Data Object Wizard

' 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.GetValueString("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.GetValueString("Callvariable1")

    ' copy call variable1 to the clipboard
    Clipboard.SetText m_callvar1
End Function
```
Chapter 4  Building Your Application

Using the COM CIL in Visual Basic

- **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** (ctiossessionresolver.dll). The Session Resolver is used to share a single CTI OS Session across multiple compilation unites (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 to Session object when your application closes:

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 “Connecting to CTI OS”.

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:

- For detailed information on the CTI OS Client Interface Library objects, see Chapters 8 through 12.
- For a complete sample application written in Visual Basic, see Appendix C, “Sample Visual Basic Application.”
Microsoft Foundation Classes (MFC). The following header file are required for MFC applications to use COM: *afxwin.h*, *afxext.h*, *afxdisp.h*, and *afxdtctl.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: *atbase.h*.

**Note**
The sample application in Appendix D, “Sample COM for C++ Application” is built using the MFC class support. Either MFC or COM is acceptable for using the CIL for COM in a C++ application.

### Using the CIL Dynamic Link Libraries

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

```csharp
#import '..\..\Distribution\COM\ctiossessionresolver.dll' using namespace CTIOSSESSIONRESOLVERLib;

#import '..\..\Distribution\COM\ctiosclient.dll' using namespace CTIOSCLIENTLib;
```

**Note**
You must register three DLLs, but you do not need to import the *arguments.dll* into your project since it is imported by the *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.

```csharp
// Create SessionResolver and Session object
hRes = m_pSessionResolver.CreateInstance (OLESTR("CTIOSSessionResolver.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 example in Appendix D, “Sample COM for C++ 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. (For a complete example of an event sink, refer to the COM CIL sample for C++ in Appendix D, “Sample COM for C++ Application.”)

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

```cpp
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:

```cpp
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 “Getting Connected to 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 Appendix D, “Sample COM for C++ Application.”

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.

```c++
#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.

---

**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, ws2_32.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 “Preprocessor,” 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:

```c++
// 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:

```c++
// 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://wwwsgi.com/tech/stl.

Next Steps

- For detailed information on the CTI OS client start up and shut down sequence, see the section “Connecting to CTI OS”.
- 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 interface with C++ static libraries, see Appendix B, “Sample C++ Application.”
Getting Connected to CTI OS Server

One of the first things a client application should do is establish a connection to the CTI OS Server. The CIL provides the interface to this functionality, and handles automatic recovery from connection failures. A client application can connect in one of two possible modes:

- **Agent mode.** The client receives events (agent state change and call related) for a specific agent.
- **Monitor mode.** The client specifies a filter expression that selects the types of events that the client application will receive.

Creating a Session

The Session object is responsible for establishing and maintaining a connection to an active CTI OS server. In addition to connecting, its two most important functions are automatic recovery from connection failures and the filtering and distribution of events. It receives events (determined by the connection mode) that are broadcast by the CTI OS server. For each event, it determines whether that event applies to an agent, an existing call, or a new call (which requires the creation of a new call object).

Connecting to CTI OS

The first step towards setting up a connection to CTI OS Server is creating a Session object and calling the `Connect()` method on it. At a minimum, the name of a machine running CTI OS Server is required as a parameter for the `Connect()` request. See Chapter 8, “Session Object” for details on the `Connect()` and other Session methods.

After calling the `Connect()` method, the client should wait for an asynchronous event that indicates if the request succeeded. The `OnConnection` event indicates a successful completion of the connect request; the `OnConnectionFailure` and `OnConnectionRejected` events indicate that the request did not succeed. The logs should be a starting point for the developer to figure out why a connection request did not succeed. The location of the client logfile is determined by the registry key:

```
KEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems\CTIOS\Logging\TraceFileName.
```
The following VB code sample demonstrates the creation of a Session object and using it to connect to CTI OS:

```vba
Dim WithEvents m_session As CTIOSCLIENTLib.Session
Set m_session = New Session 'Create a Session object
Dim m_Args As New Arguments
  'Assemble arguments for the connect request
  m_Args.AddItem "CtiosA", "YourServerNameA"
  m_Args.AddItem "portA", 42028
  m_Args.AddItem "CtiosB", " YourServerNameB"
  m_Args.AddItem "portB", 42028
  m_Args.AddItem "heartbeat", 100
  'Send the Connect request
  m_session.Connect m_Args
```

### Setting the Connection Mode

Once a connection to the CTI OS server has been established, the client should specify the connection mode for this session. The client application can connect in Agent mode or Monitor mode.

#### Agent Mode

In this mode the client receives events for a specific agent, including agent events and call events for calls on that agent’s instrument, along with system events. This is the mode that will be used by client applications developed for the Agent’s desktop. The `SetAgent()` method on the Session object issues a request to select this mode. The `OnSetAgentMode` event indicates that the request succeeded and now the client can proceed with logic for Agent functionality (e.g. a `Login()` request). An `OnCtiosFailure` event is received if the request did not succeed.

Following is a code sample for specifying the mode for the client application. A good place for this code is the `OnConnection` event that ensures that a connection has been successfully established:

```vba
Private Sub m_session_OnConnection(ByVal pDispParam As Object)
  'Set up the agent object
  Dim m_Agent As CTIOSCLIENTLib.Agent
```
Set m_Agent = New Agent
//Set up the agent object
m_Agent.SetValue "agentid", "275"
m_Agent.SetValue "peripheralid", "5002"
m_session.SetAgent m_Agent

**Monitor Mode**

In this mode, the client provides a filter expression that specifies the events that it (the client) is interested in. For more information on valid Message Filters, see “Message Filter Syntax” in Chapter 8, “Session Object.” These events could be related to calls, agent state (for one or more agents) or statistics. Monitor mode also allows agent and call control functionality for objects known to the session (determined by the event filter).

The SetMessageFilter() method on the Session object specifies Monitor mode implicitly and the event filter is detailed in the input parameters. The OnMonitorModeEstablished event indicates that Monitor mode was successfully established and the client can proceed with rest of logic for a Monitor type application. An OnCtiosFailure event is received if the request did not succeed.

Setting the mode for the client application is an important part of connecting to CTI OS Server. In case of a failure that results in the connection going down, the Session object will automatically reconnect with a redundant CTI OS server. Upon successfully reconnecting, it will also reestablish the connection mode so the application can be returned to the state that it was in before the failure occurred. This includes resetting the current agent (if in Agent mode) and the event filter (in Monitor mode).

**Disconnecting from CTI OS Server**

Disconnecting from CTI OS Server (via the Disconnect() method) before shutting down is an important part of the client application functionality. The Disconnect() method closes the socket connection between the client application and CTI OS. On most switches, it does not log the agent out. If no logout request was issued before the Disconnect(), then on most switches the agent stays logged into the instrument even after the client application has shut down.
Upon `Disconnect()` each object maintained by the Session (Call, Skillgroup, Wait) is released and no further events are received. Cleaning up the Agent object is the developer’s responsibility since it was handed to the Session (via the `SetAgent()` method.

Code sample:
```csharp
m_session.Disconnect
```

## 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 4-1.

**Table 4-1 ActiveX Control DLLs**

<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>
<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>chatctl.dll</td>
<td>Chat ActiveX control</td>
</tr>
<tr>
<td>conferencectl.dll</td>
<td>Conference ActiveX control</td>
</tr>
<tr>
<td>cticommondlns.dll</td>
<td>Common Dialogs utility COM object</td>
</tr>
<tr>
<td>CTIOSAgentStatistics.dll</td>
<td>AgentStatistics 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 4-2 lists the dependencies of CTI OS ActiveX controls.

### Table 4-1  ActiveX Control DLLs (continued)

<table>
<thead>
<tr>
<th>DLL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>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 4-2 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, cticommondlg.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>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, cticommondlg.dll</td>
</tr>
<tr>
<td>cticommondlg.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, cticommondlg.dll</td>
</tr>
<tr>
<td>ctiosclient.dll</td>
<td>ATL.dll, arguments.dll</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, Gridcontrol.dll</td>
</tr>
<tr>
<td>ctiosstatusbar.dll</td>
<td>Atl.dll, ctiosclient.dll, arguments.dll, cticommondlg.dll</td>
</tr>
<tr>
<td>EmergencyAssistCtl.dll</td>
<td>ATL.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll</td>
</tr>
<tr>
<td>gridcontrol.dll</td>
<td>ATL.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
- CtiосSessionresolver.dll (only if used – see discussion above)
- ATL.DLL (only if not already available on target system)

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).
Deployment of Custom CTI OS Applications

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\sysnt32 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. Therefore, there is no need to ship additional components other than the linked executable.
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 Supervisor Desktop 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 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>AgentRelatedControls</td>
<td></td>
</tr>
<tr>
<td>AgentStateCtl</td>
<td>Provides UI to 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.

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).

RecordCtl 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.
Table 5-1  CTI OS ActiveX Controls (continued)

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
</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
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.

Figure 5-1   Login Dialog

![Login Dialog](image-url)
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.

![Figure 5-2 Reason Code Dialog for Logout](image)

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](image)

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>2 3 4</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 agent state 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

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:

![Answer Icon](image)

Release Icon:

![Release Icon](image)

**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.

![Bad Line Icon](image)

**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).

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.
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
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**

![Dial Dialog](image)

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**

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.

![SkillGroupStatisticsCtl Displaying Sample Data for Three Skillgroups](image)

StatusBarCtl

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

![StatusBar Control Displaying Sample Data](image)

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.
Chapter 5  CTI OS ActiveX Softphone Controls

CTI OS ActiveX Control Descriptions

- 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.

**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 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

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
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 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.
**ISessionEvents Interface**

- **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.
- **IAllInOne.** This interface publishes the complete set of events available from the CIL. This is also the default dispatch interface used by scripting clients, such as Visual Basic.

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.

**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.
Chapter 6  Event Interfaces and Events

ISessionEvents Interface

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)
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.
Chapter 6  Event Interfaces and Events

ISessionEvents Interface

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>
<tr>
<td>SystemEventArg1</td>
<td>INT</td>
<td>Present only if SystemEventID equals SysPeripheralOnline or SysPeripheralOffline. Contains the peripheral ID of the affected peripheral.</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>STRING</td>
<td>An error message.</td>
</tr>
</tbody>
</table>

Following are the enumerated values for Failure Code:

```c
enum enumCTIOS_FailureCode
{
    eDriverOutOfService     = 1,
    eServiceNotSupported    = eDriverOutOfService + 1,
    eOperationNotSupported  = eServiceNotSupported + 1,
    eInvalidPrivilege       = eOperationNotSupported + 1,
    eUnknownRequestID       = eInvalidPrivilege + 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:

```cpp
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
}
```

**OnCurrentAgentReset**

The OnCurrentAgentReset message is generated when the current agent is removed from the session.

**Syntax**

**C++:**
```cpp
void OnCurrentAgentReset(Arguments& args)
```

**COM:**
```cpp
void OnCurrentAgentReset (IArguments * args)
```

**VB:**
```cpp
session_OnCurrentAgentReset (ByVal args As CtiosCLIENTLib.IArguments)
```

**Parameters**

`args`

Arguments array containing the following fields.
Chapter 6  Event Interfaces and Events

ISessionEvents Interface

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 old current agent that was just removed.</td>
</tr>
</tbody>
</table>

OnGlobalSettingsDownloadConf

Private; for internal use only.

OnHeartbeat

The OnHeartbeat event is generated when a heartbeat response is received from a CTI OS server. It returns the time of day.
### ISessionEvents Interface

#### Syntax

**C++:**

```cpp
void OnHeartbeat(Arguments& args)
```

**COM:**

```cpp
void OnHeartbeat (IArguments * args)
```

**VB:**

```vbnet
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) the connection to a new CTI OS server is automatically restarted.

#### Syntax

**C++:**

```cpp
void OnMissingHeartbeat(Arguments& args)
```

**COM:**

```cpp
void OnMissingHeartbeat (IArguments * args)
```

**VB:**

```vbnet
session_OnMissingHeartbeat (ByVal args As CtiosCLIENTLib.IArguments)
```

#### Parameters

**args**

Arguments array containing the following fields.
Chapter 6      Event Interfaces and Events

IISessionEvents Interface

OnMonitorModeEstablished

The OnMonitorModeEstablished event is generated when Monitor Mode is established or released. There is no corresponding OnAgentModeEstablished. Agent mode can be identified by the first OnAgentStateChange event. The event returns the current state of the Monitor Mode, either SERVER_MONITORMODE, or SERVER_MONITORMODE_EXIT.

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>EventTime</td>
<td>INT</td>
<td>Integer value with time of day expressed in milliseconds.</td>
</tr>
<tr>
<td>Consecutive MissedHeartbeats</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>

<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>Mode</td>
<td>INT</td>
<td>SERVER_MONITORMODE or SERVER_MONITORMODE_EXIT</td>
</tr>
</tbody>
</table>
OnSetAgentMode

The OnSetAgentMode event indicates that the client has made a successful AgentMode connection.

Syntax

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

Parameters

args
Not used, reserved for future use.

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 CTIServer.

Syntax

C++:   int OnAlternateCallConf (Arguments & args);
COM:  HRESULT OnAlternateCallConf ([in] IArguments * args);
VB:  Session_ OnAlternateCallConf (ByVal args as CTOISCLIENTLIB.IArguments)

Parameters

args
Not used; reserved for future use.

OnAnswerCallConf

The OnAnswerCallConf event is fired to the client to indicate that an Answer request was received by the CTIServer.

Syntax

C++:   int OnAnswerCallConf (Arguments & args);
COM:  HRESULT OnAnswerCallConf ([in] IArguments * args);
VB:  Session_ OnAnswerCallConf (ByVal args as CTOISCLIENTLIB.IArguments)

Parameters

args
Not used; reserved for future use.
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. The event contains the changed call data.

**Note**

There can be multiple calls with the same ConnectionCallID value.

**Syntax**

C++: `void OnCallBegin(Arguments& args)`

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>
</tbody>
</table>
### ICallEvents Interface

#### RouterCallKeyDay INT
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 INT
The call key created by the ICM. The ICM resets this counter at midnight.

#### 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.

<table>
<thead>
<tr>
<th><strong>Keyword</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></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>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**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<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 UniqueID (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

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)

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>ICMEnterpriseUniqueID (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++:  void OnCallDataUpdate(Arguments& args)
COM:  void OnCallDataUpdate (IArguments * args)
VB:  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>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>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>
</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</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>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>
</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.

**Syntax**

**C++:**

```cpp
void OnCallDelivered(Arguments& args)
```

**COM:**

```cpp
void OnCallDelivered (IArguments * args)
```

### OnCallDelivered

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.

### ECC (optional)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

### CTIClients (Optional)

<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:</td>
</tr>
<tr>
<td>(Optional)</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>

### ICMEnterprise UniqueID (Optional)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMEnterprise</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>
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>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>
</tbody>
</table>
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.

<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>ICMEnterpriseUniqueID</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>

**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.

Note

Under certain switches, when a call is made outside of the ACD, this event may not be received. See `OnCallReachedNetwork` for more detail.
### 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)
Chapter 6  Event Interfaces and Events

ICallEvents Interface

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.
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.

<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>
Chapter 6  Event Interfaces and Events

ICallEvents Interface

ICallEvents Interface

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>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>
</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.
Chapter 6   Event Interfaces and Events

ICallEvents Interface

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>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</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>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>
</tbody>
</table>
### ICallEvents Interface

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<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>
</tbody>
</table>

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>
</tbody>
</table>
### ICallEvents Interface

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>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>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The number of the agent SkillGroup the call is attributed to, as known to</td>
</tr>
<tr>
<td>(Optional)</td>
<td></td>
<td>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 the call is attributed to.</td>
</tr>
<tr>
<td>(Optional)</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>SkillGroupPriority</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not</td>
</tr>
<tr>
<td>(Optional)</td>
<td></td>
<td>applicable or not available.</td>
</tr>
<tr>
<td>CallWrapupData</td>
<td>STRING</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>(Optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CallVariable1</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>(Optional)</td>
<td></td>
<td></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>(Optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>(optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECC (optional)</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains all of the Expanded Call Context variables in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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

OnCallDiverted

The removal of a call from one delivery target and forwarded to a different target may generate an OnCallDiverted event.

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>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: CTIClient[1] CTIClientSignature CTIClientTimestamp</td>
</tr>
<tr>
<td>ICMEnterpriseUnique ID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>
### ICallEvents Interface

#### OnAgentPrecallEvent

**Note**

The `OnAgentPrecallEvent` event is applicable to IPCC only. The equivalent on all other TDM events is `TranslationRouteEvent`.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of <code>ConnectionDeviceID</code> 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 <code>NULL_SERVICE</code> 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 <code>NULL_SERVICE</code> 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 <code>DivertingDeviceID</code> field.</td>
</tr>
<tr>
<td>CalledDeviceType</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the <code>CalledDeviceID</code> field.</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>DivertingDeviceID (Optional)</td>
<td>STRING</td>
<td>The device identifier of the device from which the call was diverted.</td>
</tr>
<tr>
<td>CalledDeviceID (Optional)</td>
<td>STRING</td>
<td>The device identifier of the device to which the call was diverted.</td>
</tr>
</tbody>
</table>
The OnAgentPrecallEvent event is a pre-call indication that 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 an 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 OnAgentPrecallEvent(Arguments& args)

**COM:**
void OnAgentPrecallEvent (IArguments * args)

**VB:**
session_OnAgentPrecallEvent (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>
### ICallEvents Interface

<table>
<thead>
<tr>
<th>Field</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>AgentInstrument</td>
<td>STRING</td>
<td>The agent instrument that the call will be routed to.</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>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral.</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>SkillGroupNumber</td>
<td>INT</td>
<td>The number of the agent SkillGroup 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>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>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>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

ICallEvents Interface

OnAgentPrecallAbortEvent

Note

The OnAgentPrecallAbortEvent event is applicable to IPCC only.

The OnAgentPrecallAbortEvent event is received only if a previously indicated routing (OnAgentPrecallEvent) decision is reversed. The contact is uniquely identified by the ICMEnterpriseUniqueID, which is a field based on the ICM’s 64-bit unique key (RouterCallKeyDay and RouterCallKeyCallID). Upon receipt of an OnAgentPrecallAbortEvent, any data pre-fetch work that was started on an OnAgentPrecallEvent should be cleaned up.

Syntax

C++:  void OnAgentPrecallAbortEvent (Arguments* args)
COM:  void OnAgentPrecallAbortEvent (IArguments * args)
VB:   session_OnAgentPrecallAbortEvent (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>CallerEnteredDigits</th>
<th>STRING</th>
<th>The digits entered by the caller in response to IVR prompting.</th>
</tr>
</thead>
<tbody>
<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>
ICallEvents Interface

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++:**
```cpp
void OnCallServiceInitiatedEvent (Arguments & args)
```

**COM:**
```cpp
void OnCallServiceInitiatedEvent (IArguments * args)
```

**VB:**
```vb
session_OnCallServiceInitiatedEvent (ByVal args As CtiosCLIENTLib.IArguments)
```

---

### Keyword Type Description

<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>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKey CallID 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>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>The agent instrument that the call will be routed to.</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 (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</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

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>LastRedirectedDeviceID</td>
<td>STRING</td>
<td>The device identifier of the redirecting device.</td>
</tr>
<tr>
<td>LastRedirectedDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the LastRedirectDeviceID.</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>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>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

ICallEvents Interface

OnCallDequeuedEvent

The explicit removal of a call from the ACD queue may generate an OnCallDequeuedEvent message to the CTI Client.

Syntax

C++:    void OnCallDequeuedEvent (Arguments& args)
COM:     void OnCallDequeuedEvent (IArguments * args)
VB:       session_OnCallDequeuedEvent (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>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maximum of 20. This value also indicates the number of SkillGroup Number,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SkillGroupId and SkillGroup Priority floating fields present in the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>floating part of the message.</td>
</tr>
<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>LocalConnection State</td>
<td>SHORT</td>
<td>The state of the local end of the connection.</td>
</tr>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

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.
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>
</tbody>
</table>
OnCallStartRecordingConf

The OnCallStartRecordingConf event is fired to the client to indicate that a StartRecord request was received by the CTIServer.

Syntax

C++:    int OnCallStartRecordingConf (Arguments & args);
COM:    HRESULT OnCallStartRecordingConf ([in] IArguments * args);
VB:    Session_OnCallStartRecordingConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args
Not used; reserved for future use.

OnCallStopRecordingConf

The OnCallStopRecordingConf event is fired to the client to indicate that a StopRecord request was received by the CTIServer.

Syntax

C++:    int OnCallStopRecordingConf (Arguments & args);
COM:    HRESULT OnCallStopRecordingConf ([in] IArguments * args);
VB:    Session_OnCallStopRecordingConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args
Not used; reserved for future use.
OnClearCallConf

The OnClearCallConf event is fired to the client to indicate that a Clear request was received by the CTIServer.

Syntax

C++: int OnClearCallConf (Arguments & args);
COM: HRESULT OnClearCallConf ([in] IArguments * args);
VB: Session_ OnClearCallConf (ByVal args as
    CTIOSCLIENTLIB.IArguments)

Parameters

args

Not used; reserved for future use.

OnClearConnectionConf

The OnClearConnectionConf event is fired to the client to indicate that a ClearConnection request was received by the CTIServer.

Syntax

C++: int OnClearConnectionConf (Arguments & args);
COM: HRESULT OnClearConnectionConf ([in] IArguments * args);
VB: Session_ OnClearConnectionConf (ByVal args as
    CTIOSCLIENTLIB.IArguments)

Parameters

args

Not used; reserved for future use.
OnConferenceCallConf

The OnConferenceCallConf event is fired to the client to indicate that a ConferenceCall or SingleStepConferenceCall request was received by the CTIServer.

Syntax

C++: int OnConferenceCallConf (Arguments & args);
COM: HRESULT OnConferenceCallConf ([in] IArguments * args);
VB: Session_OnConferenceCallConf (ByVal args as CtiOSCLIENTLIB.IArguments)

Parameters

args

Not used; reserved for future use.

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: Session_OnControlFailureConf (ByVal args As CtiOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following fields.
The OnConsultationCallConf event is fired to the client to indicate that a MakeConsultCall request was received by the CTIServer.

**Syntax**

**C++:**
```cpp
int OnConsultationCallConf (Arguments & args);
```

**COM:**
```cpp
HRESULT OnConsultationCallConf ([in] IArguments * args);
```

**VB:**
```vbnet
Session_ OnConsultationCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)
```

**Parameters**

`args`

Not used; reserved for future use.

<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>PeripheralError</td>
<td>INT</td>
<td>Peripheral-specific error data, if available. Zero otherwise.</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>
<tr>
<td>MessageType</td>
<td>INT</td>
<td>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</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>STRING</td>
<td>String text containing the description of the failure.</td>
</tr>
</tbody>
</table>
OnDeflectCallConf

The OnDeflectCallConf event is fired to the client to indicate that a DeflectCall request was received by the CTIServer.

Syntax

C++: int OnDeflectCallConf (Arguments & args);
COM: HRESULT OnDeflectCallConf ([in] IArguments * args);
VB: Session_ OnDeflectCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Not used; reserved for future use.

OnHoldCallConf

The OnHoldCallConf event is fired to the client to indicate that a Hold request was received by the CTIServer.

Syntax

C++: int OnHoldCallConf (Arguments & args);
COM: HRESULT OnHoldCallConf ([in] IArguments * args);
VB: Session_ OnHoldCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Not used; reserved for future use.
OnReconnectCallConf

The OnReconnectCallConf event is fired to the client to indicate that a Reconnect request was received by the CTIServer.

Syntax

C++:    int OnReconnectCallConf (Arguments & args);
COM:     HRESULT OnReconnectCallConf ([in] IArguments * args);
VB:      Session_.OnReconnectCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args
Not used; reserved for future use.

OnRetrieveCallConf

The OnRetrieveCallConf event is fired to the client to indicate that a RetrieveCall request was received by the CTIServer.

Syntax

C++:    int OnRetrieveCallConf (Arguments & args);
COM:     HRESULT OnRetrieveCallConf ([in] IArguments * args);
VB:      Session_.OnRetrieveCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args
Not used; reserved for future use.
OnSendDTMFFConf

The OnSendDTMFFConf event is fired to the client to indicate that a SendDTMF request was received by the CTIServer.

Syntax

C++: int OnSendDTMFFConf (Arguments & args);
COM: HRESULT OnSendDTMFFConf ([in] IArguments * args);
VB: Session_OnSendDTMFFConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args
Not used; reserved for future use.

OnSetCallDataConf

The OnSetCallDataConf event is fired to the client to indicate that a SetCallData request was received by the CTIServer. An OnCallDataUpdate event indicates that the call’s data has actually been updated.

Syntax

C++: int OnSetCallDataConf (Arguments & args);
COM: HRESULT OnSetCallDataConf ([in] IArguments * args);
VB: Session_OnSetCallDataConf (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.

<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>ConnectionCallID</td>
<td>INT</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>ANI</td>
<td>STRING</td>
<td>The calling line ID of the caller.</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>
### 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</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 (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>
OnTransferCallConf

The OnTransferCallConf event is fired to the client to indicate that a TransferCall or SingleStepTransferCall request was received by the CTIServer.

Syntax

C++:       int OnTransferCallConf (Arguments & args);
COM:       HRESULT OnTransferCallConf ([in] IArguments * args);

Keyword | Type    | Description
--------|---------|-------------
ECC    | 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
RouterCallKeyDay | INT    | 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.
RouterCalKeyCallID | INT | The call key created by the ICM. The ICM resets this counter at midnight.
EnablementMask    | INT    | Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.
UniqueObjectID    | STRING | An object ID that uniquely identifies the call object.
CallStatus        | SHORT  | The current status of the call.
ICallEvents Interface

VB: Session_OnTransferCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Not used; reserved for future use.

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.
### ICallEvents Interface

<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>
<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>RouterCalKeyCallID</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>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>
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

C++: void OnAgentStateChange(Arguments& args)
COM: void OnAgentStateChange (IArguments * args)
VB: 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>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>
</tbody>
</table>
### SkillGroupID
**Type:** 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
**Type:** INT  
The number of seconds since the agent entered this state (typically 0).

### SkillGroupPriority
**Type:** SHORT  
The priority of the skill group, or 0 when skill group priority is not applicable or not available.

### EventReasonCode
**Type:** SHORT  
A peripheral-specific code indicating the reason for the state change.

### SkillGroupState
**Type:** SHORT  
Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.

### AgentID
**Type:** STRING  
The agent’s ACD login ID.

### AgentExtension
**Type:** STRING  
The agent’s ACD teleset extension.

### CTIClientSignature (Optional)
**Type:** STRING  
The Client Signature of the CTI Client that is associated with this agent.

### Enablement Mask
**Description:** Contains the bit-mask that specifies what buttons can be enabled or disabled when the agent is on this state.

### UniqueObjectID
**Type:** STRING  
A unique object ID for the agent object.

### AgentInstrument
**Type:** STRING  
The agent’s ACD instrument number.

### 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>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

IAgentEvents Interface

## OnAgentStatistics

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).

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).

### Table 6-1  AgentState Values (continued)

<table>
<thead>
<tr>
<th>enum Value</th>
<th>Description</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<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>
Chapter 6  Event Interfaces and Events

IAgentEvents Interface

Syntax

**C++:**  
```
int OnAgentStatistics (Arguments & args);
```

**COM:**  
```
HRESULT OnAgentStatistics ([in] IArguments * args);
```

**VB:**  
```
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><strong>Session Statistics</strong></td>
<td></td>
<td></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>AgentOutCallsTalkTime</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>
</tbody>
</table>
### IAgentEvents Interface

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentOutCallsTime</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</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</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</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</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</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>IncomingCallsHeld</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</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</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>AutoOutCallsTime 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>
<td>INT</td>
</tr>
<tr>
<td>AutoOutCallsHeld Session</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>AutoOutCallsHeldTime Session</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold.</td>
<td>INT</td>
</tr>
</tbody>
</table>
## IAgentEvents Interface

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviewCallsSession</td>
<td>Total number of outbound Preview calls completed by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>PreviewCallsTalkTime 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>
<td>INT</td>
</tr>
<tr>
<td>PreviewCallsTime Session</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>PreviewCallsHeld Session</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>PreviewCallsHeldTime Session</td>
<td>Total number of seconds outbound Preview calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>ReservationCallsSession</td>
<td>Total number of agent reservation calls completed by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td>ReservationCallsTalk TimeSession</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>
### IAgentEvents Interface

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

### Today Statistics

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<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>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>Event Description</td>
<td>Description</td>
<td>Data Type</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td><strong>AgentOutCallsTalkTime</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>
<td>INT</td>
</tr>
<tr>
<td><strong>AgentOutCallsTime</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>
<td>INT</td>
</tr>
<tr>
<td><strong>AgentOutCallsHeldToday</strong></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><strong>AgentOutCallsHeldTime</strong></td>
<td>Total number of seconds outbound ACD calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td><strong>HandledCallsToday</strong></td>
<td>The number of inbound ACD calls handled by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td><strong>HandledCallsTalkTimeToday</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>
<td>INT</td>
</tr>
<tr>
<td><strong>HandledCallsAfterCallTimeToday</strong></td>
<td>Total after call work time in seconds for Inbound ACD calls counted as 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>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>
<tr>
<td>AutoOutCallsToday</td>
<td>Total number of AutoOut (predictive) calls completed by the agent.</td>
<td>INT</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><strong>AutoOutCallsTalkTimeToday</strong></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><strong>AutoOutCallsTimeToday</strong></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><strong>AutoOutCallsHeldToday</strong></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><strong>AutoOutCallsHeldTimeToday</strong></td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td><strong>PreviewCallsToday</strong></td>
<td>Total number of outbound Preview calls completed by the agent.</td>
<td>INT</td>
</tr>
<tr>
<td><strong>PreviewCallsTalkTimeToday</strong></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>
</tbody>
</table>
### IAgentEvents Interface

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<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>
<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>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

IAgentEvents Interface

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationCallsHeld</td>
<td>Total number of seconds agent reservation calls were placed on hold.</td>
<td>INT</td>
</tr>
<tr>
<td>TimeToday</td>
<td></td>
<td></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 (required)</td>
<td>The agent’s ACD teleset extension.</td>
<td>STRING</td>
</tr>
<tr>
<td>AgentID (required)</td>
<td>The agent’s ACD login ID.</td>
<td>STRING</td>
</tr>
<tr>
<td>AgentInstrument (required)</td>
<td>The agent’s ACD instrument number.</td>
<td>STRING</td>
</tr>
</tbody>
</table>

**OnChatMessage**

The OnChatMessage event is generated when an asynchronous text message from another user (agent) is received.

**Syntax**

**C++:**
```cpp
void OnChatMessage(Arguments& args)
```

**COM:**
```cpp
void OnChatMessage(IArguments * args)
```

**VB:**
```cpp
session_OnChatMessage (ByVal args As CtiosCLIENTLib.IArguments)
```

**Parameters**

`args`

Arguments array containing the following fields.
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.

<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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See CtiLink.h for a list of the Control Failure Codes.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID that represents a specific client.</td>
</tr>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

IAgentEvents Interface

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>NewConnectionDeviceIDType</td>
<td>Indicates the type of the connection identifier supplied in the NewConnectionDeviceID floating field.</td>
<td>SHORT</td>
</tr>
</tbody>
</table>
The OnPostLogout event is generated after the agent has logged out.

**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 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>
</tbody>
</table>
Chapter 6      Event Interfaces and Events

IAgentEvents Interface

## 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

```cpp
void OnPreLogout(Arguments& args)
```
IAgentEvents Interface

**Parameters**

```
COM:    void OnPreLogout (IArguments * args)
VB:    session_OnPreLogout (ByVal args As CtiosCLIENTLib.IArguments)
```

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>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

IAgentEvents Interface

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

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.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>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>
</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.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumCalls</td>
<td>The number of active calls associated with this device, up to a maximum of 16.</td>
<td>SHORT</td>
</tr>
<tr>
<td>CallConnectionCallID (optional)</td>
<td>The Call ID value assigned to one of the calls. There may be more than one CallConnectionCallID field in the message (see NumCalls).</td>
<td>INT</td>
</tr>
<tr>
<td>CallConnectionDeviceID Type (optional)</td>
<td>Indicates the type of the connection identifier supplied in the following CallConnectionDeviceID floating field. There may be more than one CallConnectionDeviceIDType field in the message (see NumCalls). This field always immediately follows the corresponding CallConnectionCallID field.</td>
<td>SHORT</td>
</tr>
</tbody>
</table>
OnLogoutFailed

The OnLogoutFailed ia 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>
</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)
COM:    void OnNewAgentTeamMember (IArguments * args)
VB:     session_OnNewAgentTeamMember (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>ConfigOperation</td>
<td>SHORT</td>
<td>When this optional parameter is provided in the event it describes the realized operation on the team member. The possible values are: 0x0001 CONFIG_OPERATION_ADD_AGENT = 0x0002 CONFIG_OPERATION_REMOVE_AGENT = 0x0002</td>
</tr>
</tbody>
</table>
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>
<tr>
<td>ConnectionDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of the connection identifier supplied in the 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>
</tbody>
</table>
### IAgentEvents Interface

#### 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)
  ```

**Parameters**

- **args**
  Arguments array containing the following fields.

**Keyword** | **Type** | **Description**
--- | --- | ---
AgentExtension (Required) | STRING | The Agent’s teleset extension.
AgentID (required) | STRING | The Agent’s ACD login in.
AgentInstrument (required) | STRING | The Agent’s ACD instrument number.

**Remarks**

This is a Supervisor specific event.
OnStopMonitoringAgent

The OnStopMonitoringAgent event is generated monitoring of an agent is dropped in response to a StopMonitoringAgent() request.

Syntax

C++: void OnStopMonitoringAgent (Arguments& args)
COM: void OnStopMonitoringAgent (IArguments * args)
VB: 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>Agent ID for the agent to stop monitoring.</td>
</tr>
</tbody>
</table>

Remarks

This is a Supervisor specific event.

OnQueryAgentStateConf

The OnQueryAgentStateConf event is generated as a response to the QueryAgentState() 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>
</tbody>
</table>

Each SkillGroup[jj] field in the message contains the following information

<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 an agent SkillGroup queue that the call has been added to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
</tbody>
</table>
Chapter 6      Event Interfaces and Events

IAgentEvents Interface

OnAgentInfoEvent

The OnAgentInfoEvent event is generated as a response to a query to the Agent Lookup Service and carries the agent’s name.

Syntax

C++:    void OnAgentInfoEvent (Arguments& args)
COM:    void OnAgentInfoEvent (IArguments * args)
VB:     session_OnAgentInfoEvent (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>A unique object ID for the agent object</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>
</tbody>
</table>
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>

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)
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 CtiосCLIENTLib.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>
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.

<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 on which the agent resides.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The number of the agent skill group as known to the peripheral. Must contain the special value NULL_SKILL_GROUP when not available.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The ICM SkillGroupID of the skill group. May contain the special value NULL_SKILL_GROUP when not available.</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>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 6-2.</td>
</tr>
</tbody>
</table>

Table 6-2  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>
<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>Enablement Event</td>
<td>Bit Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------</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_SET_AGENT_LOGOUT</td>
<td>0xFFFFFFFE</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_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>Enablement Event</th>
<th>Bit Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE_SET_AGENT_LOGOUT</td>
<td>0x00000001</td>
</tr>
<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>0xFFFFFFFFFD</td>
</tr>
<tr>
<td>DISABLE_SILENTMONITOR</td>
<td>0xFFFFFFFFFB</td>
</tr>
</tbody>
</table>
Chapter 6  Event Interfaces and Events

IButtonEnablementEvents

Table 6-2  Table of Enablement Bits (continued)

<table>
<thead>
<tr>
<th>Enablement Bits</th>
<th>Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE_BARGE_IN</td>
<td>0xFFFFFFFB</td>
</tr>
<tr>
<td>DISABLE_INTERCEPT</td>
<td>0xFFFFFFFF7</td>
</tr>
<tr>
<td>DISABLE_CLEAR</td>
<td>0xFFFFFFFFF</td>
</tr>
<tr>
<td>DISABLE_SUPERVISE_CALL</td>
<td>DISABLE_BARGE_IN, DISABLE_INTERCEPT, DISABLE_CLEAR, DISABLE_SUPERVISE_CALL, 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>

**OnSupervisorButtonChange**

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.

**Parameters**

args

Arguments array containing the following fields.
**IMonitoredAgentEvents Interface**

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);`

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupervisorBtn</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 6-2.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td></td>
</tr>
</tbody>
</table>

**IMonitoredCallEvents Interface**

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 CTIIOS_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);
- OnMonitoredCallConfereced([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);
IMonitoredCallEvents Interface
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.

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.
DumpProperties

The DumpProperties method returns all the properties of the object in string “name = value” format.

Syntax

**C++:**
```
string DumpProperties ()
```

**COM:**
```
HRESULT DumpProperties (/*[out,retval]*/ BSTR* bstrValue)
```

**VB:**
```
DumpProperties () As String
```

Parameters

*bstrValue*

The output parameter (return parameter in VB) containing a string listing the names and values of the object’s properties.

Table 7-1  **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>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>
Chapter 7      CtiOs Object

Methods

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

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.”
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

C++: 
Arg& GetElement (string& key, int element)
Arg& GetElement (int key, int element)
Arg& GetElement (char* key, int element)

COM: 
HRESULT GetElement /*[in]*/ VARIANT* key, /*[in]*/ int element, /*[out,retval]*/ IArg** pIArg)

VB: 
GetElement (key As VARIANT) As CTIOSCLIENTLib.IArg

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.

GetNumProperties

The GetNumProperties method returns the number of properties of that object.
Methods

Chapter 7      CtiOs Object

Syntax

C++: int GetNumProperties ()
COM: HRESULT GetNumProperties ([out,retval]*/ int * num
VB: GetNumProperties () As Long

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

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:**

```vbnet
HRESULT GetPropertyType(/*[in]*/ VARIANT* key, /*[out,retval]*/ int* value)
```

**VB:**

```vbnet
GetPropertyType (key As VARIANT) As Int
```

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>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARG_NOTSET</td>
<td>0</td>
<td>Argument type not determined</td>
</tr>
<tr>
<td>ARG_INT</td>
<td>1</td>
<td>Signed integer</td>
</tr>
</tbody>
</table>
Chapter 7  CtiOs Object

Methods

GetValue

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++:  
Arg& GetValue (string& key)
Arg& GetValue (int key)
Arg& GetValue (char* key)

COM:  
HRESULT GetValue ([in]/*VARIANT* key, [out,retval]*/
IArg** value)

VB:  
GetValue (key As VARIANT) As CTIOSCLIENTLib.IArg

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.

<table>
<thead>
<tr>
<th>ARG_SHORT</th>
<th>4</th>
<th>2 bytes signed integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARG_BOOL</td>
<td>5</td>
<td>1 byte integer</td>
</tr>
<tr>
<td>ARG_STRING</td>
<td>6</td>
<td>STL character string</td>
</tr>
<tr>
<td>ARG_ARGARRAY</td>
<td>7</td>
<td>Variable length array of Arg</td>
</tr>
</tbody>
</table>
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++:**
```c++
Arg& GetValueArray (string& key)
Arg& GetValueArray (enum_Keywords key)
Arg& GetValue (char * key)
```

**COM:**
```c
HRESULT GetValueArray (/*[in]*/ VARIANT * key, /*[out,retval]*/
IArguments ** value)
```

**VB:**
```vbnet
GetValueArray (key As VARIANT) As CTIOSCLIENTLib.IArguments
```

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.

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)  
int GetValueInt (char* key)  

COM:  
HRESULT GetValueInt /*[in]*/ VARIANT* key, /*[out,retval]*/  
int* value)  

VB:  
GetValueInt (key As VARIANT) As Integer

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.
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++:**
```cpp
string GetValueString (string& key)
string GetValueString (int key)
string GetValueString (char* key)
```

**COM:**
```cpp
HRESULT GetValueString (/*[in]*/ VARIANT* key, /*[out,retval]*/
BSTR* value)
```

**VB:**
```vbnet
GetValueString (key As VARIANT) As String
```

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.
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

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.
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 “Getting Connected to 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,
Session Object Properties

The remainder of this chapter describes the data properties and interface methods of the Session object.

### 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>

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 (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>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>
<tr>
<td>PortB</td>
<td>INT</td>
<td>TCP/IP port for ctiosB.</td>
</tr>
</tbody>
</table>
Methods

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>Connect</td>
<td>Establishes a connection to a CTI OS server.</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>
</tbody>
</table>
### Table 8-2  Session Object Methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>IsAgentMode</td>
<td>Checks the current mode and returns true if agent mode.</td>
</tr>
<tr>
<td>IsSupervisorMode</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>RequestDesktopSettings</td>
<td>Sends a message request to the CTI OS Server to retrieve the desktop settings configured for this site.</td>
</tr>
<tr>
<td>SetAgent</td>
<td>Sets an agent to a session object.</td>
</tr>
<tr>
<td>SetCurrentCall</td>
<td>Associates the current call to a session object.</td>
</tr>
<tr>
<td>SetMessageFilter</td>
<td>Sets a message filter that the client application must pass in order to connect in Monitor Mode,</td>
</tr>
<tr>
<td>SetValue</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
</tbody>
</table>
Connect

The Connect method establishes a connection to a CTI OS server.

Syntax

**C++:**
```cpp
int Connect(Arguments& args)
```

**COM:**
```cpp
HRESULT Connect(IArguments *args, int * errorcode)
```

**VB:**
```vbnet
Connect(args As CTIOSCLIENTLib.IArguments) As Long
```

Parameters

**args**

An arguments array containing the connection parameters listed in Table 8-3.

**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.”

<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>
Remarks

A successful request will result in an OnConnection event. Otherwise, an OnConnectionFailure event results.

Disconnect

The Disconnect method disconnects the open connection to the CTI OS server.

Syntax

C++: void Disconnect (Arguments& args);
COM: HRESULT Disconnect /* [in, optional */ IArguments *args,
     /*[out]*/ int * errorcode );
VB: Disconnect(args As CTIOSCLIENTLib.IArguments) As Long

Parameters

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.”

Dump Properties

Methods

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)
```

Parameters

**args**

- **C++:** 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.”

GetAllCalls

The GetAllCalls method provides a list of all call objects that the session knows about.
Chapter 8      Session Object

Methods

Syntax

C++: int GetAllCalls (Arguments *args)
COM: HRESULT GetAllCalls(/*[out, retval]*/ VARIANT *args)
VB: GetAllCalls (args As VARIANT)

Parameters

args

C++: 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++: int GetAllSkillGroups (Arguments *args)
COM: HRESULT GetAllSkillGroups(/*[out, retval]*/ VARIANT *args)
VB: GetAllSkillGroups (args As VARIANT)
Parameters

args

  **C++:** 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.”

GetCurrentAgent

The GetCurrentAgent method returns the Agent specified when the Agent Node connection was established. Use this method rather than GetValue(“CurrentAgent”).

Syntax

  **C++:** int GetCurrentAgent()
  **COM:** HRESULT GetCurrentAgent(/*[out, retval]*/ IAgent *agent)
  **VB:** GetCurrentAgent () As CTIOSCLIENTLib.IAgent

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.
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

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.

getElement

See Chapter 7, “CtiOs Object” for a description of the getElement method.

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,
   CClisObject ** 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

Parameters

uniqueObjectID
A string reference that contains the UniqueObjectID of the requested Call, Agent, or Skillgroup object.

object
A pointer to either a CClisObject 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.

Remarks

Many events use UniqueObjectIDs instead of the objects themselves. Use this method to get the object if it is necessary for processing.
GetPropertyName

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.

GetValue 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

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

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.

IsValid


RequestDesktopSettings

The RequestDesktopSettings method sends a request to the CTI OS Server to download the configuration settings defined for a desktop application.
Chapter 8  Session Object

Methods

Syntax

**C++:**  
```cpp
int RequestDesktopSettings(Arguments& args)
```

**COM:**  
```cpp
HRESULT RequestDesktopSettings(/* [in] */ IArguments *args,
    /* [out] */ int *errorcode)
```

**VB:**  
```vb
RequestDesktopSettings (args As CTIOSCLIENTLib.IArguments) As Long
```

Parameters

**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:
- `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.
Methods

Syntax

**C++:**    int SetAgent(CAgent& agent)
**COM:**    HRESULT SetAgent(/*[in]*/IAgent *agent, /*[out, retval]*/ int * errorcode)
**VB:**    SetAgent (agent As CTIOSCLIENTLib.IAgent) As Long

Parameters

agent

The agent to be assigned to the Session object.

errocode

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 SetAgent request results in an OnSetAgentMode event.

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]*/
**VB:**    SetCurrentCall (call As CTIOSCLIENTLib.ICall)
Chapter 8  Session Object

Methods

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.

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)

VB:  
SetMessageFilter (filter As String, retVal As Long)

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.

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:

`'put filter expression in here
Dim sFilterExpression As String

'call SetMessageFilter
m_session.SetMessageFilter 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:

```plaintext
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:

```plaintext
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:

```plaintext
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 OR 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, or any event 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 (from listing in Appendix A) 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.
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() (see the section “Connecting to CTI OS” in Chapter 4, “Building Your Application”).
- 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. |
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")

Table 9-1 Agent Properties

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>Statistics</td>
<td>ARGUMENTS</td>
<td>An arguments array containing the statistics listed in Table 9-2.</td>
</tr>
</tbody>
</table>
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>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailTimeSession</td>
<td>INT</td>
<td>Total time, in seconds, the agent was in the Available state for any skill group.</td>
</tr>
<tr>
<td>LoggedOnTimeSession</td>
<td>INT</td>
<td>Total time, in seconds, the agent has been logged on.</td>
</tr>
<tr>
<td>NotReadyTimeSession</td>
<td>INT</td>
<td>Total time, in seconds, the agent was in the Not Ready state for all skill groups.</td>
</tr>
<tr>
<td>AgentOutCallsSession</td>
<td>INT</td>
<td>Total number of completed outbound ACD calls made by the agent.</td>
</tr>
<tr>
<td>AgentOutCallsTalkTimeSession</td>
<td>INT</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>
</tbody>
</table>
### Table 9-2  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentOutCallsTimeSession</td>
<td>INT</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>AgentOutCallsHeldSession</td>
<td>INT</td>
<td>The total number of completed outbound ACD calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeSession</td>
<td>INT</td>
<td>Total number of seconds outbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>HandledCallsSession</td>
<td>INT</td>
<td>The number of inbound ACD calls handled by the agent.</td>
</tr>
<tr>
<td>HandledCallsTalkTimeSession</td>
<td>INT</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>HandledCallsAfterCallTimeSession</td>
<td>INT</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.</td>
</tr>
<tr>
<td>HandledCallsTimeSession</td>
<td>INT</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>
</tbody>
</table>
### Table 9-2 Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IncomingCallsHeldSession</td>
<td>INT</td>
<td>The total number of completed inbound ACD calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeSession</td>
<td>INT</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>InternalCallsSession</td>
<td>INT</td>
<td>Number of internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCallsTimeSession</td>
<td>INT</td>
<td>Number of seconds spent on internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCallsRcvdSession</td>
<td>INT</td>
<td>Number of internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeSession</td>
<td>INT</td>
<td>Number of seconds spent on internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCallsHeldSession</td>
<td>INT</td>
<td>The total number of internal calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCallsHeldTimeSession</td>
<td>INT</td>
<td>Total number of seconds completed internal calls were placed on hold.</td>
</tr>
<tr>
<td>AutoOutCalls Session</td>
<td>INT</td>
<td>Total number of AutoOut (predictive) calls completed by the agent.</td>
</tr>
<tr>
<td>AutoOutCallsTalkTimeSession</td>
<td>INT</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>
AutoOutCallsTimeSession | INT | 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.

AutoOutCallsHeldSession | INT | The total number of completed AutoOut (predictive) calls the agent has placed on hold at least once.

AutoOutCallsHeldTimeSession | INT | Total number of seconds AutoOut (predictive) calls were placed on hold.

PreviewCallsSession | INT | Total number of outbound Preview calls completed by the agent.

PreviewCallsTalkTimeSession | INT | 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.
Table 9-2  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviewCallsTimeSession</td>
<td>INT</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>PreviewCallsHeldSession</td>
<td>INT</td>
<td>The total number of completed outbound Preview calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>PreviewCallsHeldTimeSession</td>
<td>INT</td>
<td>Total number of seconds outbound Preview calls were placed on hold.</td>
</tr>
<tr>
<td>Reservation CallsSession</td>
<td>INT</td>
<td>Total number of agent reservation calls completed by the agent.</td>
</tr>
<tr>
<td>ReservationCallsTalkTimeSession</td>
<td>INT</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>INT</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>
</tbody>
</table>
### Table 9-2  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservation Calls Held Session</td>
<td>INT</td>
<td>The total number of completed agent reservation calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>Reservation Calls Held Time</td>
<td>INT</td>
<td>Total number of seconds agent reservation calls were placed on hold.</td>
</tr>
<tr>
<td>BargeInCallsSession</td>
<td>INT</td>
<td>Total number of supervisor call barge-ins completed.</td>
</tr>
<tr>
<td>InterceptCallsSession</td>
<td>INT</td>
<td>Total number of supervisor call intercepts completed.</td>
</tr>
<tr>
<td>MonitorCallsSession</td>
<td>INT</td>
<td>Total number of supervisor call monitors completed.</td>
</tr>
<tr>
<td>WhisperCallsSession</td>
<td>INT</td>
<td>Total number of supervisor whisper calls completed.</td>
</tr>
<tr>
<td>EmergencyCallsSession</td>
<td>INT</td>
<td>Total number of emergency calls.</td>
</tr>
<tr>
<td>AvailTimeToday</td>
<td>INT</td>
<td>Total time, in seconds, the agent was in the Available state for any skill group.</td>
</tr>
<tr>
<td>LoggedOnTimeToday</td>
<td>INT</td>
<td>Total time, in seconds, the agent has been logged on.</td>
</tr>
<tr>
<td>NotReadyTimeToday</td>
<td>INT</td>
<td>Total time, in seconds, the agent was in the Not Ready state for all skill groups.</td>
</tr>
<tr>
<td>AgentOutCallsToday</td>
<td>INT</td>
<td>Total number of completed outbound ACD calls made by agent.</td>
</tr>
</tbody>
</table>
Agent Object

Chapter 9

Table 9-2  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentOutCallsTalkTimeToday</td>
<td>INT</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>AgentOutCallsTimeToday</td>
<td>INT</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>AgentOutCallsHeldToday</td>
<td>INT</td>
<td>The total number of completed outbound ACD calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeToday</td>
<td>INT</td>
<td>Total number of seconds outbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>HandledCallsToday</td>
<td>INT</td>
<td>The number of inbound ACD calls handled by the agent.</td>
</tr>
<tr>
<td>HandledCallsTalkTimeToday</td>
<td>INT</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>HandledCallsAfterCallTimeToday</td>
<td>INT</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HandledCallsTimeToday</td>
<td>INT</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>IncomingCallsHeldToday</td>
<td>INT</td>
<td>The total number of completed inbound ACD calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeToday</td>
<td>INT</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>InternalCallsToday</td>
<td>INT</td>
<td>Number of internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCallsTimeToday</td>
<td>INT</td>
<td>Number of seconds spent on internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCallsRcvdToday</td>
<td>INT</td>
<td>Number of internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeToday</td>
<td>INT</td>
<td>Number of seconds spent on internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCallsHeldToday</td>
<td>INT</td>
<td>The total number of internal calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCallsHeldTimeToday</td>
<td>INT</td>
<td>Total number of seconds completed internal calls were placed on hold.</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>Enables agent statistic messages.</td>
</tr>
<tr>
<td>DisableSkillGroupStatistics</td>
<td>Enables skill group statistic messages.</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>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>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>MakeEmergencyCall</td>
<td>Lets an agent makes an emergency call to the supervisor.</td>
</tr>
<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>
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:
  
  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++:    int DisableAgentStatistics (Arguments& reserved_args)
COM:    HRESULT DisableAgentStatistics ([in]/ IArguments
       reserved_args, [out, retval]/ int * errorcode)
VB:     DisableAgentStatistics (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 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

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.

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.”
EnableAgentStatistics

The EnableAgentStatistics method is sent by an agent to request that real-time
statistics be sent to that agent.

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

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.
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

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.
Chapter 9  Agent Object

Methods

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

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++:   CAgent* GetMonitoredAgent()
COM:    HRESULT GetMonitoredAgent (/*[out, retval]*/IAgent **agent)
VB:     GetMonitoredAgent () As CTIOSCLIENTLib.IAgent
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.

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

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.

GetNumProperties

Chapter 9    Agent Object

Methods

GetProperty


GetPropertyType


GetSkillGroups

The GetSkillGroups method is reserved for future use. Do not use.

GetValue Methods


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
Input Parameters

args

Arguments array that contains the login parameters listed in Table 9-4.

Table 9-4 Login Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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.</td>
</tr>
<tr>
<td>PositionID</td>
<td>STRING, maximum length 12</td>
<td>Required for Alcatel only.</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING, maximum length 16</td>
<td>The agent’s teleset extension. Optional if AgentInstrument is provided.</td>
</tr>
<tr>
<td>AgentPassword (optional)</td>
<td>STRING, maximum length 16</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>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

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 request is unsuccessful, the client will receive an OnControlFailureConf event.

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

Input Parameters

args

Input parameter in the form of an Arguments array that contains the Logout parameters listed in Table 9-5.
errorcode

An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”

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 (optional)</td>
<td>STRING, maximum length 12</td>
<td>The agent’s password.</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>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>
<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 (optional)</td>
<td>INT</td>
<td>The ICM Peripheral ID of the ACD the agent is attached to.</td>
</tr>
</tbody>
</table>

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

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>
</tbody>
</table>
Table 9-6 MakeCall Parameters (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>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>
</tbody>
</table>
Table 9-6  MakeCall Parameters (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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></td>
<td>...</td>
<td>...</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>
</tbody>
</table>
### Table 9-7  CallPlacemntType Values (continued)

<table>
<thead>
<tr>
<th>CallPlacementType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<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 &quot;online&quot; (available to interact with the destination party).</td>
<td>1</td>
</tr>
<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>
</tbody>
</table>
### Table 9-9  CallOption Values (continued)

<table>
<thead>
<tr>
<th>CallOption</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<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 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>
<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>
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:

```csharp
HRESULT MakeEmergencyCall (/*[in, optional]*/  IArguments reserved_args, /* [out, retval]*/ int * errorcode)
```

VB:

```vb
MakeEmergencyCall () As Long
MakeEmergencyCall (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 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. 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 is reserved for future use. Do not use.

**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

**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.”
Chapter 9      Agent Object

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

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

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++: virtual int RequestSupervisorAssist();

int RequestSupervisorAssist (Arguments & reserved_args)

COM: HRESULT RequestSupervisorAssist (/*[in, optional]*/ IArguments reserved_args, /* [out, retval]*/ int * errorcode)

VB: RequestSupervisorAssist () 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.”

Remarks

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.
Chapter 9  Agent Object

Methods

Syntax

\[
\begin{align*}
\text{C++:} & \quad \text{int SendChatMessage (Arguments& args)} \\
\text{COM:} & \quad \text{HRESULT SendChatMessage ([in] IArguments *args,} \\
& \quad \text{[out,retval] int * errorcode)} \\
\text{VB:} & \quad \text{SendChatMessage (args As CTIOSCLIENTLib.IArguments) As Long)
\end{align*}
\]

Input Parameters

args

Input parameter in the form of an Arguments array that contains one or more of the SendChatMessage parameters listed in Table 9-12.

<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”.</td>
</tr>
<tr>
<td>Target</td>
<td>STRING</td>
<td>When the Distribution is set to DistributeToAgent, this field must be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>included with the AgentID of the intended recipient.</td>
</tr>
<tr>
<td>Message</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++:**
```cpp
int SetAgentState(Arguments& args)
```

**COM:**
```cpp
HRESULT SetAgentState (/*[in]*/ IArguments *args,  
/*[out,retval]*/ int * errorcode)
```

**VB:**
```vbnet
SetAgentState (args As CTIOSCLIENTLib.IArguments) As Long
```

Input Parameters

**args**

Input parameter in the form of an Arguments array that contains one or more of the SetAgentState parameters listed in Table 9-13.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</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>
</tbody>
</table>
Table 9-13  SetAgentState Parameters (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentWorkMode</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</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</td>
<td>INT</td>
<td>Reason for logging out. Required for IPCC, optional for all other switches.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The ICM Peripheral ID of the ACD the agent is attached to.</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>
</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++:

```c++
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:

```c
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++:

```cpp
int StartMonitoringAgent(Arguments& args)
```

COM:

```c
HRESULT StartMonitoringAgent (/*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode)
```

VB:

```vb
StartMonitoringAgent (args As CTIOSCLIENTLib.IArguments) As Long
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>args</td>
<td>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”.</td>
</tr>
<tr>
<td>errorcode</td>
<td>An output parameter (return parameter in VB) that contains an error code from Table 3-2 in Chapter 3, “CIL Coding Conventions.”</td>
</tr>
</tbody>
</table>

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.

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++:**

```cpp
int StartMonitoringAgentTeam (Arguments& args)
```

**COM:**

```cpp
HRESULT StartMonitoringAgentTeam (/*[in]*/ IArguments args,
                                 /*[out,retval]*/ int * errorCode)
```

**VB:**

```vb
StartMonitoringAgentTeam (args as CTIOSCLIENTLib.IArguments) As Long
```

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.

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

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.

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

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.

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
```

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.
StopMonitoringAgentTeam

The StopMonitoringAgentTeam method allows the client, which must be a supervisor, to stop monitoring the specified agent team.

Syntax

C++:

```cpp
int StopMonitoringAgentTeam (Arguments& args)
```

COM:

```csharp
HRESULT StopMonitoringAgentTeam (/*[in]*) IArguments args,
/*[out,retval]*)*/ int * errorCode)
```

VB:

```vb
StopMonitoringAgentTeam (args as CTIOSCLIENTLib.IArguments) As Long
```

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.
StopMonitoringAllAgentTeams

The StopMonitoringAllAgentTeams method allows the client, which must be a supervisor, to stop monitoring all of the agent teams.

Syntax

**C++:**
```cpp
int StopMonitoringAllAgentTeams (Arguments& reserved_args)
```

**COM:**
```cpp
HRESULT StopMonitoringAllAgentTeams (/*[in,optional]*/
    IArguments reserved_args, /*[out,retval]*/ int * errorcode)
```

**VB:**
```vbnet
StopMonitoringAllAgentTeams([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 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.
SuperviseCall

The SuperviseCall method allows the client, which must be a supervisor, to perform a supervisory action specified by the args parameter.

Syntax

\[
\begin{align*}
\text{C++:} & \quad \text{int SuperviseCall(Arguments\& args)} \\
\text{COM:} & \quad \text{HRESULT SuperviseCall (/*[in]*/ IArguments \* args,} \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{/*[out,retval]*/ int errorCode)} \\
\text{VB:} & \quad \text{SuperviseCall (args As CTIOSCLIENTLib.IArguments ) As Long}
\end{align*}
\]

Input Parameters

\text{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.

The following applies to the IPCC implementation of this request:

- 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;
```
The same thing in VB would be as follows:

```vbnet
Dim MyECCData As CTIOSARGUMENTSLib.Arguments
Dim MyECCVariable As String
Dim MyECCArrayVariable As Integer

If MyCall.IsValid("ECC") = True Then
    Set MyECCData = MyCall.GetValueArray("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
```
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:

```c++
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();
```

The same thing in VB would be:

```vbnet
Dim MyRequestArgs As New CTIOSARGUMENTSLib.Arguments
Dim MyECCData As CTIOSARGUMENTSLib.Arguments
If MyCall.IsValid("ECC") Then
Set MyECCData = MyCall.GetValueArray("ECC")
Else
Set MyECCData = New CTIOSARGUMENTSLib.Arguments
End If
Set MyECCData = New CTIOSARGUMENTSLib.Arguments
MyECCData.AddItem("user.MyECC", "FirstECCVariable")
MyECCData.AddItem("user.MyArray[2]", 2222)
MyRequestArgs.AddItem("ECC", MyECCData)
MyCall.SetCallData(MyRequestArgs)
```
Properties

Table 10-1 lists the available call object properties.

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>
</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>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>
<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>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCall KeyCallID 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>
</tbody>
</table>
Chapter 10  Call Object

Methods

Table 10-1  Call Object Properties (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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_GROU...</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_GROU...</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>UserToUserInfo</td>
<td>STRING</td>
<td>The ISDN user-to-user information element.</td>
</tr>
</tbody>
</table>

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 only two parties on the call it clears the call.</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>Conference</td>
<td>Either establishes a three party conference call or adds a new party to an existing conference call.</td>
</tr>
<tr>
<td>Deflect</td>
<td>Allows a client to take an alerting call from a known device and move it to another device.</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>
<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>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, GetValueArray,</td>
<td>Retrieves a property from the Agent object based on the property’s name key.</td>
</tr>
<tr>
<td>GetValueInt, GetValueString</td>
<td></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>
</tbody>
</table>
Table 10-2  Call Object Methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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:

  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)
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++:
int Alternate()
int Alternate(Arguments & reserved_args);

COM:
HRESULT Alternate (/*[in,optional]*/ IArguments
*reserved_args,  (/*[out, retval]*/ int * errorcode );

VB:
Alternate([reserved_args As 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.”

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
```

**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

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++:**
```c++
int Clear()
int Clear(Arguments & reserved_args);
```

**COM:**
```c++
HRESULT Clear (/*[in,optional]*/ IArguments *reserved_args,
        (/*[out, retval]*/ int * errorcode )
```

**VB:**
```vb
Clear([reserved_args As 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 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 ClearConnection below. Under certain switches the Clear request will must 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

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++:**

```cpp
int ClearConnection()
int ClearConnection(Arguments & reserved_args);
```

**COM:**

```com
HRESULT ClearConnection (/*[in,optional]*/ IArguments *
reserved_args,     (/*[out, retval]*/ int * errorcode)
```

**VB:**

```vb
ClearConnection([reserved_args As 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 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 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:

- 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.
Chapter 10  Call Object

Methods

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

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

### Deflect

The Deflect method allows a client to take an alerting call connection and move it to another device without answering it.

### Syntax

**C++:**

```cpp
int Deflect(Arguments & args);
```

**COM:**

```cpp
HRESULT Deflect (/*[in]*/ IArguments *args,   (/*[out, retval]*/ int * errorcode )
```

**VB:**

```cpp
Deflect(args As IArguments) As Long
```

### Parameters

**args**

An Arguments array that contains a number with a string value that is the dialed number (string) that is the desired target for the call to be delivered, using the keyword “DialedNumber.”
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 Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”

### Remarks

The Deflect request must be made via a call whose status is either LCS_ALERT or it will fail on certain switches.

The following events will be received if this request is successful:
- OnCallDiverted

The following events will be received by the call making the Alternate request if this request fails:
- OnControlFailureConf event

### 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++:**
```cpp
int GetCallContext(Arguments& args)
```

**COM:**
```cpp
HRESULT GetCallContext(/*[out,retval]*/ IArguments **args)
```

**VB:**
```vbnet
GetCallContext (CTIOSCLIENTLib.IArguments args)
```

#### 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>
<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>
</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>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++:    int GetCallData(Arguments& args)
COM:    HRESULT GetCallData /*[out retval]*/ IArguments ** args)
VB:     GetCallData (CTIOSCLIENTLib.IArguments args)

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 which 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.

GetValue Methods


Hold

The Hold method holds a current call.
Chapter 10      Call Object

Methods

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

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)

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>
</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>CallPlacementType</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</td>
<td>INT</td>
<td>A value specifying additional call processing options identified in Table 10-6.</td>
</tr>
<tr>
<td>AlertRings</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</td>
<td>INT</td>
<td>A value from Table 10-7 specifying additional peripheral-specific call options.</td>
</tr>
<tr>
<td>FacilityType</td>
<td>INT</td>
<td>A value from Table 10-8 indicating the type of facility to be used.</td>
</tr>
<tr>
<td>AnsweringMachine</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</td>
<td>BOOL</td>
<td>This field should be set to TRUE if the call should receive priority handling.</td>
</tr>
<tr>
<td>PostRoute</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</td>
<td>STRING, maximum length 40</td>
<td>The ISDN user-to-user information.</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>CallVariable1 (optional)</td>
<td>STRING, maximum</td>
<td>Call variable data that should be</td>
</tr>
<tr>
<td></td>
<td>length 40</td>
<td>set in the new call in place of the corresponding data in the current</td>
</tr>
<tr>
<td></td>
<td></td>
<td>call.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>data in the current call.</td>
</tr>
<tr>
<td>CallWrapupData (optional)</td>
<td>STRING, maximum</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td></td>
<td>length 40</td>
<td></td>
</tr>
<tr>
<td>FacilityCode (optional)</td>
<td>STRING, maximum</td>
<td>A trunk access code, split extension, or other data needed to access the</td>
</tr>
<tr>
<td></td>
<td>length 40</td>
<td>chosen facility.</td>
</tr>
<tr>
<td>AuthorizationCode (optional)</td>
<td>STRING, maximum</td>
<td>An authorization code needed to access the resources required to initiate</td>
</tr>
<tr>
<td></td>
<td>length 40</td>
<td>the call.</td>
</tr>
<tr>
<td>AccountCode (optional)</td>
<td>STRING, maximum</td>
<td>A cost-accounting or client number used by the peripheral for charge-back</td>
</tr>
<tr>
<td></td>
<td>length 40</td>
<td>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>
</tbody>
</table>
### Table 10-5 CallPlacementType Values (continued)

<table>
<thead>
<tr>
<th>CallPlacementType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<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>
<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>
</tbody>
</table>
### Table 10-7 CallOption Values (continued)

<table>
<thead>
<tr>
<th>CallOption Value</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<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>
<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>
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:

- 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++:
int Reconnect()
int Reconnect(Arguments & reserved_args)

COM:
HRESULT Reconnect (/*[in,optional]*/ IArguments * reserved_args, /*[out, retval]*/ int * errorcode )

VB:
Reconnect([reserved_args As 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.”

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
```

### 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++:
```cpp
int SendDTMFSignal(Arguments& args)
```

COM:
```csharp
HRESULT SendDTMFSignal(/*[in]*/ args *arguments, /*[out, retval]*/ int * errorcode)
```

VB:
```vbnet
SendDTMFSignal (args As CTIOSCLIENTLib.IArguments, errorcode As Long)
```

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.
Chapter 10  Call Object

Methods

Table 10-10 SendDTMFSignal Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTMFString (required)</td>
<td>STRING, maximum</td>
<td>The sequence of tones to be generated.</td>
</tr>
<tr>
<td></td>
<td>length 32</td>
<td></td>
</tr>
<tr>
<td>ToneDuration (optional)</td>
<td>INT</td>
<td>Specifies the duration in milliseconds of DTMF digit tones. Use 0 to take</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the default. May be ignored if the peripheral is unable to alter the DTMF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tone timing.</td>
</tr>
<tr>
<td>PauseDuration (optional)</td>
<td>INT</td>
<td>Specifies the duration in milliseconds of DTMF inter-digit spacing. Use 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to take the default. May be ignored if the peripheral is unable to alter the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DTMF tone timing.</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 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)

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

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)

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>
</tbody>
</table>
### Table 10-11 SingleStepConference Parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>...</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:

- OnAgentStateChange event (Hold)
- OnCallHeld event
- OnAgentStateChange event (Talking)
- OnBeginCall event
• 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++:**
```cpp
int SingleStepTransfer(C++Arguments& args)
```

**COM:**
```cpp
HRESULT SingleStepTransfer (/*[in]*/ IArguments * args, /*[out, retval]*/ int * errorcode)
```

**VB:**
```vbnet
SingleStepTransfer (args As CTIOSCLIENTLib.IArguments, errorcode As Long)
```

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.”
Chapter 10      Call Object

Methods

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 OnSnapshotCallConf in Chapter 6, “Event Interfaces and Events” for more information.

Syntax

C++
int Snapshot()
int Snapshot(Arguments & optional_args)

COM: HRESULT Snapshot /*[in,optional]*/ IArguments *
optional_args, /*[out, retval]*/ int * errorcode

VB: Snapshot({optional_args As IArguments}) As Long

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 OnSnapshotConf event.
The following events will be received if this request is successful:
- OnSnapshotCallConf 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++:    int StartRecord()
        int StartRecord(Arguments & reserved_args);
COM:     HRESULT StartRecord (/*[in,optional]*/  IArguments
                  *reserved_args,  (/*[out, retval]*/ int * errorcode )
VB:     StartRecord([reserved_args As 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 Value

Default CTI OS return values. See Chapter 3, “CIL Coding Conventions.”
Remarks

Calling this method causes the CTIServer 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
```

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++: int Transfer();
int Transfer(Arguments& optional_args)

COM: HRESULT Transfer ( [in, optional]  IArguments *optional_args,
   /*[out, retval]*/ int * errorcode )

VB: Transfer({optional_args As IArguments}) As Long

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 will be 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.

The SkillGroup object can be used in either an AgentMode or a MonitorMode application:

- **In an Agent Mode application.** When an Agent is logged in, the Session object receives updates for each skill group the agent is logged into. Based on these events, the Session creates and updates one SkillGroup object for each of the agent’s skill groups.

- **In a MonitorMode application.** The application creates SkillGroup objects for each of the skills it wishes to monitor.
Table 11-1 lists the available SkillGroup properties.

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 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>

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.

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  SkillGroup Statistics

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentsLoggedOn</td>
<td>INT</td>
<td>Number of agents that are currently logged on to the skill group</td>
</tr>
<tr>
<td>AgentsAvail</td>
<td>INT</td>
<td>Number of agents in the skill group in Available state.</td>
</tr>
<tr>
<td>AgentsNotReady</td>
<td>INT</td>
<td>Number of agents in the skill group in Not Ready state.</td>
</tr>
<tr>
<td>AgentsReady</td>
<td>INT</td>
<td>Number of agents in the skill group in Ready state.</td>
</tr>
<tr>
<td>AgentsTalkingIn</td>
<td>INT</td>
<td>Number of agents in the skill group currently talking on inbound calls.</td>
</tr>
<tr>
<td>AgentsTalkingOut</td>
<td>INT</td>
<td>Number of agents in the skill group currently talking on outbound calls.</td>
</tr>
<tr>
<td>AgentsTalkingOther</td>
<td>INT</td>
<td>Number of agents in the skill group currently talking on internal (not inbound or outbound) calls.</td>
</tr>
<tr>
<td>AgentsWorkNotReady</td>
<td>INT</td>
<td>Number of agents in the skill group in Work Not Ready state.</td>
</tr>
<tr>
<td>AgentsWorkReady</td>
<td>INT</td>
<td>Number of agents in the skill group in Work Ready state.</td>
</tr>
<tr>
<td>AgentsBusyOther</td>
<td>INT</td>
<td>The number of agents in the skill group currently busy with calls assigned to other skill groups.</td>
</tr>
<tr>
<td>AgentsReserved</td>
<td>INT</td>
<td>Number of agents in the skill group in Reserved state.</td>
</tr>
<tr>
<td>AgentsHold</td>
<td>INT</td>
<td>Number of agents in the skill group with calls currently on hold.</td>
</tr>
</tbody>
</table>
### Table 11-2 SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentsTalkingAutoOut</td>
<td>INT</td>
<td>Number of agents in the skill group currently talking on AutoOut (predictive) calls.</td>
</tr>
<tr>
<td>AgentsTalkingPreview</td>
<td>INT</td>
<td>Number of agents in the skill group currently talking on outbound Preview calls.</td>
</tr>
<tr>
<td>AgentsTalkingReservation</td>
<td>INT</td>
<td>Number of agents in the skill group currently talking on agent reservation calls.</td>
</tr>
<tr>
<td>RouterCallsQNow</td>
<td>INT</td>
<td>The number of calls currently queued by the ICM call router to the skill group. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>LongestRouterCallQNow</td>
<td>INT</td>
<td>The queue time in seconds of the ICM call router queued call that has been queued to this skill group the longest. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>CallsQNow</td>
<td>INT</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>INT</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>
</tbody>
</table>
### Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LongestCallQNow</td>
<td>INT</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>
<tr>
<td>AvailTimeTo5</td>
<td>INT</td>
<td>Total seconds agents in the skill group were in the Available state during the last five minutes.</td>
</tr>
<tr>
<td>LoggedOnTimeTo5</td>
<td>INT</td>
<td>Total time, in seconds, agents in the skill group were logged on during the last five minutes.</td>
</tr>
<tr>
<td>NotReadyTimeTo5</td>
<td>INT</td>
<td>Total seconds agents in the skill group were in the Not Ready state during the last five minutes.</td>
</tr>
<tr>
<td>AgentOutCallsTo5</td>
<td>INT</td>
<td>Total number of completed outbound ACD calls made by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>AgentOutCallsTalkTimeTo5</td>
<td>INT</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by agents in the skill group during the last five minutes. 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  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentOutCallsTimeTo5</td>
<td>INT</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by agents in the skill group during the last five minutes. 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>INT</td>
<td>The total number of completed outbound ACD calls agents in the skill group have placed on hold at least once during the last five minutes.</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeTo5</td>
<td>INT</td>
<td>Total number of seconds outbound ACD calls were placed on hold by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>HandledCallsTo5</td>
<td>INT</td>
<td>The number of inbound ACD calls handled by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>HandledCallsTalkTimeTo5</td>
<td>INT</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by agents in the skill group during the last five minutes. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>HandledCallsAfterCallTimeTo5</td>
<td>INT</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the skill group during the last five minutes.</td>
</tr>
</tbody>
</table>
### Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandledCallsTimeTo5</td>
<td>INT</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the skill group during the last five minutes. 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>INT</td>
<td>The total number of completed inbound ACD calls agents in the skill group placed on hold at least once during the last five minutes.</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeTo5</td>
<td>INT</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>InternalCallsRcvdTo5</td>
<td>INT</td>
<td>Number of internal calls received by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeTo5</td>
<td>INT</td>
<td>Number of seconds spent on internal calls received by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>InternalCallsHeldTo5</td>
<td>INT</td>
<td>The total number of internal calls agents in the skill group placed on hold at least once during the last five minutes.</td>
</tr>
<tr>
<td>InternalCallsHeldTimeTo5</td>
<td>INT</td>
<td>Total number of seconds completed internal calls were placed on hold by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AutoOutCallsTo5</td>
<td>INT</td>
<td>Total number of AutoOut (predictive) calls completed by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>AutoOutCallsTalkTimeTo5</td>
<td>INT</td>
<td>Total talk time in seconds for completed AutoOut (predictive) calls handled by agents in the skill group during the last five minutes. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCallsTimeTo5</td>
<td>INT</td>
<td>Total handle time in seconds for completed AutoOut (predictive) calls handled by agents in the skill group during the last five minutes. 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>AutoOutCallsHeldTo5</td>
<td>INT</td>
<td>Total number of completed AutoOut (predictive) calls that agents in the skill group have placed on hold at least once during the last five minutes.</td>
</tr>
<tr>
<td>AutoOutCallsHeldTimeTo5</td>
<td>INT</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group during the last five minutes.</td>
</tr>
</tbody>
</table>
Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviewCallsTo5</td>
<td>INT</td>
<td>Total number of outbound Preview calls completed by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>PreviewCallsTalkTimeTo5</td>
<td>INT</td>
<td>Total talk time in seconds for completed outbound Preview calls handled by agents in the skill group during the last five minutes. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCallsTimeTo5</td>
<td>INT</td>
<td>Total handle time in seconds for completed outbound Preview calls handled by agents in the skill group during the last five minutes. 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>PreviewCallsHeldTo5</td>
<td>INT</td>
<td>Total number of completed outbound Preview calls that agents in the skill group have placed on hold at least once during the last five minutes.</td>
</tr>
<tr>
<td>PreviewCallsHeldTimeTo5</td>
<td>INT</td>
<td>Total number of seconds outbound Preview calls were placed on hold by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>ReservationCallsTo5</td>
<td>INT</td>
<td>Total number of agent reservation calls completed by agents in the skill group during the last five minutes.</td>
</tr>
</tbody>
</table>
### Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationCallsTalkTimeTo5</td>
<td>INT</td>
<td>Total talk time in seconds for completed agent reservation calls handled by agents in the skill group during the last five minutes. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>ReservationCallsTimeTo5</td>
<td>INT</td>
<td>Total handle time in seconds for completed agent reservation calls handled by agents in the skill group during the last five minutes. 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>ReservationCallsHeldTo5</td>
<td>INT</td>
<td>Total number of completed agent reservation calls that agents in the skill group have placed on hold at least once during the last five minutes.</td>
</tr>
<tr>
<td>ReservationCallsHeldTimeTo5</td>
<td>INT</td>
<td>Total number of seconds agent reservation calls were placed on hold by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>BargeInCallsTo5</td>
<td>INT</td>
<td>Total number of supervisor call barge-ins completed in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>InterceptCallsTo5</td>
<td>INT</td>
<td>Total number of supervisor call intercepts completed in the skill group during the last five minutes.</td>
</tr>
</tbody>
</table>
### Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitorCallsTo5</td>
<td>INT</td>
<td>Total number of supervisor call monitors completed in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>WhisperCallsTo5</td>
<td>INT</td>
<td>Total number of supervisor call whispers completed in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>EmergencyCallsTo5</td>
<td>INT</td>
<td>Total number of emergency calls completed in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>CallsQ5</td>
<td>INT</td>
<td>The number of calls queued to the skill group during the last five minutes. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>CallsQTime5</td>
<td>INT</td>
<td>The total queue time, in seconds, of calls queued to the skill group during the last five minutes. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>LongestCallQ5</td>
<td>INT</td>
<td>The longest queue time, in seconds, of all calls queued to the skill group during the last five minutes. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>AvailTimeToHalf</td>
<td>INT</td>
<td>Total seconds agents in the skill group were in the Available state during the last half-hour.</td>
</tr>
<tr>
<td>LoggedOnTimeToHalf</td>
<td>INT</td>
<td>Total time, in seconds, agents in the skill group were logged on during the last half-hour.</td>
</tr>
</tbody>
</table>
### Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotReadyTimeToHalf</td>
<td>INT</td>
<td>Total seconds agents in the skill group were in the Not Ready state during the last half-hour.</td>
</tr>
<tr>
<td>AgentOutCallsToHalf</td>
<td>INT</td>
<td>Total number of completed outbound ACD calls made by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>AgentOutCallsTalkTimeToHalf</td>
<td>INT</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by agents in the skill group during the last half-hour. 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>AgentOutCallsTimeToHalf</td>
<td>INT</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by agents in the skill group during the last half-hour. 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>INT</td>
<td>The total number of completed outbound ACD calls agents in the skill group have placed on hold at least once during the last half-hour.</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeToHalf</td>
<td>INT</td>
<td>Total number of seconds outbound ACD calls were placed on hold by agents in the skill group during the last half-hour.</td>
</tr>
</tbody>
</table>
### Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandledCallsToHalf</td>
<td>INT</td>
<td>The number of inbound ACD calls handled by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>HandledCallsTalkTimeToHalf</td>
<td>INT</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by agents in the skill group during the last half-hour. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>HandledCallsAfterCallTimeToHalf</td>
<td>INT</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>HandledCallsTimeToHalf</td>
<td>INT</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the skill group during the last half-hour. 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>IncomingCallsHeldToHalf</td>
<td>INT</td>
<td>The total number of completed inbound ACD calls agents in the skill group placed on hold at least once during the last half-hour.</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeToHalf</td>
<td>INT</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>InternalCallsRcvdToHalf</td>
<td>INT</td>
<td>Number of internal calls received by agents in the skill group during the last half-hour.</td>
</tr>
</tbody>
</table>
Table 11-2 SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InternalCallsRcvdTimeToHalf</td>
<td>INT</td>
<td>Number of seconds spent on internal calls received by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>InternalCallsHeldToHalf</td>
<td>INT</td>
<td>The total number of internal calls agents in the skill group placed on hold at least once during the last half-hour.</td>
</tr>
<tr>
<td>InternalCallsHeldTimeToHalf</td>
<td>INT</td>
<td>Total number of seconds completed internal calls were placed on hold by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>AutoOutCallsToHalf</td>
<td>INT</td>
<td>Total number of AutoOut (predictive) calls completed by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>AutoOutCallsTalkTimeToHalf</td>
<td>INT</td>
<td>Total talk time in seconds for completed AutoOut (predictive) calls handled by agents in the skill group during the last half-hour.</td>
</tr>
</tbody>
</table>

The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.
### Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoOutCallsTimeToHalf</td>
<td>INT</td>
<td>Total handle time in seconds for completed AutoOut (predictive) calls handled by agents in the skill group during the last half-hour. 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>AutoOutCallsHeldToHalf</td>
<td>INT</td>
<td>Total number of completed AutoOut (predictive) calls that agents in the skill group have placed on hold at least once during the last half-hour.</td>
</tr>
<tr>
<td>AutoOutCallsHeldTimeToHalf</td>
<td>INT</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>PreviewCallsToHalf</td>
<td>INT</td>
<td>Total number of outbound Preview calls completed by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>PreviewCallsTalkTimeToHalf</td>
<td>INT</td>
<td>Total talk time in seconds for completed outbound Preview calls handled by agents in the skill group during the last half-hour. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
### Table 11-2 SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviewCallsTimeToHalf</td>
<td>INT</td>
<td>Total handle time in seconds for completed outbound Preview calls handled by agents in the skill group during the last half-hour. 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>PreviewCallsHeldToHalf</td>
<td>INT</td>
<td>Total number of completed outbound Preview calls that agents in the skill group have placed on hold at least once during the last half-hour.</td>
</tr>
<tr>
<td>PreviewCallsHeldTimeToHalf</td>
<td>INT</td>
<td>Total number of seconds outbound Preview calls were placed on hold by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>ReservationCallsToHalf</td>
<td>INT</td>
<td>Total number of agent reservation calls completed by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>ReservationCallsTalkTimeToHalf</td>
<td>INT</td>
<td>Total talk time in seconds for completed agent reservation calls handled by agents in the skill group during the last half-hour. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
### Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationCallsTimeToHalf</td>
<td>INT</td>
<td>Total handle time in seconds for completed agent reservation calls handled by agents in the skill group during the last half-hour. 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>ReservationCallsHeldToHalf</td>
<td>INT</td>
<td>Total number of completed agent reservation calls that agents in the skill group have placed on hold at least once during the last half-hour.</td>
</tr>
<tr>
<td>ReservationCallsHeldTime</td>
<td>INT</td>
<td>Total number of seconds agent reservation calls were placed on hold by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>ToHalf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BargeInCallsToHalf</td>
<td>INT</td>
<td>Total number of supervisor call barge-ins completed in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>InterceptCallsToHalf</td>
<td>INT</td>
<td>Total number of supervisor call intercepts completed in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>MonitorCallsToHalf</td>
<td>INT</td>
<td>Total number of supervisor call monitors completed in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>WhisperCallsToHalf</td>
<td>INT</td>
<td>Total number of supervisor call whispers completed in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>EmergencyCallsToHalf</td>
<td>INT</td>
<td>Total number of emergency calls completed in the skill group during the last half-hour.</td>
</tr>
</tbody>
</table>
## Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallsQHalf</td>
<td>INT</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>INT</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>INT</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>INT</td>
<td>Total seconds agents in the skill group were in the Available state.</td>
</tr>
<tr>
<td>LoggedOnTimeToday</td>
<td>INT</td>
<td>Total time, in seconds, agents in the skill group were logged on.</td>
</tr>
<tr>
<td>NotReadyTimeToday</td>
<td>INT</td>
<td>Total seconds agents in the skill group were in the Not Ready state.</td>
</tr>
<tr>
<td>AgentOutCallsToday</td>
<td>INT</td>
<td>Total number of completed outbound ACD calls made by agents in the skill group.</td>
</tr>
</tbody>
</table>
Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentOutCallsTalkTimeToday</td>
<td>INT</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>INT</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>INT</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>INT</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>INT</td>
<td>The number of inbound ACD calls handled by agents in the skill group.</td>
</tr>
<tr>
<td>HandledCallsTalkTimeToday</td>
<td>INT</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>
</tbody>
</table>
### Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandledCallsAfter CallTimeToday</td>
<td>INT</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>HandledCallsTimeToday</td>
<td>INT</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>INT</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>INT</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsRcvdToday</td>
<td>INT</td>
<td>Number of internal calls received by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeToday</td>
<td>INT</td>
<td>Number of seconds spent on internal calls received by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsHeldToday</td>
<td>INT</td>
<td>The total number of internal calls agents in the skill group placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCallsHeldTimeToday</td>
<td>INT</td>
<td>Total number of seconds completed internal calls were placed on hold by agents in the skill group.</td>
</tr>
</tbody>
</table>
### Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoOutCallsToday</td>
<td>INT</td>
<td>Total number of AutoOut (predictive) calls completed by agents in the skill group during the current day.</td>
</tr>
<tr>
<td>AutoOutCallsTalkTimeToday</td>
<td>INT</td>
<td>Total talk time in seconds for completed AutoOut (predictive) calls handled by agents in the skill group during the current day. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCallsTimeToday</td>
<td>INT</td>
<td>Total handle time in seconds for completed AutoOut (predictive) calls handled by agents in the skill group during the current day. 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>AutoOutCallsHeldToday</td>
<td>INT</td>
<td>Total number of completed AutoOut (predictive) calls that agents in the skill group have placed on hold at least once during the current day.</td>
</tr>
<tr>
<td>AutoOutCallsHeldTimeToday</td>
<td>INT</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group during the current day.</td>
</tr>
<tr>
<td>PreviewCallsToday</td>
<td>INT</td>
<td>Total number of outbound Preview calls completed by agents in the skill group during the current day.</td>
</tr>
</tbody>
</table>
## Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviewCallsTalkTimeToday</td>
<td>INT</td>
<td>Total talk time in seconds for completed outbound Preview calls handled by agents in the skill group during the current day. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCallsTimeToday</td>
<td>INT</td>
<td>Total handle time in seconds for completed outbound Preview calls handled by agents in the skill group during the current day. 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>PreviewCallsHeldToday</td>
<td>INT</td>
<td>Total number of completed outbound Preview calls that agents in the skill group have placed on hold at least once during the current day.</td>
</tr>
<tr>
<td>PreviewCallsHeldTimeToday</td>
<td>INT</td>
<td>Total number of seconds outbound Preview calls were placed on hold by agents in the skill group during the current day.</td>
</tr>
<tr>
<td>ReservationCallsToday</td>
<td>INT</td>
<td>Total number of agent reservation calls completed by agents in the skill group during the current day.</td>
</tr>
</tbody>
</table>
### Table 11-2  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationCallsTalkTimeToday</td>
<td>INT</td>
<td>Total talk time in seconds for completed agent reservation calls handled by agents in the skill group during the current day. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>ReservationCallsTimeToday</td>
<td>INT</td>
<td>Total handle time in seconds for completed agent reservation calls handled by agents in the skill group during the current day. 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>ReservationCallsHeldToday</td>
<td>INT</td>
<td>Total number of completed agent reservation calls that agents in the skill group have placed on hold at least once during the current day.</td>
</tr>
<tr>
<td>ReservationCallsHeldTimeToday</td>
<td>INT</td>
<td>Total number of seconds agent reservation calls were placed on hold by agents in the skill group during the current day.</td>
</tr>
<tr>
<td>BargeInCallsToday</td>
<td>INT</td>
<td>Total number of supervisor call barge-ins completed in the skill group during the current day.</td>
</tr>
<tr>
<td>InterceptCallsToday</td>
<td>INT</td>
<td>Total number of supervisor call intercepts completed in the skill group during the current day.</td>
</tr>
</tbody>
</table>
Table 11-3  lists the SkillGroup object methods.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitorCallsToday</td>
<td>INT</td>
<td>Total number of supervisor call monitors completed in the skill group during the current day.</td>
</tr>
<tr>
<td>WhisperCallsToday</td>
<td>INT</td>
<td>Total number of supervisor call whispers completed in the skill group during the current day.</td>
</tr>
<tr>
<td>EmergencyCallsToday</td>
<td>INT</td>
<td>Total number of emergency calls completed in the skill group during the current day.</td>
</tr>
<tr>
<td>CallsQToday</td>
<td>INT</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>INT</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>INT</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>
</tbody>
</table>
Chapter 11  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</td>
<td>See Chapter 7, “CtiOs Object.”</td>
</tr>
<tr>
<td>GetValueList</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>

### DisableSkillGroupStatistics

The DisableSkillGroupStatistics method requests that sending real-time statistics to the skillgroup object be stopped.

### Syntax

**C++:**

```cpp
int DisableSkillGroupStatistics(Arguments & args)
```

**COM:**

```cpp
HRESULT DisableSkillGroupStatistics (IArguments * args, int * errorCode)
```

**VB:**

```vb
DisableSkillGroupStatistics (args As CTIOSCLIENTLib.IArguments, 
errorCode As Long)
```

### Parameters

- **args**
  
  Not currently used, reserved for future use.
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.

**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++:
```
int EnableSkillGroupStatistics(Arguments & args)
```

COM:
```
HRESULT EnableSkillgroupStatistics (IArguments * args, int * errorCode)
```

VB:
```
EnableSkillgroupStatistics (args As CTIOSCLIENTLib.IArguments, errorCode As Long)
```

**Parameters**

args

Not currently used, reserved for future use.
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.

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 CCoIoObject (C++ only). For instance, it can hold reference to a CAgent, CCall, CSkillGroup, CCoIoSession, or CWaitObject.

- **CCtiosException.** The CCtiosException class is used by CTIOS to provide detailed information when an exception occurs (C++ only). When an exception is caught as CCtiosException, the programmer can query it for details such as error codes and error messages.
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.

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>
<tr>
<td>GetValueInt</td>
<td>Returns the value stored in the argument.</td>
</tr>
<tr>
<td>GetValueUInt</td>
<td></td>
</tr>
<tr>
<td>GetValueUShort</td>
<td></td>
</tr>
<tr>
<td>GetValueString</td>
<td></td>
</tr>
<tr>
<td>GetValueBool</td>
<td></td>
</tr>
<tr>
<td>SetValue</td>
<td>Sets the data in the Arg object.</td>
</tr>
</tbody>
</table>

In many scenarios, programmers will stick to Arguments (see the preceding section), which wraps many Arg methods and encapsulates a collection of Arg objects.
Chapter 12 Helper Classes

Arg Class

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.

Syntax

C++: void AddRef()
COM: HRESULT AddRef()
VB: Not used in VB

Parameters

None.

Return Values

COM: Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”
Others: None.

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 CTIPLATFORMLib.IArg
Chapter 12      Helper Classes

Arg Class

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; otherwise, it will throw a CCtiosException with iCode set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED.

CreateInstance

The CreateInstance method creates an object of type Arg class. It is important to release the object when it is no longer in use in the program.

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

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.

**Syntax**

**C++:**
```cpp
string DumpArg()
```

**COM:**
```vbnet
HRESULT DumpArg([out,retval] BSTR* arg_string);
```

**VB:**
```vbnet
DumpArg() as String
```

**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.
Chapter 12  Helper Classes

Arg Class

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++:
```cpp
enumArgTypes GetType()
```

COM:
```cpp
HRESULT GetType(/*[out, retval]*/ int* type);
```

VB:
```vbnet
GetType () as Integer
```

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 12-2  enumArgTypes

<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 * pValue);
bool GetValueBool( bool * value);
bool GetValueString(string* value);

COM:  
HRESULT GetValue(/*[out, retval]*/ VARIANT* value);

VB:  
GetValue() as Variant

Parameters

Value

Output 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>
</tbody>
</table>
Chapter 12 Helper Classes

Arg Class

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++: void Release()
COM: HRESULT Release()
VB: Not used in VB

Parameters

None.

Table 12-3 Variant types supported by GetValue (COM)

<table>
<thead>
<tr>
<th>VT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT_BOOL</td>
<td>Bool</td>
</tr>
<tr>
<td>VT_BSTR</td>
<td>string, const string and char *</td>
</tr>
</tbody>
</table>

Cisco ICM Software CTI OS Developer's Guide
Chapter 12 Helper Classes

Arg Class

Return Values

**COM:** Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

**Others:** None.

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:**

```c
HRESULT SetValue(/*[in]*/ VARIANT * pVariant, /*[out,retval]*/
VARIANT_BOOL * errorcode );
```

**VB:**

```vb
SetValue (key as String, value as Variant) as Boolean
```

**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>
</tbody>
</table>
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 12-4  Supported Variant Types (continued)

<table>
<thead>
<tr>
<th>Variant Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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.
Table 12-5 lists the Arguments class methods.

### 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 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></td>
<td>(also GetElementInt, GetElementUInt, GetElementUShort, GetElementBool,</td>
</tr>
<tr>
<td></td>
<td>GetElementString, GetElementArg, GetElementKey</td>
</tr>
<tr>
<td></td>
<td>GetElementArgType)</td>
</tr>
<tr>
<td>GetValue</td>
<td>Returns the value stored under a specified key.</td>
</tr>
<tr>
<td></td>
<td>(also GetValueInt, GetValueUShort, GetValueShort, GetValueBool, GetValue</td>
</tr>
<tr>
<td></td>
<td>String, GetValueArray, GetValueArg)</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)
Chapter 12 Helper Classes

Arguments Class

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++:

```c++
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 );
```

```c++
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 );
```

```c++
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 );
```
### Arguments Class

```c++
bool AddItem( enum_Keywords key, const Args& value );
bool AddItem( enum_Keywords key, Arguments& value );
bool AddItem( enum_Keywords key, const Arguments& value);
```

**COM:**

```c++
HRESULT AddItem(/*[in]*/ VARIANT *key, /*[in]*/ VARIANT *value,
    /*[out,retval]*/ VARIANT_BOOL success) As Boolean;
```

**VB:**

```c++
AddItem( Key as Variant, Value as Variant)
```

#### 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 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.

#### Syntax

**C++:**

```c++
void AddRef()
```

**COM:**

```c++
HRESULT AddRef()
```

**VB:**

Not used in VB
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()

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:
Arguments & argsCopy = (Arguments&) argsOrig.Clone();

To cast in VB, do the following:
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

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. It is important to release the object when it is no longer in use in the program.

Syntax

C++: static Arguments & CreateInstance()
     static bool CreateInstance(Arguments ** args)
COM: Not exposed, called by CoCreateInstance.
VB: Not exposed, called by New.
Chapter 12      Helper Classes

Arguments Class

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

C++:  string DumpArgs()
COM:  HRESULT DumpArgs([out,retval] BSTR* arg_string);
VB:   DumpArgs() as String

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;...”.
Chapter 12 Helper Classes

Arguments Class

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. Index is 1-based. 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++:

```cpp
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:

```cpp
HRESULT GetElementKey(/*[in]*/ int index, /*[out]*/ BSTR * key);
HRESULT GetElement(/*[in]*/ int index, /*[out]*/ VARIANT * value);
```
Chapter 12  Helper Classes

Arguments Class

VB:
GetElement (Integer index, Variant value)
GetElement (Integer index, String key)

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++:
Args Get Value( enum_Keywords eKey );
bool Get Value( enum_Keywords key, Arg ** value );
Args Get Value( std::string& key );
bool Get Value( std::string& key, Arg ** value );
Args Get ValueArg( 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 GetValueUInt( 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 GetValueUInt( 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);
bool GetValueString( std::string& key, std::string * value);
int GetValueInt( char * key); /*throws exception*/
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);
## Arguments Class

### Arguments Class

**bool GetValueArg( char * key, Arg ** value);**

**COM:**

```c
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
```

### 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.
Chapter 12      Helper Classes

Arguments Class

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 );

COM:
HRESULT IsValid( /*[in]*/ VARIANT* key, /*[out, retval]*/ VARIANT_BOOL* bIsvalid);

VB:
IsValid (key as string) as Boolean

Parameters

key/arg

Either the key of the desired Arguments member or the entire member.

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).
Arguments Class

Syntax

C++:    int NumElements();
COM:    HRESULT NumElements(/*[out, retval]*/ int * num_elements);
VB: NumElements as Integer

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++:    void Release()
COM:    HRESULT Release()
VB: Not used in VB

Parameters

None.
Return Values

COM: Default HRESULT return values. See Chapter 3, “CIL Coding Conventions.”

Others: None.

RemoveItem

The RemoveItem method removes a value and its associated key from an arguments array. Subsequent attempts to access a value that was removed using RemoveItem 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

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.
Arguments Class

Chapter 12  Helper Classes

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]*/ success;

VB:  SetElement (index as Integer, value as Variant) as Boolean

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.
Arguments Class

Chapter 12  Helper Classes

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, 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, 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, Arguments& value );
bool SetValue( enum_Keywords key, const Arguments& value );
bool SetValue( enum_Keywords key, char * value );
```

**COM:**

```cpp
HRESULT SetValue( /*[in]*/ VARIANT* key, /*[in]*/ VARIANT* value, /*[out, retval]*/ VARIANT_BOOL* success );
```
Chapter 12 Helper Classes

CILRefArg Class (C++ only)

VB: SetValue (key as String, value as Variant) as Boolean

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++ only)

Note The CILRefArg class is only available for the CIL in C++ using the static libraries.

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 ARG_NOTSET</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_NOTSET. It is defined and enumerated type to represent CTI OS Objects references. It has the same syntax as the Arg class GetType method.

**GetValue**

The GetValue method returns the reference to CTI OS object encapsulated in the CILRefArg.

**Syntax**

```c++
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.
Chapter 12  Helper Classes

SetValu

Sets the reference to the CTI OS Object in the CILRefArg.

Syntax

    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

Note: The CCtiOsException class is only available for C++.

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>
</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

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

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

const char* GetString();

Parameters

None.

Return Values

Returns a text string containing the description of the exception.

What

The What method returns a text string containing the description of the exception, the code of an error, and the status.

Syntax

const char* What();

Parameters

None.

Return Values

Returns a text string containing the description of the exception, the code of an error, and the status.
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. It will be extended to the COM CIL (for Visual Basic and C++) in the future.

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++:

```c++
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.

## 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.
Sample C++ Application

The sample application in this appendix demonstrates the use of the CIL (Client Interface Library) in a C++ application. The following picture of the workspace in Visual Studio gives a quick overview of the files that comprise the sample application.
The following is a view of the main screen of the CIL C++ application:

A code listing of each of the files follows.
Appendix B  Sample C++ Application

-------- Start "EventSink.h" --------
// EventSink.h: interface for the CEventSink class.
////////////////////////////////////////////////////////////////////////
#if !defined(AFX_EVENTSINK_H__9B7BC0F8_A2EF_11D5_9F66_0010A4E22958__INCLUDED_
#define AFX_EVENTSINK_H__9B7BC0F8_A2EF_11D5_9F66_0010A4E22958__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "cil.h"

class CExercise2Dlg;

class CEventSink : public AllInOneEventsAdapter
{
public:
    CEventSink(CCtiOsSession *pSession, CAgent *pAgent, CExercise2Dlg *pDialog);
    virtual ~CEventSink();

    // Session Events
    void OnConnection(Arguments & rArguments);
    void OnConnectionFailure(Arguments & rArguments);
    void OnConnectionClosed(Arguments & rArguments);
    void OnSetAgentModeEvent(Arguments & rArguments);

    // Agent Events
    void OnAgentStateChange(Arguments & rArguments);
    void OnQueryAgentStateConf(Arguments & rArguments);

    // Call Events
    void OnCallBegin(Arguments & rArguments);
    void OnCallDelivered(Arguments & rArguments);
    void OnCallEstablished(Arguments & rArguments);
    void OnCallEnd(Arguments & rArguments);
    void OnCallConferenced(Arguments & rArguments);
    void OnCallTransferred(Arguments & rArguments);

    // Misc Events
    void OnButtonEnablementChange(Arguments & rArguments);

public:
    CAgent *m_pCtiAgent;  // Agent object ptr
    CCtiOsSession*m_pCtiSession; // Session object ptr
CEventSink::CEventSink(CCtiOsSession *pSession, CAgent *pAgent, CExercise2Dlg *pDialog)
{
    m_pCtiAgent = pAgent;
    m_pCtiSession = pSession;
    m_pDialog = pDialog;
}
CEventSink::~CEventSink()
{
}

// Session Events

// Handle the OnConnection Event - Connect to CTIOS Server succeeded
void CEventSink::OnConnection(Arguments & rArguments)
{
    m_pDialog->LogEvent("OnConnection", rArguments);
m_pDialog->Log("You can Login now!");
m_pDialog->m_edtConnection.SetWindowText("Connected");
m_pDialog->SetSessionConnected(TRUE);
m_pDialog->m_btnLogin.EnableWindow(TRUE);
}

// Handle the OnConnectionFailure Event - Connect to CTIOS Server failed
// void CEEventSink::OnConnectionFailure(Arguments & rArguments)
// {
//   m_pDialog->LogEvent("OnConnectionFailure", rArguments);
//   m_pDialog->m_btnLogin.EnableWindow(FALSE);
// }

// Handle the OnConnectionClosed Event - Connection to CTIOS Server has been closed
// void CEEventSink::OnConnectionClosed(Arguments & rArguments)
// {
//   m_pDialog->LogEvent("OnConnectionClosed", rArguments);
//   m_pDialog->m_edtConnection.SetWindowText("Offline");
//   m_pDialog->SetSessionConnected(FALSE);
//   m_pDialog->m_btnLogin.EnableWindow(FALSE);
// }

// Handle the OnSetAgentModeEvent - Result of SetAgent request
// Applications must wait for the OnSetAgentModeEvent event before attempting an agent login.
// void CEEventSink::OnSetAgentModeEvent(Arguments & rArguments)
// {
//   CString agentID;
//   CString password;
//   CString instrument;
//   CString peripheralID;
//   m_pDialog->LogEvent("OnSetAgentModeEvent", rArguments);
//   m_pDialog->m_edtConnection.SetWindowText("Online");
//   m_pDialog->SetSessionConnected(TRUE);
//   m_pDialog->m_btnLogin.EnableWindow(TRUE);
// }
// get agent info from dialog
m_pDialog->m_edtAgentID.GetWindowText(agentID);
m_pDialog->m_edtPassword.GetWindowText(password);
m_pDialog->m_edtInstrument.GetWindowText(instrument);
m_pDialog->m_edtPeripheralID.GetWindowText(peripheralID);

// add login request arguments
Arguments &arLoginReq = Arguments::CreateInstance();

arLoginReq.AddItem(CTIOS_AGENTID, agentID.GetBuffer(0));
arLoginReq.AddItem(CTIOS_AGENTPASSWORD, password.GetBuffer(0));
arLoginReq.AddItem(CTIOS_AGENTINSTRUMENT, instrument.GetBuffer(0));
arLoginReq.AddItem(CTIOS_PERIPHERALID, peripheralID.GetBuffer(0));

m_pDialog->AddLogMessage("Requesting Agent Login");
m_pCtiAgent->Login(arLoginReq);
arLoginReq.Release();
}

// Agent Events

// Handle the OnAgentStateChange Event - new agent state from CTIOS Server
void CEventSink::OnAgentStateChange(Arguments & rArguments)
{
    m_pDialog->LogEvent("OnAgentStateChange", rArguments);
    // Retrieve new agent state from argument list and display it on the dialog
    m_pDialog->SetAgentState((enumCTIOS_AgentState) rArguments.GetValueInt("AgentState"));
}

// Handle the OnQueryAgentStateChangeConf Event - received as a result of an agent Login
// This will provide the agent's current state
void CEventSink::OnQueryAgentStateConf(Arguments & rArguments)
{
    m_pDialog->LogEvent("OnQueryAgentStateConf", rArguments);
// Display agent state value from Agent object (Could also get state from rArguments)
    m_pDialog->SetAgentState(m_pCtiAgent->GetAgentState());

// Call Events

// Handle the OnCallBegin Event - New call has started
// void CEventSink::OnCallBegin(Arguments & rArguments)
{
    m_pDialog->LogEvent("OnBeginCall", rArguments);

    string tmp = ""

    // Check if CV1 is present in the message
    if (rArguments.IsValid(CTIOS_CALLVARIABLE1) == TRUE)
    {
        // Get CallID from message
        tmp = rArguments.GetValueString(CTIOS_CALLVARIABLE1);
    }

    // Display Call Variable1 on dialog
    m_pDialog->m_edtCallVariable1.SetWindowText(tmp.c_str());

    // Check for UniqueObjectID presense in message
    if (rArguments.IsValid(CTIOS_UNIQUEOBJECTID) == TRUE)
    {
        // Get CallID from message
        tmp = rArguments.GetValueString(CTIOS_UNIQUEOBJECTID);

        // Display CallID on dialog
        m_pDialog->m_edtCallID.SetWindowText(tmp.c_str());
    }
}

// Handle the OnCallDelivered Event - Call is now delivered/alerting at the agent phone
// void CEventSink::OnCallDelivered(Arguments & rArguments)
{
    m_pDialog->LogEvent("OnCallDelivered", rArguments);
// get mode setting from dialog
CString mode;
m_pDialog->m_edtMode.GetWindowText(mode);

// If mode is B or C then automatically answer the call
if (strcmp(mode, "B") == 0 || strcmp(mode, "C") == 0)
{
    // Get CallID from message
    string uoid = rArguments.GetValueString(CTIOS_UNIQUEOBJECTID);

    string key = "UniqueObjectID=" + uoid;

    m_pDialog->AddLogMessage((char *) key.c_str());

    // Get Call ptr from Session object
    CILRefArg & rCArg = (CILRefArg &) m_pCtiSession->GetValue(key);
    CCall *pCall = (CCall *) rCArg.GetValue();

    if (pCall)
    {
        pCall->Answer();
        pCall->Release();
    }

    rCArg.Release();
}

// Handle the OnCallEstablished Event - call has been answered
// void CEventSink::OnCallEstablished(Arguments & rArguments)
{
    m_pDialog->LogEvent("OnCallEstablished", rArguments);

    // get mode from dialog
    CString mode;
    m_pDialog->m_edtMode.GetWindowText(mode);

    // if mode is B then increment call variable 1 and pass as part of
    // automatic transfer to agent C
    // if (strcmp(mode, "B") == 0)
    {
        // Increment Call Variable value
CString cv1;
m_pDialog->m_edtCallVariable1.GetWindowText(cv1);

int cv_value = atoi(cv1.GetBuffer(0));
cv_value++;

cv1.Format("%d", cv_value);

// Get phone number to transfer to
CString dn;
m_pDialog->m_edtPhoneNumber.GetWindowText(dn);

if (dn.IsEmpty() == TRUE)
{
    m_pDialog->AddLogMessage("Must fill in phone number field for automatic transfer");
    return;
}

// get pointer to call object
CILRefArg & rCArg = (CILRefArg &) m_pCtiSession->GetValue("ActiveCall");

CCall *pCall = (CCall *)rCArg.GetValue();

// Create arg array for transfer request
Arguments &rArgTransfer = Arguments::CreateInstance();

rArgTransfer.AddItem(CTIOS_DIALEDNUMBER, dn.GetBuffer(0));
rArgTransfer.AddItem(CTIOS_CALLVARIABLE1, cv1.GetBuffer(0));

pCall->SingleStepTransfer(rArgTransfer);

// free objects
rArgTransfer.Release();
pCall->Release();
rCArg.Release();
}

// if mode is C then simply release the call as we are finished
else if (strcmp(mode, "C") == 0)
{
    // get pointer to call object
    CILRefArg & rCArg = (CILRefArg &) m_pCtiSession->GetValue("ActiveCall");

    CCall *pCall = (CCall *)rCArg.GetValue();
    }
// Clear call - no arguments
pCall->Clear();

// free objects
pCall->Release();
rCArg.Release();
}

// Handle the OnCallEnd Event - Call is officially ended
// void CEventSink::OnCallEnd(Arguments & rArguments)
{
    m_pDialog->LogEvent("OnCallEnd", rArguments);
    m_pDialog->m_edtCallID.SetWindowText("no call");
}

// Handle the OnCallConferenced Event - call has been conferenced
// void CEventSink::OnCallConferenced(Arguments & rArguments)
{
    m_pDialog->LogEvent("OnCallConferenced", rArguments);
    // Get CallID from message
    string tmp = rArguments.GetValueString(CTIOS_UNIQUEOBJECTID);
    // Display Call Variable1 on dialog
    m_pDialog->m_edtCallID.SetWindowText(tmp.c_str());
}

// Handle the OnCallTransferred Event - call has been transferred
// void CEventSink::OnCallTransferred(Arguments & rArguments)
{
    m_pDialog->LogEvent("OnCallTransfered", rArguments);
    // Get CallID from message
    string tmp = rArguments.GetValueString(CTIOS_UNIQUEOBJECTID);
    // Display Call Variable1 on dialog
    m_pDialog->m_edtCallID.SetWindowText(tmp.c_str());
}
/ Other Events

// Handle the OnButtonEnablementChange Event
// CTIOS Server sends bitmask with bits set indicating which buttons should be enabled.
// void CEventSink::OnButtonEnablementChange(Arguments & rArguments)
//
// Get mask from message
long eMask = rArguments.GetValueUInt(CTIOS_ENABLEMENTMASK);

if (eMask & ENABLE_ANSWER)
{
    m_pDialog->m_btnAnswer.EnableWindow(TRUE);
} else
{
    m_pDialog->m_btnAnswer.EnableWindow(FALSE);
}

if (eMask & ENABLE_CONFERENCE_COMPLETE)
{
    m_pDialog->m_btnConference.EnableWindow(TRUE);
} else
{
    m_pDialog->m_btnConference.EnableWindow(FALSE);
}

if (eMask & ENABLE_CONFERENCE_INIT)
{
    m_pDialog->m_btnCCConference.EnableWindow(TRUE);
} else
{
    m_pDialog->m_btnCCConference.EnableWindow(FALSE);
}

if (eMask & ENABLE_HOLD)
{
    m_pDialog->m_btnHold.EnableWindow(TRUE);
}
else
{
    m_pDialog->m_btnHold.EnableWindow(FALSE);
}

if (eMask & ENABLE_LOGIN)
{
    m_pDialog->m_btnLogin.EnableWindow(TRUE);
} else
{
    m_pDialog->m_btnLogin.EnableWindow(FALSE);
}

if (eMask & ENABLE_LOGOUT)
{
    m_pDialog->m_btnLogout.EnableWindow(TRUE);
} else
{
    m_pDialog->m_btnLogout.EnableWindow(FALSE);
}

if (eMask & ENABLE_MAKECALL)
{
    m_pDialog->m_btnMakeCall.EnableWindow(TRUE);
} else
{
    m_pDialog->m_btnMakeCall.EnableWindow(FALSE);
}

if (eMask & ENABLE_NOTREADY)
{
    m_pDialog->m_btnNotReady.EnableWindow(TRUE);
} else
{
    m_pDialog->m_btnNotReady.EnableWindow(FALSE);
}

if (eMask & ENABLE_READY)
{
    m_pDialog->m_btnReady.EnableWindow(TRUE);
} else
{
    m_pDialog->m_btnReady.EnableWindow(FALSE);
if (eMask & ENABLE_RELEASE) {
    m_pDialog->m_btnRelease.EnableWindow(TRUE);
} else {
    m_pDialog->m_btnRelease.EnableWindow(FALSE);
}

if (eMask & ENABLE_RETRIEVE) {
    m_pDialog->m_btnRetrieve.EnableWindow(TRUE);
} else {
    m_pDialog->m_btnRetrieve.EnableWindow(FALSE);
}

if (eMask & ENABLE_SINGLE_STEP_TRANSFER) {
    m_pDialog->m_btnSSTransfer.EnableWindow(TRUE);
} else {
    m_pDialog->m_btnSSTransfer.EnableWindow(FALSE);
}

if (eMask & ENABLE_SINGLE_STEP_CONFERENCE) {
    m_pDialog->m_btnSSConference.EnableWindow(TRUE);
} else {
    m_pDialog->m_btnSSConference.EnableWindow(FALSE);
}

if (eMask & ENABLE_TRANSFER_COMPLETE) {
    m_pDialog->m_btnTransfer.EnableWindow(TRUE);
} else {
    m_pDialog->m_btnTransfer.EnableWindow(FALSE);
}

if (eMask & ENABLE_TRANSFER_INIT)
m_pDialog->m_btnCCTransfer.EnableWindow(TRUE);
}
else
{
    m_pDialog->m_btnCCTransfer.EnableWindow(FALSE);
}

} // End "EventSink.cpp"

} // End "EventSink.h"

#include "cil.h"

class CExercise2Dlg;

class CEventSink : public AllInOneEventsAdapter
{
public:
    CExerciseSink(CCtiOsSession *pSession, CAgent *pAgent, CExercise2Dlg *pDialog);
    virtual ~CEventSink();

    // Session Events
    void OnConnection(Arguments & rArguments);
    void OnConnectionFailure(Arguments & rArguments);
    void OnConnectionClosed(Arguments & rArguments);
    void OnSetAgentModeEvent(Arguments & rArguments);

    // Agent Events
    void OnAgentStateChange(Arguments & rArguments);
    void OnQueryAgentStateConf(Arguments & rArguments);

    // Call Events
    void OnCallBegin(Arguments & rArguments);
    void OnCallDelivered(Arguments & rArguments);
    void OnCallEstablished(Arguments & rArguments);
    void OnCallEnd(Arguments & rArguments);
Appendix B  Sample C++ Application

void OnCallConferenced(Arguments & rArguments);
void OnCallTransferred(Arguments & rArguments);

// Misc Events
void OnButtonEnablementChange(Arguments & rArguments);

public:
CAgent *m_pCtiAgent; // Agent object ptr
CCTiOsSession*m_pCtiSession; // Session object ptr
CExercise2Dlg*m_pDialog; // dialog ptr
};
#endif // !defined(AFX_EVENTSINK_H__9B78C0F8_A2EF_11D5_9F66_0010A4E22958__INCLUDED_)

-------------- Start "exercise2.h" --------------

#include "stdafx.h"
#include "exercise2.h"
#include "exercise2Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

BEGIN_MESSAGE_MAP(CExercise2App, CWinApp)
//{{AFX_MSG_MAP(CExercise2App)
// NOTE - the ClassWizard will add and remove mapping macros here.
//    DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG
ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

BEGIN_MESSAGE_MAP(CExercise2App, CWinApp)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

CEXercise2App::CEXercise2App()
{
    // TODO: add construction code here,

    CEXercise2App::CEXercise2App()
    {
        // TODO: add construction code here,

    }

    CEXercise2App::CEXercise2App()
    {
        // TODO: add construction code here,

    }

    CEXercise2App::CEXercise2App()
    {
        // TODO: add construction code here,

    }

    CEXercise2App::CEXercise2App()
    {
        // TODO: add construction code here,

    }

    CEXercise2App::CEXercise2App()
    {
        // TODO: add construction code here,

    }

    CEXercise2App::CEXercise2App()
    {
        // TODO: add construction code here,

    }

    CEXercise2App::CEXercise2App()
    {
        // TODO: add construction code here,

    }

    CEXercise2App::CEXercise2App()
    {
        // TODO: add construction code here,

    }

    CEXercise2App::CEXercise2App()
    {
        // TODO: add construction code here,

    }

    CEXercise2App::CEXercise2App()
    {
        // TODO: add construction code here,

    }

    CEXercise2App::CEXercise2App()
    {
        // TODO: add construction code here,

    }

    CEXercise2App::CEXercise2App()
    {
        // TODO: add construction code here,
// Place all significant initialization in InitInstance
}

////////////////////////////////////////////////////////////////////////////
// The one and only CExercise2App object
CExercise2App theApp;

////////////////////////////////////////////////////////////////////////////
// CExercise2App initialization
BOOL CExercise2App::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.
    #ifdef _AFXDLL
        Enable3dControls(); // Call this when using MFC in a shared DLL
    #else
        Enable3dControlsStatic(); // Call this when linking to MFC statically
    #endif

    CExercise2Dlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
        // dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        // dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    // application, rather than start the application's message pump.
    return FALSE;
}

////////////////////////////////////////////////////////////////////////
// End "exercise2.cpp" -----------------------------------------------------

////////////////////////////////////////////////////////////////////////
// Start "exercise2Dlg.h" ---------------------------------------------------
Appendix B

Sample C++ Application

// exercise2Dlg.h : header file
//
#if !defined(AFX_EXERCISE2DLG_H__A89B260A_2BBE_49DC_86CD_132BD0D00CFE__INCLUDED_)
#define AFX_EXERCISE2DLG_H__A89B260A_2BBE_49DC_86CD_132BD0D00CFE__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include "cil.h"
#include "EventSink.h"
/////////////////////////////////////////////////////////////////////////////
// CExercise2Dlg dialog
class CExercise2Dlg : public CDialog
{
// Construction
public:
CExercise2Dlg(CWnd* pParent = NULL);// standard constructor
void AddLogMessage(char *pMessage) { Log(pMessage); }
void LogEvent(const char *msg, Arguments & rArguments);
void Log(const char *newText);
void SetSessionConnected(bool bConnected) { m_bSessionConnected = bConnected; }
void SetAgentState(enumCTIOS_AgentState newstate) {
m_edtAgentState.SetWindowText(AgentStateToString(newstate)); }
char *AgentStateToString(enumCTIOS_AgentState eState);
// Dialog Data
//{{AFX_DATA(CExercise2Dlg)
enum { IDD = IDD_EXERCISE2_DIALOG };
CButton m_btnCCConference;
CButton m_btnCCTransfer;
CButton m_btnSSConference;
CButton m_btnSSTransfer;
CButton m_btnTransfer;
CButton m_btnRetrieve;
CButton m_btnRelease;
CButton m_btnReady;
CButton m_btnNotReady;
CButton m_btnMakeCall;
CButton m_btnLogout;
CButton m_btnLogin;
CButton m_btnHold;

Cisco ICM Software CTI OS Developer’s Guide
OL-1392-03

B-17


CBUTTON m_btnDisconnect;
CBUTTON m_btnConnect;
CBUTTON m_btnConference;
CBUTTON m_btnAnswer;
CEdit m_edtLogWindow;
CEdit m_edtCallID;
CEdit m_edtConnection;
CEdit m_edtAgentState;
CEdit m_edtAgentID;
CEdit m_edtCalVariable1;
CEdit m_edtInstrument;
CEdit m_edtPassword;
CEdit m_edtPeripheralID;
CEdit m_edtPhoneNumber;
CEdit m_edtPortB;
CEdit m_edtPortA;
CEdit m_edtHostB;
CEdit m_edtHostA;
CEdit m_edtMode;
//}}AFX_DATA

//}}AFX_VIRTUAL

protected:
CEventSink *m_pEventSink; // Event Sink ptr
CTtIoSession m_ctiSession; // Session object
CAgent m_ctiAgent; // Agent object
bool m_bSessionConnected; // Connection status indicator
HICON m_hIcon;

//}}AFX_MSG

virtual BOOL OnInitDialog();
afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
afx_msg void OnConnect();
afx_msg void OnDisconnect();
afx_msg void OnLogin();
afx_msg void OnLogout();
afx_msg void OnReady();
```cpp
afx_msg void OnNotready();
afx_msg void OnMakecall();
afx_msg void OnAnswer();
afx_msg void OnRelease();
afx_msg void OnRetrieve();
afx_msg void OnTransfer();
afx_msg void OnHold();
afx_msg void OnConference();
afx_msg void OnClearlog();
afx_msg void OnClose();
afx_msg void OnCCConference();
afx_msg void OnCCTransfer();
afx_msg void OnGetContext();
afx_msg void OnGetData();
afx_msg void OnSetData();
afx_msg void OnSSConference();
afx_msg void OnSSTransfer();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
```

```cpp
//Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
#endif // !defined(AFX_EXERCISE2DLG_H__A89B260A_2BBE_49DC_86CD_132BD0D00CFE__INCLUDED_)
```

```cpp
💳 clientId method
```

```cpp
//{{AFX_INSERT_LOCATION}}
//Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
#endif // !defined(AFX_EXERCISE2DLG_H__A89B260A_2BBE_49DC_86CD_132BD0D00CFE__INCLUDED_)
```
class CAboutDlg : public CDialog  
{
public:
    CAboutDlg();

    // Dialog Data
    ///AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    ///AFX_DATA

    // ClassWizard generated virtual function overrides
    ///AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    ///AFX_VIRTUAL

    // Implementation
    protected:
    ///AFX_MSG(CAboutDlg)
    DECLARE_MESSAGE_MAP()
    ///AFX_MSG

    CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
    {
        ///AFX_DATA_INIT(CAboutDlg)
        ///AFX_DATA_INIT
    }

    void CAboutDlg::DoDataExchange(CDataExchange* pDX)
    {
        CDialog::DoDataExchange(pDX);
        ///AFX_DATA_MAP(CAboutDlg)
        ///AFX_DATA_MAP
    }

    BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    ///AFX_MSG_MAP(CAboutDlg)
    // No message handlers
    ///AFX_MSG_MAP
    END_MESSAGE_MAP()

    // CExercise2Dlg dialog
    
    CExercise2Dlg::CExercise2Dlg(CWnd* pParent /*=NULL*/)
        : CDialog(CExercise2Dlg::IDD, pParent)
    {

Appendix B  Sample C++ Application

//{{AFX_DATA_INIT(CExercise2Dlg)
//}}AFX_DATA_INIT
// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CExercise2Dlg::DoDataExchange(CDataExchange* pDX)
{
    Dialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CExercise2Dlg)
    DDX_Control(pDX, IDCCCONFERENCE, m_btnCCConference);
    DDX_Control(pDX, IDCCTRANSFER, m_btnCCTransfer);
    DDX_Control(pDX, IDSSCONFERENCE, m_btnSSConference);
    DDX_Control(pDX, IDSSTRANSFER, m_btnSSTransfer);
    DDX_Control(pDX, IDTRANSFER, m_btnTransfer);
    DDX_Control(pDX, IDRETRIEVE, m_btnRetrieve);
    DDX_Control(pDX, IDRELEASE, m_btnRelease);
    DDX_Control(pDX, IDREADY, m_btnReady);
    DDX_Control(pDX, IDNOTREADY, m_btnNotReady);
    DDX_Control(pDX, IDMARECALL, m_btmnMakeCall);
    DDX_Control(pDX, IDLOGOUT, m_btmnLogout);
    DDX_Control(pDX, IDLOGIN, m_btmnLogin);
    DDX_Control(pDX, IDHOLD, m_btmnHold);
    DDX_Control(pDX, IDDISCONNECT, m_btmnDisconnect);
    DDX_Control(pDX, IDCONNECT, m_btmnConnect);
    DDX_Control(pDX, IDC_CONFERENCE, m_btmnConference);
    DDX_Control(pDX, IDANSWER, m_btmnAnswer);
    DDX_Control(pDX, IDC_LOGWINDOW, m_btmLogWindow);
    DDX_Control(pDX, IDC_CURRENTCALLUID, m_btmCallUID);
    DDX_Control(pDX, IDC_CONNECTION, m_btmConnection);
    DDX_Control(pDX, IDC_AGENTSTATE, m_btmAgentState);
    DDX_Control(pDX, IDC_AGENTID, m_btmAgentID);
    DDX_Control(pDX, IDC_CALLVARIABLE1, m_btmCallVariable1);
    DDX_Control(pDX, IDC_INSTRUMENT, m_btmInstrument);
    DDX_Control(pDX, IDC_PASSWORD, m_btmPassword);
    DDX_Control(pDX, IDC_PERIPHERALID, m_btmPeripheralID);
    DDX_Control(pDX, IDC_PHONENUMBER, m_btmPhoneNumber);
    DDX_Control(pDX, IDC_PORTB, m_btmPortB);
    DDX_Control(pDX, IDC_PORTA, m_btmPortA);
    DDX_Control(pDX, IDC_HOSTB, m_btmHostB);
    DDX_Control(pDX, IDC_HOSTA, m_btmHostA);
    DDX_Control(pDX, IDC_MODE, m_btmMode);
   //}}AFX_DATA_MAP

BEGIN_MESSAGE_MAP(CExercise2Dlg, CDialog)
   //{{AFX_MSG_MAP(CExercise2Dlg)
    ON_WM_SYSCOMMAND()
ON_WM_PAINT()
ON_WM_QUERYDRAGICON()
ON_BN_CLICKED(IDCONNECT, OnConnect)
ON_BN_CLICKED(IDDISCONNECT, OnDisconnect)
ON_BN_CLICKED(IDLOGIN, OnLogin)
ON_BN_CLICKED(IDLOGOUT, OnLogout)
ON_BN_CLICKED(IDREADY, OnReady)
ON_BN_CLICKED(IDNOTREADY, OnNotready)
ON_BN_CLICKED(IDMAKECALL, OnMakecall)
ON_BN_CLICKED(IDANSWER, OnAnswer)
ON_BN_CLICKED(IDRELEASE, OnRelease)
ON_BN_CLICKED(IDRETRIEVE, OnRetrieve)
ON_BN_CLICKED(IDTRANSFER, OnTransfer)
ON_BN_CLICKED(IDHOLD, OnHold)
ON_BN_CLICKED(IDCLEARLOG, OnClearlog)
ON_WM_CLOSE()
ON_BN_CLICKED(IDCCONFERENCE, OnConference)
ON_BN_CLICKED(IDCTRANSFER, OnCCTransfer)
ON_BN_CLICKED(IDCLEARLOG, OnClearlog)
ON_WM_CLOSE()
ON_BN_CLICKED(IDCCONFERENCE, OnConference)
ON_BN_CLICKED(IDCTRANSFER, OnCCTransfer)
ON_BN_CLICKED(IDCLEARLOG, OnClearlog)
ON_WM_CLOSE()
ON_BN_CLICKED(IDCCONFERENCE, OnConference)
ON_BN_CLICKED(IDCTRANSFER, OnCCTransfer)
ON_BN_CLICKED(IDCLEARLOG, OnClearlog)
ON_WM_CLOSE()
ON_BN_CLICKED(IDCCONFERENCE, OnConference)
ON_BN_CLICKED(IDCTRANSFER, OnCCTransfer)
ON_BN_CLICKED(IDCLEARLOG, OnClearlog)
ON_WM_CLOSE()
ON_BN_CLICKED(IDCCONFERENCE, OnConference)
ON_BN_CLICKED(IDCTRANSFER, OnCCTransfer)
ON_BN_CLICKED(IDCLEARLOG, OnClearlog)
ON_WM_CLOSE()
ON_BN_CLICKED(IDCCONFERENCE, OnConference)
ON_BN_CLICKED(IDCTRANSFER, OnCCTransfer)
ON_BN_CLICKED(IDCLEARLOG, OnClearlog)
ON_WM_CLOSE()
Appendix B  Sample C++ Application

```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);

// initialize edit boxes
m_edtConnection.SetWindowText("Offline");
m_edtHostA.SetWindowText("localhost");
m_edtHostB.SetWindowText("localhost");
m_edtAgentID.SetWindowText("5000");
m_edtPassword.SetWindowText("5000");
m_edtInstrument.SetWindowText("5000");
m_edtPeripheralID.SetWindowText("5000");
m_edtCallID.SetWindowText("no call");
m_edtMode.SetWindowText("A, B, or C");
SetSessionConnected(FALSE);
SetAgentState(eUnknown);
Log("Welcome to the VC++ CTIOS CIL Softphone Sample\r\n");
Log("To start please connect\r\n");

// disable buttons
m_btnLogin.EnableWindow(FALSE);
m_btnLogout.EnableWindow(FALSE);
m_btnReady.EnableWindow(FALSE);
m_btnNotReady.EnableWindow(FALSE);
m_btnMakeCall.EnableWindow(FALSE);
m_btnAnswer.EnableWindow(FALSE);
m_btnRelease.EnableWindow(FALSE);
m_btnHold.EnableWindow(FALSE);
```

Cisco ICM Software CTI OS Developer's Guide
m_btnRetrieve.EnableWindow(FALSE);
m_btnCCConference.EnableWindow(FALSE);
m_btnConference.EnableWindow(FALSE);
m_btnSSConference.EnableWindow(FALSE);
m_btnCCTransfer.EnableWindow(FALSE);
m_btnTransfer.EnableWindow(FALSE);
m_btnSSTransfer.EnableWindow(FALSE);

return TRUE;    // return TRUE unless you set the focus to a control

void CExercise2Dlg::OnClose()
{

    if (m_bSessionConnected == TRUE)
    {
        // Setup to wait for the OnConnectionClosed event following Disconnect()
        //
        // Must do this since CIL is a multi-threaded library in order to allow
        // CIL to finish before deleting things out from underneath it
        //
        Arguments & argsWaitParams = Arguments::CreateInstance();

        CWaitObject *pWaitCall = m_ctiSession.CreateWaitObject(argsWaitParams);

        argsWaitParams.AddItem("Event1", eOnConnectionClosed);

        pWaitCall->SetMask(argsWaitParams);

        // Perform disconnect
        m_ctiSession.Disconnect();

        // Wait for OnConnectionClosed
        pWaitCall->WaitOnMultipleEvents();

        // free objects
        m_ctiSession.DestroyWaitObject(pWaitCall);
        argsWaitParams.Release();
    }

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

    CDialog::OnClose();
}
void CExercise2Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.
void CExercise2Dlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
// the minimized window.
HCURSOR CExercise2Dlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}
void CExercise2Dlg::OnConnect()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == TRUE)
    {
        AddLogMessage("Already Connected!");
        return;
    }

    AddLogMessage("Connecting to CTIOS Server...");

    // Create Arguments array. Note, auto AddRef so don't forget to Release
    Arguments &rArgSessionConn = Arguments::CreateInstance();

    CString hostA;
    CString portA;
    CString hostB;
    CString portB;

    m_edtHostA.GetWindowText(hostA);
    m_edtPortA.GetWindowText(portA);
    m_edtHostB.GetWindowText(hostB);
    m_edtPortB.GetWindowText(portB);

    rArgSessionConn.AddItem(CTIOS_CTIOSA, hostA.GetBuffer(0));
    rArgSessionConn.AddItem(CTIOS_PORTA, portA.GetBuffer(0));
    rArgSessionConn.AddItem(CTIOS_CTIOSB, hostB.GetBuffer(0));
    rArgSessionConn.AddItem(CTIOS_PORTB, portB.GetBuffer(0));

    rArgSessionConn.AddItem(CTIOS_HEARTBEAT, 5);

    // perform Connect
    int nRetVal = m_ctiSession.Connect(rArgSessionConn);

    // free object
    rArgSessionConn.Release();

    if (CIL_FAILED(nRetVal))
    {
        AddLogMessage("Session->Connect to CTIOS Server Failed");
    }
    else
    {
        AddLogMessage("Session->Connect to CTIOS Server Succeeded");
        m_edtConnection.SetWindowText("Connecting...");
    }
}
Appendix B  Sample C++ Application

```c++
void CExercise2Dlg::OnDisconnect()
{
    AddLogMessage("Disconnecting from CTIOS Server...");

    // perform Disconnect - no arguments required
    m_ctiSession.Disconnect();

    m_edtConnection.SetWindowText("Offline");
}

void CExercise2Dlg::OnLogin()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = m_ctiAgent.GetAgentState();
    if (.currentState != eUnknown && currentState != eLogout )
    {
        AddLogMessage("Already logged in!");
        return;
    }

    CString agentID;
    CString password;
    CString instrument;
    CString peripheralID;

    // get agent info from dialog
    m_edtAgentID.GetWindowText(agentID);
    m_edtPassword.GetWindowText(password);
    m_edtInstrument.GetWindowText(instrument);
    m_edtPeripheralID.GetWindowText(peripheralID);

    // set the agent object
    m_ctiAgent.SetValue(CTIOS_AGENTID, (LPCTSTR) agentID);
    m_ctiAgent.SetValue(CTIOS_AGENTPASSWORD, (LPCTSTR) password);
    m_ctiAgent.SetValue(CTIOS_AGENTINSTRUMENT, (LPCTSTR) instrument);
    m_ctiAgent.SetValue(CTIOS_PERIPHERALID, (LPCTSTR) peripheralID);
    AddLogMessage("Requesting SetAgent");

    // perform SetAgent
    m_ctiSession.SetAgent(m_ctiAgent);
```
void CExercise2Dlg::OnLogout()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = m_ctiAgent.GetAgentState();
    if (currentState == eUnknown || currentState == eLogout)
    {
        AddLogMessage("Must login first!");
        return;
    }

    Arguments &arLogoutReq = Arguments::CreateInstance();

    // No arguments required
    m_ctiAgent.Logout(arLogoutReq);
    arLogoutReq.Release();
    AddLogMessage("Requesting Agent Logout");
}

void CExercise2Dlg::OnReady()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = m_ctiAgent.GetAgentState();
    if (currentState == eUnknown || currentState == eLogout)
    {
        AddLogMessage("Must login first!");
    }
return;
}

// create and add arguments
Arguments &argReady = Arguments::CreateInstance();
argReady.AddItem(CTIOS_AGENTSTATE, eAvailable);
argReady.AddItem(CTIOS_REASONCODE, _T("1234"));
m_ctiAgent.SetAgentState(argReady);
argReady.Release();
AddLogMessage("Requesting Agent State = Ready");
}

void CExercise2Dlg::OnNotready()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = m_ctiAgent.GetAgentState();
    if (currentState == eUnknown || currentState == eLogout)
    {
        AddLogMessage("Must login first!");
        return;
    }

    // create and add arguments
    Arguments &argNotReady = Arguments::CreateInstance();
    argNotReady.AddItem(CTIOS_AGENTSTATE, eNotReady);
    argNotReady.AddItem(CTIOS_REASONCODE, _T("1234"));
    m_ctiAgent.SetAgentState(argNotReady);
    argNotReady.Release();
    AddLogMessage("Requesting Agent State = NotReady");
}

void CExercise2Dlg::OnMakecall()
{
// Check to see if we are already connected
if (m_bSessionConnected == FALSE)
{
    AddLogMessage("Must be connected first!");
    return;
}

// Check to see if we are already logged in
enumCTIOS_AgentState currentState = m_ctiAgent.GetAgentState();
if (currentState == eUnknown || currentState == eLogout)
{
    AddLogMessage("Must login first!");
    return;
}

// get phone number and call variable (if present) from dialog
CString dn;
CString cv;
m_edtPhoneNumber.GetWindowText(dn);
m_edtCallVariable1.GetWindowText(cv);

if (dn.IsEmpty() == TRUE)
{
    AddLogMessage("Must provide phone number to dial!");
    return;
}

// create and add arguments
Arguments argMakeCall = Arguments::CreateInstance();
argMakeCall.AddItem(CTIOS_DIALEDNUMBER, dn.GetBuffer(0));
if (cv.IsEmpty() != TRUE)
{
    argMakeCall.AddItem(CTIOS_CALLVARIABLE1, cv.GetBuffer(0));
}
m_ctiAgent.MakeCall(argMakeCall);
argMakeCall.Release();
AddLogMessage("Requesting MakeCall");

void CExercise2Dlg::OnAnswer()
{
    // Check to see if we are already connected
if (m_bSessionConnected == FALSE) {
    AddLogMessage("Must be connected first!");
    return;
}

// Check to see if we are already logged in
enumCTIOS_AgentState currentState = m_ctiAgent.GetAgentState();
if (currentState == eUnknown || currentState == eLogout) {
    AddLogMessage("Must login first!");
    return;
}

// retrieve the currently active call object
CILRefArg & rCArg = (CILRefArg &) m_ctiSession.GetValue("ActiveCall");
CCall *pCall = (CCall *)rCArg.GetValue();
if (pCall) {
    AddLogMessage("Requesting Call Answer");
    pCall->Answer();
    pCall->Release();
}
    rCArg.Release();
}

void CExercise2Dlg::OnRelease() {
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE) {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = m_ctiAgent.GetAgentState();
    if (currentState == eUnknown || currentState == eLogout) {
        AddLogMessage("Must login first!");
        return;
    }
/ retrieve the currently active call
CILRefArg & rCArg = (CILRefArg & ) m_ctiSession.GetValue("ActiveCall");
CCall *pCall = (CCall *) rCArg.GetValue();
if (pCall)
{
    AddLogMessage("Requesting Call Release");
    pCall->ClearConnection();
    pCall->Release();
}
rCArg.Release();

void CExercise2Dlg::OnHold()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = m_ctiAgent.GetAgentState();
    if (currentState == eUnknown || currentState == eLogout )
    {
        AddLogMessage("Must login first!");
        return;
    }

    // retrieve the currently active call
    CILRefArg & rCArg = (CILRefArg & ) m_ctiSession.GetValue("ActiveCall");
    CCall *pCall = (CCall *) rCArg.GetValue();
    if (pCall)
    {
        AddLogMessage("Requesting Call Hold");
        pCall->Hold();
        pCall->Release();
    }
Appendix B  Sample C++ Application

void CExercise2Dlg::OnRetrieve()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    CILRefArg & rCArg = (CILRefArg &) m_ctiSession.GetValue("ActiveCall");
    if (pCall)
    {
        AddLogMessage("Requesting Call Retrieve");
        pCall->Retrieve();
        pCall->Release();
    }
    rCArg.Release();
}

void CExercise2Dlg::OnCConference()
{
    // Get Phone number
    CString dn;
    m_edtPhoneNumber.GetWindowText(dn);

    if (dn.IsEmpty() == TRUE)
    {
        AddLogMessage("Must fill in phone number field for conference");
        return;
    }
}
// Get Call Variable1 value
CString cv1;
m edtCallVariable1.GetWindowText(cv1);

// Retrieve the currently active call object
CILRefArg & rCArg = (CILRefArg &) m_ctiSession.GetValue("ActiveCall");
CCall *pCall = (CCall *)rCArg.GetValue();
if (pCall)
{
    AddLogMessage("Requesting Consult Call - Conference");
    // create and add arguments
    Arguments &rArgConference = Arguments::CreateInstance();
    rArgConference.AddItem(CTIOS_DIALEDNUMBER, dn.GetBuffer(0));
    rArgConference.AddItem(CTIOS_CALLVARIABLE1, cv1.GetBuffer(0));
    rArgConference.AddItem(CTIOS_CONSULTTYPE, 2);
    pCall->MakeConsultCall(rArgConference);
    rArgConference.Release();
    pCall->Release();
}

void CExercise2Dlg::OnConference()
{
    // Retrieve the currently active call object
    CILRefArg & rCArg = (CILRefArg &) m_ctiSession.GetValue("ActiveCall");
    CCall *pCall = (CCall *)rCArg.GetValue();
    if (pCall)
    {
        AddLogMessage("Requesting Conference Complete");
        pCall->Conference();
        pCall->Release();
    }
```cpp
void CExercise2Dlg::OnSSConference()
{
    // Get Phone number
    CString dn;
    m_edtPhoneNumber.GetWindowText(dn);
    if (dn.IsEmpty() == TRUE)
    {
        AddLogMessage("Must fill in phone number field for conference");
        return;
    }
    // Get Call Variable1 value
    CString cv1;
    m_edtCallVariable1.GetWindowText(cv1);
    // Retrieve the currently active call object
    CILRefArg & rCArg = (CILRefArg &) m_ctiSession.GetValue("ActiveCall");
    CCall *pCall = (CCall *)rCArg.GetValue();
    if (pCall)
    {
        AddLogMessage("Requesting Single Step Conference");
        // create and add arguments
        Arguments &rArgConference = Arguments::CreateInstance();
        rArgConference.AddItem(CTIOS_DIALEDNUMBER, dn.GetBuffer(0));
        rArgConference.AddItem(CTIOS_CALLVARIABLE1, cv1.GetBuffer(0));
        pCall->SingleStepConference(rArgConference);
        rArgConference.Release();
        pCall->Release();
    }
    rCArg.Release();
}

void CExercise2Dlg::OnCCTransfer()
{
    // Get Phone number from dialog
    CString dn;
    // Get Call Variable2 value
    CString cv2;
    m_edtCallVariable2.GetWindowText(cv2);
    // Retrieve the currently active call object
    CILRefArg & rCArg = (CILRefArg &) m_ctiSession.GetValue("ActiveCall");
    CCall *pCall = (CCall *)rCArg.GetValue();
    if (pCall)
    {
        AddLogMessage("Requesting Call Transfer");
        // create and add arguments
        Arguments &rArgTransfer = Arguments::CreateInstance();
        rArgTransfer.AddItem(CTIOS_DIALEDNUMBER, dn.GetBuffer(0));
        rArgTransfer.AddItem(CTIOS_CALLVARIABLE2, cv2.GetBuffer(0));
        pCall->TransferConference(rArgTransfer);
        rArgTransfer.Release();
        pCall->Release();
    }
    rCArg.Release();
}
```

The code snippet above demonstrates how to implement Single Step Conference and Call Transfer functionalities in a Sample C++ Application using the Cisco ICM Software CTI OS Developer's Guide.
m_edtPhoneNumber.GetWindowText(dn);

if (dn.IsEmpty() == TRUE)
{
    AddLogMessage("Must fill in phone number field for transfer");
    return;
}

// Get Call Variable1 value
CString cv1;
if (!m_edtCallVariable1.GetWindowText(cv1))
{
    return;
}

// Retrieve currently active call object
CILRefArg & rCArg = (CILRefArg &) m_ctiSession.GetValue("ActiveCall");
CCall *pCall = (CCall *)rCArg.GetValue();

if (pCall)
{
    AddLogMessage("Requesting Consult Call - Transfer");

    // create and add arguments
    Arguments &rArgTransfer = Arguments::CreateInstance();
    rArgTransfer.AddItem(CTIOS_DIALEDNUMBER, dn.GetBuffer(0));
    rArgTransfer.AddItem(CTIOS_CALLVARIABLE1, cv1.GetBuffer(0));
    rArgTransfer.AddItem(CTIOS_CONSULTTYPE, 1);
    pCall->MakeConsultCall(rArgTransfer);
    rArgTransfer.Release();
    pCall->Release();
}

rCArg.Release();

void CExercise2Dlg::OnTransfer()
{
    // Retrieve currently active call object
    CILRefArg & rCArg = (CILRefArg &) m_ctiSession.GetValue("ActiveCall");

    CCall *pCall = (CCall *)rCArg.GetValue();

    if (pCall)
    {
        AddLogMessage("Requesting Transfer Complete");
    }
pCall->Transfer();

pCall->Release();
}
rCArg.Release();
}

void CExercise2Dlg::OnSSTransfer()
{
    // Get Phone number from dialog
    CString dn;
    m_edtPhoneNumber.GetWindowText(dn);

    if (dn.IsEmpty() == TRUE)
    {
        AddLogMessage("Must fill in phone number field for transfer");
        return;
    }

    // Get Call Variable1 value
    CString cv1;
    m_edtCallVariable1.GetWindowText(cv1);

    // Retrieve currently active call object
    CILRefArg & rCArg = (CILRefArg&) m_ctiSession.GetValue("ActiveCall");
    CCall *pCall = (CCall*) rCArg.GetValue();

    if (pCall)
    {
        AddLogMessage("Requesting Single Step Transfer");

        // create and add arguments
        Arguments & rArgTransfer = Arguments::CreateInstance();
        rArgTransfer.AddItem(CTIOS_DIALEDNUMBER, dn.GetBuffer(0));
        rArgTransfer.AddItem(CTIOS_CALLVARIABLE1, cv1.GetBuffer(0));

        pCall->SingleStepTransfer(rArgTransfer);
        rArgTransfer.Release();
        pCall->Release();
    }

    rCArg.Release();
}
void CExercise2Dlg::OnGetData()
{
    int nErrorCode;
    AddLogMessage("Get Data pressed.");
    // Retrieve currently active call object
    CILRefArg & rCArg = (CILRefArg &) m_ctiSession.GetValue(\"ActiveCall\");
    CCall *pCall = (CCall *)rCArg.GetValue();
    if (pCall)
    {
        AddLogMessage("Requesting Get Data");
        // create arguments
        Arguments &rArgs = Arguments::CreateInstance();
        nErrorCode = pCall->GetCallData(rArgs);
        LogEvent("GetCallData returns:\n", rArgs);
        rArgs.Release();
        pCall->Release();
    }
    else
    {
        AddLogMessage("No active call to get data from.");
    }
    rCArg.Release();
}

void CExercise2Dlg::OnSetData()
{
    int nErrorCode;
    AddLogMessage("Set Data pressed.");
    // Retrieve currently active call object
    CILRefArg & rCArg = (CILRefArg &) m_ctiSession.GetValue(\"ActiveCall\");
    CCall *pCall = (CCall *)rCArg.GetValue();
if (pCall)
{
    // create arguments
    Arguments &rArgs = Arguments::CreateInstance();
    Arguments &rECCArgs = Arguments::CreateInstance();

    // regular variables
    rArgs.AddItem("ANI", "MyANI");
    rArgs.AddItem("DNIS", "MyDNIS");
    rArgs.AddItem("CED", "MyCED");
    rArgs.AddItem("CallVariable1", "MyCallVariable1");

    // ECC variables
    rECCArgs.AddItem("Scalar1", "MyScalar1");
    rECCArgs.AddItem("Scalar2", "MyScalar2");
    rECCArgs.AddItem("Array1[1]", "FirstArrayValue");
    rECCArgs.AddItem("Array1[2]", "SecondArrayValue");

    // Add ECC arguments to args
    rArgs.AddItem("ECC", rECCArgs);

    nErrorCode = pCall->SetCallData(rArgs);
    LogEvent("SetData sending:", rArgs);
}
else
{
    AddLogMessage("No active call to set data into.");
}

void CExercise2Dlg::OnGetContext()
{
    int nErrorCode;

    AddLogMessage("Get Context pressed.");

    // Retrieve currently active call object
    CILRefArg & rCArg = (CILRefArg &) m_ctiSession.GetValue("ActiveCall");
    CCall *pCall = (CCall *)rCArg.GetValue();

    if (pCall)
    {
        // create arguments
        Arguments &rArgs = Arguments::CreateInstance();

nErrorCode = pCall->GetCallContext(rArgs);
LogEvent("GetCallContext returns:\", rArgs);
else
{
    AddLogMessage("No active call to get context data from.");
}

void CExercise2Dlg::OnClearlog()
{
    // clear text from log window
    m_edtLogWindow.SetWindowText("\n");
}

// Convert agent state enum to string
//
char * CExercise2Dlg::AgentStateToString(enumCTIOS_AgentState eState)
{
    switch( eState )
    {
        case eLogin:
            return "Login";
        case eLogout:
            return "Logout";
        case eNotReady:
            return "NotReady";
        case eAvailable:
            return "Available";
        case eTalking:
            return "Talking";
        case eWorkNotReady:
            return "WorkNotReady";
        case eWorkReady:
            return "WorkReady";
        case eBusyOther:
            return "BusyOther";
        case eReserved:
            return "Reserved";
        case eUnknown:
            return "Unknown";
        case eHold:
            return "Hold";
        default:
            return "Invalid State";
void CExercise2Dlg::LogEvent(const char *msg, Arguments & rArguments)
{
    CString str;
    // display event name
    str = msg;
    str += " Event received";
    Log(str.GetBuffer(0));

    // create and display event contents
    str = "\t";
    std::string s = rArguments.DumpArgs();
    str += s.c_str();
    //str += sz;

    str.Replace(" ", "\r\n\t"); // print each element on new line plus tab offset
    str += "\r\n";
    Log(str.GetBuffer(0));
}

void CExercise2Dlg::Log(const char *newText)
{
    CString oldMsg;
    CString newMsg;
    m_edtLogWindow.GetWindowText(oldMsg);
    newMsg = oldMsg + "\r\n" + newText;
    m_edtLogWindow.SetWindowText(newMsg);
    m_edtLogWindow.SetSel(newMsg.GetLength(), -1);
}

}
// stdafx.h : include file for standard system include files, 
// or project specific include files that are used frequently, but 
// are changed infrequently 
//
#if !defined(AFX_STDAFX_H__980BDB20_1D40_4E38_92B0_EEE21B53B738__INCLUDED_) 
define AFX_STDAFX_H__980BDB20_1D40_4E38_92B0_EEE21B53B738__INCLUDED_

#if _MSC_VER > 1000 
#pragma once 
#endif // _MSC_VER > 1000 

#define VC_EXTRALEAN// Exclude rarely-used stuff from Windows headers 
#include <afxwin.h>         // MFC core and standard components 
#include <afxext.h>         // MFC extensions 
#include <afxdisp.h>        // MFC Automation classes 
#include <afxdtctl.h>// MFC support for Internet Explorer 4 Common Controls 
ifndef _AFX_NO_AFXCMN_SUPPORT 
#include <afxcmn.h> // MFC support for Windows Common Controls 
endif // _AFX_NO_AFXCMN_SUPPORT

//{{AFX_INSERT_LOCATION}} 
// Microsoft Visual C++ will insert additional declarations immediately before the 
// previous line. 
#endif // !defined(AFX_STDAFX_H__980BDB20_1D40_4E38_92B0_EEE21B53B738__INCLUDED_) ----------------------------------------------- End "stdafx.h" -----------------------------------------------

--- Start "stdafx.cpp" -----------------------------------------------
// stdafx.cpp : source file that includes just the standard includes 
// exercise2.pch will be the pre-compiled header 
// stdafx.obj will contain the pre-compiled type information 

#include "stdafx.h"

End "stdafx.cpp" -----------------------------------------------

--- Start "resource.h" -----------------------------------------------
//{{NO_DEPENDENCIES}} 
// Microsoft Developer Studio generated include file. 
// Used by exercise2.rc 
//
#define IDCONNECT 3 
#define IDD_DISCONNECT 4 
#define IDLOGIN 5

--- End "resource.h" -----------------------------------------------
Appendix B      Sample C++ Application

#define IDREADY 6
#define IDLOGOUT 7
#define IDNOTREADY 8
#define IDCLEARLOG 9
#define IDMAKECALL 11
#define IDANSWER 13
#define IDRELEASE 14
#define IDHOLD 15
#define IDM_ABOUTBOX 0x0010
#define IDRETRIEVE 16
#define IDTRANSFER 17
#define IDC_ABOUTBOX 18
#define IDD_ABOUTBOX 100
#define IDS_ABOUTBOX 101
#define IDD_EXERCISE2_DIALOG 102
#define IDR_MAINFRAME 128
#define IDC_HOSTA 1000
#define IDC_HOSTA 1001
#define IDC_HOSTB 1002
#define IDC_PORTB 1003
#define IDC_AGENTID 1004
#define IDC_PASSWORD 1005
#define IDC_INSTRUMENT 1006
#define IDC_PERIPHERALID 1007
#define IDC_CONNECTION 1008
#define IDC_AGENTSTATE 1009
#define IDC_MODE 1011
#define IDC_PHONENUMBER 1012
#define IDC_CALLVARIABLE1 1013
#define IDC_CURRENTCALLUID 1014
#define IDC покупатель 1015
#define IDC покупатель 1017
#define IDC покупатель 1018
#define IDC покупатель 1019
#define IDC покупатель 1020
#define IDC покупатель 1021
#define IDC покупатель 1022
#define IDC покупатель 1023

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#define APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE 129
#define _APS_NEXT_COMMAND_VALUE 32771
#define _APS_NEXT_CONTROL_VALUE 1024
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif

------------------------ End "resource.h" --------------------------
Sample Visual Basic Application

This appendix provides a sample CTI OS Visual Basic application for the example CTI OS phone shown in Figure C-1.

Figure C-1   Example CTI OS Phone

* VB sample for a simple CTIOS phone
' needs references to CTIOSCLIENTLib and CTIOSSESSIONRESOLVERLib
'
' 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
'
' dim an agent and a call object
Dim m_Agent As CTIOSCLIENTLib.Agent
Dim m_call As CTIOSCLIENTLib.Call
'
' flag used to keep track of connection status
Dim m_bConnected As Boolean

Private Sub Form_Load()

' instantiate the sessionresolver
Set m_sessionresolver = New CTIOSSESSIONRESOLVERLib.SessionResolver
Set m_Agent = Nothing

' get the session pointer and hold on to it
Set m_session = m_sessionresolver.GetSession(""

m_bConnected = False

connectionstatus.Text = "Not Connected"
agentstate.Text = "Unknown"
calluid.Text = "no call"

' Disable buttons
Login.Enabled = False
Logout.Enabled = False
Ready.Enabled = False
NotReady.Enabled = False
MakeCall.Enabled = False
Answer.Enabled = False
Release.Enabled = False
Hold.Enabled = False
Retrieve.Enabled = False
CCConference.Enabled = False
Conference.Enabled = False
SSConference.Enabled = False
CCTransfer.Enabled = False
Transfer.Enabled = False
SSTransfer.Enabled = False
Log "Welcome to the VB CTIOS COM Softphone Sample"
Log ""
Log "To start please connect"
check_details = 1
End Sub

Private Sub Form_Unload(Cancel As Integer)
    If m_bConnected = True Then
        m_session.Disconnect
    End If
    Call m_sessionresolver.RemoveSession("")
    Set m_Agent = Nothing
    Set m_session = Nothing
End Sub

Private Sub Connect_Click()
    Log "Connect Button Clicked"
    If m_bConnected = True Then
        Log "Already connected"
    Else
        ' Not connected so pull values from dialog and use to make connection attempt
        Dim m_Args As New Arguments
        m_Args.AddItem "CtiosA", hostA.Text
        Dim pA As Long
        pA = Val(portA.Text)
        m_Args.AddItem "portA", pA
        m_Args.AddItem "CtiosB", hostB.Text
        Dim pB As Long
        pB = Val(portB.Text)
        m_Args.AddItem "portB", pB
        Dim hb As Long
        hb = Val(heartbeats.Text)
        m_Args.AddItem "heartbeat", hb
        Dim nRetVal As Long
        nRetVal = m_session.Connect(m_Args)
Log "Sending Connect request"
End If

Private Sub Disconnect_Click()
    Log "Send Disconnect Request"
    m_session.Disconnect
End Sub

Private Sub Login_Click()
    Log "Login button clicked"
    If m_bConnected = False Then
        Log "Cannot Login - Please connect first"
    Else
        ' Connected so verify required info is available on the dialog
        If ((0 = Len(agentid.Text)) Or (0 = Len(instrument.Text)) Or (0 = Len(peripheralid.Text))) Then
            Log "Missing required field for Login"
            Log "Please enter Agent ID, Instrument and Peripheral ID"
        Else
            Dim nRetVal As Long

            ' create a new agent
            Set m_Agent = New Agent

            ' initialize the agent object using SetValue
            m_Agent.SetValue "Agentid", agentid.Text
            m_Agent.SetValue "AgentPassword", agentpw.Text
            m_Agent.SetValue "AgentInstrument", instrument.Text
            m_Agent.SetValue "PeripheralID", peripheralid.Text

            Log "SetAgent request"

            ' tell the session about the agent
            m_session.SetAgent m_Agent

        End If
    End If
End Sub

Private Sub Logout_Click()
    Log "Logout Button Clicked"
    If m_Agent Is Nothing Then
        Log "Please Login first "
    End If
End Sub
Else
   ' Verify agent state is Not_Ready first
   If agentstate.Text = "Ready" Then
      Log "To Logout please Make Agent Not_Ready first"
   Else
      Dim m_Args As New Arguments

      m_Agent.Logout m_Args

      Set m_Agent = Nothing
   End If
End If
End Sub

Private Sub Ready_Click()
   Log "Ready button clicked"

   If m_Agent Is Nothing Then
      Log "Please Login first"
   Else
      HandleSetAgentStateRequest 3
   End If
End Sub

Private Sub NotReady_Click()
   Log "NotReady button clicked"

   If m_Agent Is Nothing Then
      Log "Please Login first"
   Else
      HandleSetAgentStateRequest 2
   End If
End Sub

Private Sub MakeCall_Click()
   Dim nRetVal As Long

   Log "MakeCall Button Clicked"

   If dialthis.Text = "" Then
      Log "To make a call please enter dialstring first"
   Else
      If m_Agent Is Nothing Then
         Log "Please Login first"
      Else
         ' Pack up arguments and send MakeCall request
Dim m_Args As New Arguments
m_Args.AddItem "dialednumber", dialthis.Text
If Not 0 = Len(callvar1.Text) Then
    ' set callvar1 if present
    m_Args.AddItem "Callvariable1", callvar1.Text
End If

' send makecall request
m_Agent.MakeCall m_Args

Log "Sending MakeCall request DialedNumber =" + dialthis.Text
End If
End If
End Sub

Private Sub Answer_Click()
    Log "Answer button clicked"
    If Not m_call Is Nothing Then
        m_call.Answer
    Else
        Log "no call - cannot answer"
    End If
End Sub

Private Sub Release_Click()
    Log "Release button clicked"
    If Not m_call Is Nothing Then
        m_call.ClearConnection
    Else
        Log "no call - cannot release"
    End If
End Sub

Private Sub Hold_Click()
    Log "Hold button clicked"
    If Not m_call Is Nothing Then
        m_call.Hold
    Else
        Log "no call - cannot put on hold"
    End If
End Sub

Private Sub Retrieve_Click()
Log "Retrieve Button clicked"
If Not m_call Is Nothing Then
  m_call.Retrieve
Else
  Log "No call - cannot retrieve"
End If
End Sub

Private Sub CCConference_Click()
Log "CC Conf Button clicked"
If m_call Is Nothing Then
  Log "no call - Conference not possible"
Else
  ' Pack up arguments and pass to MakeConsultCall
  Dim m_Args As New Arguments
  If 0 = Len(dialthis.Text) Then
    Log "Please enter Dialstring in Dial Box first"
  Else
    m_Args.AddItem "dialednumber", dialthis.Text
    If 0 = Len(callvar1.Text) Then
      ' set callvar1 if present
      m_Args.AddItem "Callvariable1", callvar1.Text
    End If
    ' Set ConsultType to Conference (2)
    m_Args.AddItem "ConsultType", 2
    ' send request
    m_call.MakeConsultCall m_Args
    Log "Sending CC Conference request DialedNumber =" + dialthis.Text
  End If
End If
End Sub

Private Sub Conference_Click()
Log "Conference Button clicked"
If m_call Is Nothing Then
  Log "no call - Conference not possible"
Else
  ' Pack up arguments and pass to Conference
Dim m_Args As New Arguments
' Add args here
' send request
m_call.Conference m_Args
Log "Sending Conference request"

End If
End Sub

Private Sub SSConference_Click()
Log "SS Conf Button clicked"
If m_call Is Nothing Then
  Log "no call - Conference not possible"
Else
  ' Pack up arguments and pass to SingleStepConference
  Dim m_Args As New Arguments
  If 0 = Len(dialthis.Text) Then
    Log "Please enter Dialstring in Dial Box first"
  Else
    m_Args.AddItem "dialednumber", dialthis.Text
    If Not 0 = Len(callvar1.Text) Then
      ' set callvar1 if present
      m_Args.AddItem "Callvariable1", callvar1.Text
    End If
  End If
  ' send request
  m_call.SingleStepConference m_Args
  Log "Sending SS Conference request DialedNumber =" + dialthis.Text
End If

End If
End Sub

Private Sub CCTransfer_Click()
Log "CC Transfer Button clicked"
If m_call Is Nothing Then
  Log "no call - Transfer not possible"
Else
' Pack up arguments and pass to MakeConsultCall
Dim m_Args As New Arguments
If 0 = Len(dialthis.Text) Then
  Log "Please enter Dialstring in Dial Box first"
Else
  m_Args.AddItem "dialednumber", dialthis.Text
  If Not 0 = Len(callvar1.Text) Then
    ' set callvar1 if present
    m_Args.AddItem "Callvariable1", callvar1.Text
  End If
  ' Set ConsultType to Transfer (1)
  m_Args.AddItem "ConsultType", 1
  ' send request
  m_call.MakeConsultCall m_Args
  Log "Sending CC Transfer request DialedNumber =" + dialthis.Text
  End If
End If
End Sub

Private Sub Transfer_Click()
  Log "Transfer Button clicked"
  If m_call Is Nothing Then
    Log "no call - Transfer not possible"
  Else
    ' Pack up arguments and pass to Transfer
    Dim m_Args As New Arguments
    ' Add args here
    ' send request
    m_call.Transfer m_Args
    Log "Sending Transfer request"
  End If
End Sub

Private Sub SSTransfer_Click()
  Log "SS Transfer Button clicked"
If m_call Is Nothing Then
  Log "no call - Transfer not possible"
Else
  ' Pack up arguments and pass to SingleStepTransfer
  Dim m_Args As New Arguments
  If 0 = Len(dialthis.Text) Then
    Log "Please enter Dialstring in Dial Box first"
  Else
    m_Args.AddItem "dialednumber", dialthis.Text
    If Not 0 = Len(callvar1.Text) Then
      ' set callvar1 if present
      m_Args.AddItem "Callvariable1", callvar1.Text
    End If
    ' send request
    m_call.SingleStepTransfer m_Args
    Log "Sending SS Transfer request DialedNumber =" + dialthis.Text
  End If
End If
End Sub

Private Sub GetData_Click()
  Dim args As Arguments
  Log "GetData button clicked"
  If m_call Is Nothing Then
    Log "No call current from which to get data."
  Else
    Set args = m_call.GetCallData
    LogEvent "GetCallData: ", args
  End If
End Sub

Private Sub SetData_Click()
  Dim nErrorCode As Long
  Dim args As New Arguments
  Dim ECCArgs As New Arguments
  Log "SetData button clicked"
  If m_call Is Nothing Then
Log "No call current to set data."

Else

' regular variables
args.AddItem "ANI", "MyANI"
args.AddItem "DNIS", "MyDNIS"
args.AddItem "CallerEnteredDigits", "MyCED"
args.AddItem "CallVariable1", "MyCallVariable1"

' ECC variables
ECCArgs.AddItem "Scalar1", "MyScalar1"
ECCArgs.AddItem "Scalar2", "MyScalar2"
ECCArgs.AddItem "Array1[1]", "FirstArrayValue"
ECCArgs.AddItem "Array1[2]", "SecondArrayValue"

' Add ECC arguments to args
args.AddItem "ECC", ECCArgs

m_call.SetCallData args, nErrorCode

LogEvent "SetData:", args
End If

Private Sub GetContext_Click()

Dim args As Arguments

Log "GetContext button clicked"

If m_call Is Nothing Then

Log "No call current from which to get context."

Else

Set args = m_call.GetCallContext

LogEvent "GetCallContext: ", args

End If

End Sub

' Handle OnCallBegin event

Private Sub m_session_OnCallBegin(ByVal pIArguments As CTIOSCLIENTLib.IArguments)

LogEvent "BeginCallEvent", pIArguments

' Collect unique callid
HandleNewCall pIArguments

' Display value of CallVariable1 from event
callvar1.Text = pIArguments.GetValueString("Callvariable1")

End Sub

' Routine which queries for the UniqueObjectID of a call given an event msg object
'
Private Sub HandleNewCall(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    ' compose string "UniqueObjectID=XXXXXX" to retrieve callpointer
    Dim m_uid As String
    m_uid = pIArguments.GetValueString("Uniqueobjectid")

    Dim tmp As String
    tmp = "UniqueObjectID=" + m_uid

    ' get callpointer and store it
    Set m_call = m_session.GetValue(tmp)

    ' display the new callid on the dialog
    calluid.Text = m_uid

End Sub

' Handle the OnCallConferenced event
'
Private Sub m_session_OnCallConferenced(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "CallConferenced", pIArguments
    HandleNewCall pIArguments

End Sub

' Handle the OnCallDelivered event
'
Private Sub m_session_OnCallDelivered(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "CallDeliveredEvent", pIArguments

    ' Automatically answer the call if the mode_text is set to B or C
    If "B" = mode_text.Text Or "C" = mode_text.Text Then
        m_call.Answer
    End If

End Sub

' Handle the OnCallEnd event
'
Private Sub m_session_OnCallEnd(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "CallEndEvent", pIArguments

End Sub
If Not m_call Is Nothing Then
    'Clear the call object
    Set m_call = Nothing
End If

' clear the display
calluid.Text = "no call"
End Sub

' Handle the OnCallEstablished event
',
Private Sub m_session_OnCallEstablished(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    Dim m_Args As New Arguments
    Dim m_callvar As Variant
    LogEvent "CallEstablishedEvent", pIArguments

    ' if the mode is set to B then perform an automatic transfer to whatever
    ' phone number is in the dialthis field. If Callvariable1 is present then
    ' increment its value before calling the SingleStepTransfer
    ',
    If "B" = mode_text.Text Then
        If m_call Is Nothing Then
            Log "no call - Transfer not possible"
        Else
            If 0 = Len(dialthis.Text) Then
                Log "Must supply dialed number for automatic transfer"
            Exit Sub
            End If
            m_Args.AddItem "dialednumber", dialthis.Text
            If Not 0 = Len(callvar1.Text) Then
                m_callvar = Format(Val(callvar1.Text) + 1)
                m_Args.AddItem "Callvariable1", m_callvar
            End If
            ' send request
            m_call.SingleStepTransfer m_Args
            Log "Sending Transfer request DialedNumber=\" + dialthis.Text + \"\" + "CallVar=\" + m_callvar + \"\"
        End If
    End If

    ' If the mode is set to C then this is the last leg of the call so just hang up
    If "C" = mode_text.Text Then
m_call.Clear
End If

' Handle the OnSetAgentModeEvent event
'
' This event happens in response to SetAgent. The application must wait for this
' event before trying to perform an agent login.
'
Private Sub m_session_OnSetAgentModeEvent(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "OnSetAgentModeEvent", pIArguments
    ' Pack up the arguments and perform an agent login
    Dim m_Args As New Arguments
    Dim nRetVal As Long
    m_Args.AddItem "Agentid", agentid.Text
    m_Args.AddItem "AgentPassword", agentpw.Text
    m_Args.AddItem "AgentInstrument", instrument.Text
    m_Args.AddItem "PeripheralID", peripheralid.Text

    ' send login request
    nRetVal = m_Agent.Login(m_Args)

    Log "Login returns(" + Str(nRetVal) + ")"
End Sub

' Handle the OnCallHeld event
'
Private Sub m_session_OnCallHeld(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "CallHeldEvent", pIArguments
End Sub

' Handle the OnCallOriginated event
'
Private Sub m_session_OnCallOriginated(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "CallOriginated", pIArguments
End Sub

' Handle the OnCallRetrieved event
'
Private Sub m_session_OnCallRetrieved(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "CallRetrieved", pIArguments
End Sub

' Handle the OnCallTransferConferenceInitiated event

Appendix C  Sample Visual Basic Application

Private Sub m_session_OnCallTransferConferenceInitiated(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "CallTransferConferenceInitiated", pIArguments
End Sub

' Handle the OnCallTransferred event
'
Private Sub m_session_OnCallTransferred(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "CallTransferred", pIArguments
End Sub

' Handle the OnConnection event - Connection attempt success
'
Private Sub m_session_OnConnection(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    ' Retrieve CurrentPort value and display on dialog
    Dim P As Long
    P = pIArguments.GetValueInt("CurrentPort")
    Log "Connected to port " + Str(P)

    LogEvent "OnConnection", pIArguments

    connectionstatus.Text = "Connected"
    m_bConnected = True

    'Enable Login button
    Login.Enabled = True

    Log "You can Login now!"
End Sub

' Handle the OnConnectionClosed event - Connection has been closed
'
Private Sub m_session_OnConnectionClosed(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "OnConnectionClosed", pIArguments

    connectionstatus.Text = "Not Connected"
    agentstate.Text = "Unknown"

    m_bConnected = False

    'Disable Login button
    Login.Enabled = False
End Sub

' Handle the OnConnectionFailure event - Connection attempt has failed
'
Private Sub m_session_OnConnectionFailure(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "OnConnectionFailure", pIArguments
connectionstatus.Text = "Not Connected"
agentstate.Text = "Unknown"

m_bConnected = False

Set m_Agent = Nothing
End Sub

' Handle the OnControlFailureConf event - A third party request (MakeCall, etc.) has failed
'
Private Sub m_session_OnControlFailureConf(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "ControlFailureConf", pIArguments
End Sub

' Handle the OnFailureEvent event - A non third party request has failed
'
Private Sub m_session_OnFailureEvent(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "OnFailure", pIArguments
End Sub

' Handle the OnHeartbeat event - A heartbeat request has been received
'
Private Sub m_session_OnHeartbeat(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "OnHeartbeat", pIArguments
End Sub

' Handle the OnMissingHeartbeat event - Heartbeats from the CTIOS Server have been missed
'
Private Sub m_session_OnMissingHeartbeat(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "OnMissingHeartbeat", pIArguments
End Sub

' Handle the OnQueryAgentStateConf event - This event happens as result of Login request as well as the result of a QueryAgentState
'
Private Sub m_session_OnQueryAgentStateConf(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "QueryAgentStateConf", pIArguments

    Dim newstate As Integer
    newstate = pIArguments.GetValueInt("AgentState")
    HandleAgentState newstate
End Sub

' Handle the OnAgentStateChange event
'
Appendix C  Sample Visual Basic Application

Private Sub m_session_OnAgentStateChange(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "AgentStateChange", pIArguments
    ' Retrieve new state and display it
    Dim newstate As Integer
    newstate = pIArguments.GetValueInt("AgentState")
    HandleAgentState newstate
End Sub

' Convert AgentState enum to text string and display it on the dialog
Private Sub HandleAgentState(newstate As Integer)
    Dim StateStr As String
    StateStr = "Unknown"
    Select Case newstate
        Case eLogin
            StateStr = "Login"
        Case eLogout
            StateStr = "Logout"
        Case eNotReady
            StateStr = "Not Ready"
        Case eAvailable
            StateStr = "Ready"
        Case eTalking
            StateStr = "Talking"
        Case eWorkReady
            StateStr = "Wrapup"
        Case eWorkNotReady
            StateStr = "Wrapup"
        Case eBusyOther
            StateStr = "BusyOther"
        Case eReserved
            StateStr = "Reserved"
        Case eUnknown
            StateStr = "Unknown"
        Case eHold
            StateStr = "Hold"
    End Select
    agentstate.Text = StateStr
End Sub

' Handle the OnSnapshotCallConf event - Sent by the CTIOS Server to inform the client
' about existing call status upon login

Private Sub m_session_OnSnapshotCallConf(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "OnSnapshotCallConf", pIArguments
    HandleNewCall pIArguments
End Sub

' Handle the OnButtonEnablementChange event
'
' Enable/Disable buttons based on EnablementMask
'
Private Sub m_session_OnButtonEnablementChange(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    LogEvent "OnButtonEnablementChange", pIArguments

    Dim eMask As Long
    eMask = pIArguments.GetValueInt("EnablementMask")
    If eMask And ENABLE_ANSWER Then
        Answer.Enabled = True
    Else
        Answer.Enabled = False
    End If
    If eMask And ENABLE_CONFERENCE_COMPLETE Then
        Conference.Enabled = True
    Else
        Conference.Enabled = False
    End If
    If eMask And ENABLE_CONFERENCE_INIT Then
        CCConference.Enabled = True
    Else
        CCConference.Enabled = False
    End If
    If eMask And ENABLE_HOLD Then
        Hold.Enabled = True
    Else
        Hold.Enabled = False
    End If
    If eMask And ENABLE_LOGIN Then
        Login.Enabled = True
    Else
        Login.Enabled = False
    End If
If eMask And ENABLE_LOGOUT Then
   Logout.Enabled = True
Else
   Logout.Enabled = False
End If

If eMask And ENABLE_MAKECALL Then
   MakeCall.Enabled = True
Else
   MakeCall.Enabled = False
End If

If eMask And ENABLE_NOTREADY Then
   NotReady.Enabled = True
Else
   NotReady.Enabled = False
End If

If eMask And ENABLE_READY Then
   Ready.Enabled = True
Else
   Ready.Enabled = False
End If

If eMask And ENABLE_RELEASE Then
   Release.Enabled = True
Else
   Release.Enabled = False
End If

If eMask And ENABLE_RETRIEVE Then
   Retrieve.Enabled = True
Else
   Retrieve.Enabled = False
End If

If eMask And ENABLE_SINGLE_STEP_CONFERENCE Then
   SSConference.Enabled = True
Else
   SSConference.Enabled = False
End If

If eMask And ENABLE_SINGLE_STEP_TRANSFER Then
   SSTransfer.Enabled = True
Else
   SSTransfer.Enabled = False
End If
If eMask And ENABLE_TRANSFER_COMPLETE Then
    Transfer.Enabled = True
Else
    Transfer.Enabled = False
End If

If eMask And ENABLE_TRANSFER_INIT Then
    CCTransfer.Enabled = True
Else
    CCTransfer.Enabled = False
End If

End Sub

' Generic routing which packs up the arguments and calls SetAgentState
',
Private Sub HandleSetAgentStateRequest(newstate As Integer)
    Dim m_Args As New Arguments
    Dim state As Integer
    state = newstate
    m_Args.AddItem "AgentState", state
    m_Args.AddItem "ReasonCode", 1 ' add a default reasoncode

    ' send request
    m_Agent.SetAgentState m_Args

    Log "Sending SetAgentState Request " + Str(newstate)
End Sub

' Format and display an events contents to the log window
',
Private Sub LogEvent(txt As String, ByVal pIArguments As CTIOSCLIENTLib.IArguments)
    ' logs the event and its arguments to the logwindow
    Dim m_txt As String
    m_txt = txt + " Event received"
    Log m_txt

    Dim S As String
    If 1 = check_details Then
        ' Call the event objects DumpArgs function to format the event data
        S = " " + pIArguments.DumpArgs
        Log S
    End If
End Sub

' this routine is a helper function that writes a log entry to the logwindow
',
Private Sub Log(txt As String)
    ' write logentry with carriage return (13) linefeed (10)
    logwindow.Text = logwindow.Text + Chr(13) + Chr(10) + txt

    ' set selection to the end so the window scrolls down
    logwindow.SelStart = Len(logwindow.Text)
End Sub

' Clear the text from the log window
'
Private Sub clearlog_Click()
    logwindow.Text = ""
End Sub
Sample COM for C++ Application

The sample application in this appendix demonstrates the use of the COM dll in a C++ application. The picture of the workspace in Visual Studio (Figure D-1) gives a quick overview of the files that comprise the sample application.

Figure D-1   Sample COM Application Visual Basic Workspace
Figure D-2 shows a view of the main screen of the application.

Figure D-2  COM for C++ Application Main Screen

```
#define AFX_EVENTSINK_H__99E5509A_4F70_4B34_826F_23F014D16A44__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "EventSink.h"

```

class CExercise3Dlg;

 //////////////////////////////////////////////////////////////////////////
 // CEventSink command target
 class CEventSink : public CCmdTarget
 { 
 DECLARE_DYNCREATE(CEventSink)
 CEventSink(); // protected constructor used by dynamic creation
 virtual ~CEventSink();
 
 // Attributes
 public:
 // Operations
 public:
 // Overrides
 // ClassWizard generated virtual function overrides
 ////{{AFX_VIRTUAL(CEventSink)
 public:
 virtual void OnFinalRelease();
 ////}}AFX_VIRTUAL
 
 // Implementation
 public:
 // Generated message map functions
 ////{{AFX_MSG(CEventSink)
 // NOTE - the ClassWizard will add and remove member functions here.
 ////}}AFX_MSG
 DECLARE_MESSAGE_MAP()
 DECLARE_DISPATCH_MAP()
 DECLARE_INTERFACE_MAP()
 public:
 void SetSessionPtr(CTIOSCLIENTLib::ISessionPtr pSession) { m_pSession = pSession; }
 void SetAgentPtr(CTIOSCLIENTLib::IAgentPtr pAgent) { m_pAgent = pAgent; }
 void SetDialogPtr(CExercise3Dlg *pDialog) { m_pDialog = pDialog; }
// Session Events
void OnConnection(CTIOSCLIENTLib::IArguments *pIArguments);
void OnConnectionFailure(CTIOSCLIENTLib::IArguments *pIArguments);
void OnConnectionClosed(CTIOSCLIENTLib::IArguments *pIArguments);
void OnSetAgentModeEvent(CTIOSCLIENTLib::IArguments *pIArguments);

// Agent Events
void OnAgentStateChange(CTIOSCLIENTLib::IArguments *pIArguments);
void OnQueryAgentStateConf(CTIOSCLIENTLib::IArguments *pIArguments);

// Call Events
void OnCallBegin(CTIOSCLIENTLib::IArguments *pIArguments);
void OnCallDelivered(CTIOSCLIENTLib::IArguments *pIArguments);
void OnCallEstablished(CTIOSCLIENTLib::IArguments *pIArguments);
void OnCallEnd(CTIOSCLIENTLib::IArguments *pIArguments);
void OnCallConferenced(CTIOSCLIENTLib::IArguments *pIArguments);
void OnCallTransferred(CTIOSCLIENTLib::IArguments *pIArguments);

// Misc Events
void OnButtonEnablementChange(CTIOSCLIENTLib::IArguments *pIArguments);

CTIOSCLIENTLib::ISessionPtr m_pSession;// Session object ptr
CTIOSCLIENTLib::IAgentPtr m_pAgent;// Agent object ptr
CExercise3Dlg* m_pDialog; // dialog ptr

/*********************** End "EventSink.h" ******************************/

/*********************** Start "EventSink.cpp" ******************************

// EventSink.cpp : implementation file

#include "stdafx.h"
#include "exercise3.h"
#include "exercise3Dlg.h"
#include "EventSink.h"
#include <atlbase.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////
// CEventSink
IMPLEMENT_DYNCREATE(CEventSink, CCmdTarget)

CEventSink::CEventSink()
{
    EnableAutomation();
}

CEventSink::~CEventSink()
{
}

BEGIN_MESSAGE_MAP(CEventSink, CCmdTarget)
//{{AFX_MSG_MAP(CEventSink)
    // NOTE - the ClassWizard will add and remove mapping macros here.
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_DISPATCH_MAP(CEventSink, CCmdTarget)
//{{AFX_DISPATCH_MAP(CEventSink)
// let's use DISP_FUNCTION_ID instead DISP_FUNCTION so we can implement random
// event handlers
    // SEE HOWTO: Create a Sink Interface in MFC-Based COM Client
    // KNOWLEDGE BASE ID: Q181845
    DISP_FUNCTION_ID(CEventSink, "OnConnection", 0x1, OnConnection, VT_EMPTY, VTS_DISPATCH)
    DISP_FUNCTION_ID(CEventSink, "OnConnectionFailure", 0x2, OnConnectionFailure, VT_EMPTY, VTS_DISPATCH)
    DISP_FUNCTION_ID(CEventSink, "OnConnectionClosed", 0x27, OnConnectionClosed, VT_EMPTY, VTS_DISPATCH)
    DISP_FUNCTION_ID(CEventSink, "OnSetAgentModeEvent", 0x2C, OnSetAgentModeEvent, VT_EMPTY, VTS_DISPATCH)
    DISP_FUNCTION_ID(CEventSink, "OnAgentStateChange", 0x8, OnAgentStateChange, VT_EMPTY, VTS_DISPATCH)
    DISP_FUNCTION_ID(CEventSink, "OnQueryAgentStateConf", 0xCE, OnQueryAgentStateConf, VT_EMPTY, VTS_DISPATCH)
//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
DISP_FUNCTION_ID(CEventSink, "OnCallBegin", 0xC, OnCallBegin, VT_EMPTY, VTS_DISPATCH)
DISP_FUNCTION_ID(CEventSink, "OnCallDelivered", 0xF, OnCallDelivered, VT_EMPTY, VTS_DISPATCH)
DISP_FUNCTION_ID(CEventSink, "OnCallEstablished", 0x10, OnCallEstablished, VT_EMPTY, VTS_DISPATCH)
DISP_FUNCTION_ID(CEventSink, "OnCallEnd", 0xD, OnCallEnd, VT_EMPTY, VTS_DISPATCH)
DISP_FUNCTION_ID(CEventSink, "OnCallConferenced", 0x1A, OnCallConferenced, VT_EMPTY, VTS_DISPATCH)
DISP_FUNCTION_ID(CEventSink, "OnCallTransferred", 0x19, OnCallTransferred, VT_EMPTY, VTS_DISPATCH)
DISP_FUNCTION_ID(CEventSink, "OnButtonEnablementChange", 0x25, OnButtonEnablementChange, VT_EMPTY, VTS_DISPATCH)
//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()

BEGIN_INTERFACE_MAP(CEventSink, CCmdTarget)
    INTERFACE_PART(CEventSink, __uuidof(_IAllEvents), Dispatch)
END_INTERFACE_MAP()

void CEventSink::OnFinalRelease()
{
    // TODO: Add your specialized code here and/or call the base class
    CCmdTarget::OnFinalRelease();
}

void CEventSink::OnConnection(CTIOSCLIENTLib::IArguments *pIArguments)
{
    m_pDialog->LogEvent("OnConnection", pIArguments);
    m_pDialog->Log("You can Login now!");
    m_pDialog->m_edtConnection.SetWindowText("Connected");
    m_pDialog->SetSessionConnected(TRUE);
m_pDialog->m_btnLogin.EnableWindow(TRUE);
}

// Handle the OnConnectionFailure Event - Connect to CTIOS Server failed
//
void CEventSink::OnConnectionFailure(CTIOSCLIENTLib::IArguments *pIArguments)
{
    m_pDialog->LogEvent("OnConnectionFailure", pIArguments);
    m_pDialog->m_btnLogin.EnableWindow(FALSE);
}

// Handle the OnConnectionClosed Event - Connection to CTIOS Server has been closed
//
void CEventSink::OnConnectionClosed(CTIOSCLIENTLib::IArguments *pIArguments)
{
    m_pDialog->LogEvent("OnConnectionClosed", pIArguments);
    m_pDialog->m_edtConnection.SetWindowText("Offline");
    m_pDialog->SetSessionConnected(FALSE);
    m_pDialog->m_btnLogin.EnableWindow(FALSE);
}

// Handle the OnSetAgentMode Event - Result of SetAgent request
//
// Applications must wait for the OnSetAgentMode event before attempting an agent login.
//
void CEventSink::OnSetAgentModeEvent(CTIOSCLIENTLib::IArguments *pIArguments)
{
    HRESULT hRes;
    CString agentID;
    CString password;
    CString instrument;
    CString peripheralID;

    m_pDialog->LogEvent("OnSetAgentMode", pIArguments);

    // get agent info from dialog
    m_pDialog->m_edtAgentID.GetWindowText(agentID);
    m_pDialog->m_edtPassword.GetWindowText(password);
    m_pDialog->m_edtInstrument.GetWindowText(instrument);
    m_pDialog->m_edtPeripheralID.GetWindowText(peripheralID);
// add login request arguments
CTIOSCLIENTLib::IArgumentsPtr pArgs;
hRes = pArgs.CreateInstance("CtiOsComArguments.ComArguments");
pArgs->AddItem(_variant_t("AgentID"), _variant_t(agentID.GetBuffer(0)));
pArgs->AddItem(_variant_t("AgentPassword"), _variant_t(password.GetBuffer(0)));
pArgs->AddItem(_variant_t("AgentInstrument"),
    _variant_t(instrument.GetBuffer(0)));
pArgs->AddItem(_variant_t("PeripheralID"),
    _variant_t(peripheralID.GetBuffer(0)));

m_pDialog->AddLogMessage("Requesting Agent Login");

// perform Login
m_pAgent->Login(pArgs);
}

// Agent Events

// Handle the OnAgentStateChange Event - new agent state from CTIOS Server
void CEventSink::OnAgentStateChange(CTIOSCLIENTLib::IArguments *pIArguments)
{
    m_pDialog->LogEvent("OnAgentStateChange", pIArguments);

    // Retrieve new agent state from argument list and display it on the dialog
    long state = pIArguments->GetValueInt(_variant_t("AgentState"));
    m_pDialog->SetAgentState((enumCTIOS_AgentState) state);
}

// Handle the OnQueryAgentStateChangeConf Event - received as a result of an agent Login
// This will provide the agent's current state
void CEventSink::OnQueryAgentStateConf(CTIOSCLIENTLib::IArguments *pIArguments)
{
    m_pDialog->LogEvent("OnQueryAgentStateConf", pIArguments);

    // Display agent state value from Agent object (Could also get state from pDispatch)
    m_pDialog->SetAgentState((enumCTIOS_AgentState) m_pAgent->GetAgentState());
}
//
// Call Events
//
// Handle the OnCallBegin Event - New call has started
void CEventSink::OnCallBegin(CTIOSCLIENTLib::IArguments *pIArguments)
{
    _bstr_t cv1("");
    m_pDialog->LogEvent("OnBeginCall", pIArguments);

    // Retrieve new agent state from argument list and display it on the dialog
    long state = pIArguments->GetValueInt(&_variant_t("AgentState"));

    // Check if CV1 is present in the message
    if (pIArguments->IsValid(&_variant_t("CALLVARIABLE1")) == VARIANT_TRUE)
    {
        // Get Call Variable 1 from message
        _variant_t tmp = pIArguments->GetValueString(&_variant_t("CALLVARIABLE1"));

        // convert to BSTR
        cv1 = (_bstr_t) tmp;
    }

    // Display Call Variable1 on dialog
    m_pDialog->m_edtCallVariable1.SetWindowText((char *) cv1);

    // Check for UniqueObjectID presence in message
    if (pIArguments->IsValid(&_variant_t("UNIQUEOBJECTID")) == VARIANT_TRUE)
    {
        // Get CallID from message
        _variant_t tmp = pIArguments->GetValueString(&_variant_t("UNIQUEOBJECTID"));

        // convert to BSTR
        _bstr_t callID = (_bstr_t) tmp;

        // Display CallID on dialog
        m_pDialog->m_edtCallID.SetWindowText((char *) callID);
    }
}

//
// Handle the OnCallDelivered Event - Call is now delivered/alerting at the agent phone
//
void CEventSink::OnCallDelivered(CTIOSCLIENTLib::IArguments *pIArguments)
{
    m_pDialog->LogEvent("OnCallDelivered", pIArguments);

    // get mode setting from dialog
    CString mode;
    m_pDialog->m_edtMode.GetWindowText(mode);

    // If mode is B or C then automatically answer the call
    if (strcmp(mode, "B") == 0 || strcmp(mode, "C") == 0)
    {
        // Check for UniqueObjectID presence in message
        if (pIArguments->IsValid(&_variant_t("UNIQUEOBJECTID")) == VARIANT_TRUE)
        {
            // Get CallID from message
            _variant_t callID = pIArguments->GetValueString(&_variant_t("UNIQUEOBJECTID"));

            // convert to BSTR and append to key name
            _bstr_t key = "UniqueObjectID=";
            key += (_bstr_t) callID;

            m_pDialog->AddLogMessage((char *) key);

            // Get Call ptr from Session object
            CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t(key));

            if (pCall)
            {
                // Create arguments object
                CTIOSCLIENTLib::IArgumentsPtr pArgs;
                HRESULT hRes = pArgs.CreateInstance("CtiOsComArguments.ComArguments");

                // No actual arguments needed for Answer
                pCall->Answer(pArgs);
            }
        }
    }
}

// Handle the OnCallEstablished Event - call has been answered
// void CEventSink::OnCallEstablished(CTIOSCLIENTLib::IArguments *pIArguments)
m_pDialog->LogEvent("OnCallEstablished", pIArguments);

// get mode from dialog
CString mode;
m_pDialog->m_edtMode.GetWindowText(mode);

// if mode is B then increment call variable 1 and pass as part of
// automatic transfer to agent C
if (strcmp(mode, "B") == 0)
{
    // Increment Call Variable value
    CString cv1;
m_pDialog->m_edtCallVariable1.GetWindowText(cv1);

    int cv_value = atoi(cv1.GetBuffer(0));
    cv_value++;
    cv1.Format("%d", cv_value);

    // Get Phone number to transfer to
    CString dn;
m_pDialog->m_edtPhoneNumber.GetWindowText(dn);

    if (dn.IsEmpty() == TRUE)
    {
        m_pDialog->AddLogMessage("Must fill in phone number field for automatic transfer");
        return;
    }

    // Create arg array for transfer request
    CTIOSCLIENTLib::IArgumentsPtr pArgs;
    HRESULT hRes = pArgs.CreateInstance("CtiOsComArguments.ComArguments");
    pArgs->AddItem(&_variant_t("DIALEDNUMBER"), &_variant_t(dn.GetBuffer(0)));
    pArgs->AddItem(&_variant_t("CALLVARIABLE1"),
                   &_variant_t(cv1.GetBuffer(0)));

    // Get pointer to call object and initiate the transfer
    CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t("CurrentCall"));

    if (pCall)
    {
        pCall->SingleStepTransfer(pArgs);
    }
}
// if mode is C then simply release the call as we are finished
else if (strcmp(mode, "C") == 0)
{
    // Get pointer to call object and initiate the transfer
    CTIOSCLIENTLib::ICallPtr pCall =
        m_pSession->GetValue(&_variant_t("CurrentCall"));

    if (pCall)
    {
        // Create arguments object
        CTIOSCLIENTLib::IArgumentsPtr pArgs;
        HRESULT hRes =
            pArgs.CreateInstance("CtiOsComArguments.ComArguments");

        // No actual arguments needed for Clear
        pCall->Clear(pArgs);
    }
}

// Handle the OnCallEnd Event - Call is officially ended
void CEventSink::OnCallEnd(CTIOSCLIENTLib::IArguments *pIArguments)
{
    m_pDialog->LogEvent("OnCallEnd", pIArguments);
    m_pDialog->m_edtCallID.SetWindowText("no call");
}

// Handle the OnCallConferenced Event - call has been conferenced
void CEventSink::OnCallConferenced(CTIOSCLIENTLib::IArguments *pIArguments)
{
    m_pDialog->LogEvent("OnCallConferenced", pIArguments);

    // Check for UniqueObjectID presence in message
    if (pIArguments->IsValid(&_variant_t("UNIQUEOBJECTID")) == VARIANT_TRUE)
    {
        // Get CallID from message
        _variant_t callID =
            pIArguments->GetValueString(&_variant_t("UNIQUEOBJECTID"));
// Display Call Variable1 on dialog
m_pDialog->m_edtCallID.SetWindowText((char *) (_bstr_t) callID);
}

// Handle the OnCallTransferred Event - call has been transferred
void CEventSink::OnCallTransferred(CTIOSCLIENTLib::IArguments *pIArguments)
{
    m_pDialog->LogEvent("OnCallTransferred", pIArguments);

    // Check for UniqueObjectID presence in message
    if (pIArguments->IsValid(&_variant_t("UNIQUEOBJECTID")) == VARIANT_TRUE)
    {
        // Get CallID from message
        _variant_t callID = pIArguments->GetValueString(&_variant_t("UNIQUEOBJECTID"));

        // Display Call Variable1 on dialog
        m_pDialog->m_edtCallID.SetWindowText((char *) (_bstr_t) callID);
    }
}

// Other Events

// Handle the OnButtonEnablementChange Event
// CTIOS Server sends bitmask with bits set indicating which buttons should be enabled.
void CEventSink::OnButtonEnablementChange(CTIOSCLIENTLib::IArguments *pIArguments)
{
    m_pDialog->LogEvent("OnButtonEnablementChange", pIArguments);

    // Get mask from message
    long eMask = pIArguments->GetValueInt(&_variant_t("ENABLEMENTMASK"));

    if (eMask & ENABLE_ANSWER)
    {
        m_pDialog->m_btnAnswer.EnableWindow(TRUE);
    }
    else
    {
        m_pDialog->m_btnAnswer.EnableWindow(FALSE);
if (eMask & ENABLE_CONFERENCE_COMPLETE) {
    m_pDialog->m_btnConference.EnableWindow(TRUE);
} else {
    m_pDialog->m_btnConference.EnableWindow(FALSE);
}

if (eMask & ENABLE_CONFERENCE_INIT) {
    m_pDialog->m_btnCCConference.EnableWindow(TRUE);
} else {
    m_pDialog->m_btnCCConference.EnableWindow(FALSE);
}

if (eMask & ENABLE_HOLD) {
    m_pDialog->m_btnHold.EnableWindow(TRUE);
} else {
    m_pDialog->m_btnHold.EnableWindow(FALSE);
}

if (eMask & ENABLE_LOGIN) {
    m_pDialog->m_btnLogin.EnableWindow(TRUE);
} else {
    m_pDialog->m_btnLogin.EnableWindow(FALSE);
}

if (eMask & ENABLE_LOGOUT) {
    m_pDialog->m_btnLogout.EnableWindow(TRUE);
} else {
    m_pDialog->m_btnLogout.EnableWindow(FALSE);
}

if (eMask & ENABLE_MAKECALL)
{ 
    m_pDialog->m_btnMakeCall.EnableWindow(TRUE);
} 
else 
{ 
    m_pDialog->m_btnMakeCall.EnableWindow(FALSE);
}

if (eMask & ENABLE_NOTREADY) 
{ 
    m_pDialog->m_btnNotReady.EnableWindow(TRUE);
} 
else 
{ 
    m_pDialog->m_btnNotReady.EnableWindow(FALSE);
}

if (eMask & ENABLE_READY) 
{ 
    m_pDialog->m_btnReady.EnableWindow(TRUE);
} 
else 
{ 
    m_pDialog->m_btnReady.EnableWindow(FALSE);
}

if (eMask & ENABLE_RELEASE) 
{ 
    m_pDialog->m_btnRelease.EnableWindow(TRUE);
} 
else 
{ 
    m_pDialog->m_btnRelease.EnableWindow(FALSE);
}

if (eMask & ENABLE_RETRIEVE) 
{ 
    m_pDialog->m_btnRetrieve.EnableWindow(TRUE);
} 
else 
{ 
    m_pDialog->m_btnRetrieve.EnableWindow(FALSE);
}

if (eMask & ENABLE_SINGLE_STEP_CONFERENCE) 
{ 
    m_pDialog->m_btnSSConference.EnableWindow(TRUE);
}
else
{
  m_pDialog->m_btnSSConference.EnableWindow(FALSE);
}

if (eMask & ENABLE_SINGLE_STEP_TRANSFER)
{
  m_pDialog->m_btnSSTransfer.EnableWindow(TRUE);
} else
{
  m_pDialog->m_btnSSTransfer.EnableWindow(FALSE);
}

if (eMask & ENABLE_TRANSFER_COMPLETE)
{
  m_pDialog->m_btnTransfer.EnableWindow(TRUE);
} else
{
  m_pDialog->m_btnTransfer.EnableWindow(FALSE);
}

if (eMask & ENABLE_TRANSFER_INIT)
{
  m_pDialog->m_btnCCTransfer.EnableWindow(TRUE);
} else
{
  m_pDialog->m_btnCCTransfer.EnableWindow(FALSE);
}

Appendix D  Sample COM for C++ Application

#include "resource.h"// main symbols

////////////////////////////////////////////////////////////////////////
// CExercise3App:
// See exercise3.cpp for the implementation of this class
////////////////////////////////////////////////////////////////////////
class CExercise3App : public CWinApp
{
public:
    CExercise3App();

// Overrides
    // ClassWizard generated virtual function overrides
    public:
    virtual BOOL InitInstance();

// Implementation
    // NOTE - the ClassWizard will add and remove member functions here.
    //    DO NOT EDIT what you see in these blocks of generated code!
    DECLAREMESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////

// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
#endif // !defined(AFX_EXERCISE3_H__8A0189AD_B3F8_47CD_BA3D_D96F10C9DD2A__INCLUDED_)

#endif // !defined(AFX_EXERCISE3_H__8A0189AD_B3F8_47CD_BA3D_D96F10C9DD2A__INCLUDED_)

-------------------- End "exercise3.h" --------------------------

-------------------- Start "exercise3.cpp" --------------------------
// exercise3.cpp : Defines the class behaviors for the application.
#
#include "stdafx.h"
Appendix D

Sample COM for C++ Application

#include "exercise3.h"
#include "exercise3Dlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CExercise3App
BEGIN_MESSAGE_MAP(CExercise3App, CWinApp)
//{{AFX_MSG_MAP(CExercise3App)
// NOTE - the ClassWizard will add and remove mapping macros here.
//
DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG
ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////////
// CExercise3App construction
CExercise3App::CExercise3App()
{
// TODO: add construction code here,
// Place all significant initialization in InitInstance
}
/////////////////////////////////////////////////////////////////////////////
// The one and only CExercise3App object
CExercise3App theApp;
/////////////////////////////////////////////////////////////////////////////
// CExercise3App initialization
BOOL CExercise3App::InitInstance()
{
AfxEnableControlContainer();
// Standard initialization
// If you are not using these features and wish to reduce the size
// of your final executable, you should remove from the following
// the specific initialization routines you do not need.
#ifdef _AFXDLL
Enable3dControls();

// Call this when using MFC in a shared DLL

Cisco ICM Software CTI OS Developer’s Guide

D-18

OL-1392-03


#else
    Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif

    CoInitialize(NULL); // Enable COM

    CExercise3Dlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the application, rather than start the application's message pump.
    return FALSE;

}  // End CExercise3Dlg

========================= End "exercise3.cpp" ===============================

========================= Start "exercise3Dlg.h" ============================

// exercise3Dlg.h : header file

#ifndef AFX_EXERCISE3DLG_H__A89B260A_2BBE_49DC_B6CD_132BD0D00C0F__INCLUDED_
#define AFX_EXERCISE3DLG_H__A89B260A_2BBE_49DC_B6CD_132BD0D00C0F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "EventSink.h"

#include "CExercise3Dlg.h"

class CExercise3Dlg : public CDialog
{
public:

    // Construction

CExercise3Dlg(CWnd* pParent = NULL); // standard constructor

void AddLogMessage(char *pMessage) { Log(pMessage); }
void LogEvent(const char *msg, CTIOSCLIENTLib::IArguments *pIArguments);
void Log(const char *newText);

void SetSessionConnected(bool bConnected) { m_bSessionConnected = bConnected; }
void SetAgentState(enum CTIOS_AgentState newstate) {
m_edtAgentState.SetWindowText(AgentStateToString(newstate)); }

char *AgentStateToString(enum CTIOS_AgentState eState);

// Dialog Data
//}}AFX_DATA(CExercise3Dlg)
enum { IDD = IDD_EXERCISE3_DIALOG };
CEdit m_edtHostA;
CEdit m_edtMode;
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//}}AFX_VIRTUAL(CExercise3Dlg)
protected:
void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
CEventSink m_EventSink; // Event Sink
DWORD m_dwEventSinkAdvise;

CTIOSSESSIONRESOLVERLib::ISessionResolverPtr m_pSessionResolver;
CTIOSCLIENTLib::ISessionPtr m_pSession; // Session object ptr
CTIOSCLIENTLib::IAgentPtr m_pAgent; // Agent object ptr
bool m_bSessionConnected; // Connection status indicator

HICON m_hIcon;

// Generated message map functions
//}}AFX_MSG(CExercise3Dlg)
virtual BOOL OnInitDialog();
afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
afx_msg void OnConnect();
afx_msg void OnDisconnect();
afx_msg void OnLogin();
afx_msg void OnLogout();
afx_msg void OnReady();
afx_msg void OnNotready();
afx_msg void OnMakecall();
afx_msg void OnAnswer();
afx_msg void OnRelease();
afx_msg void OnRetrieve();
afx_msg void OnTransfer();
afx_msg void OnHold();
afx_msg void OnConference();
afx_msg void OnClearlog();
afx_msg void OnClose();
afx_msg void OnCCTransfer();
afx_msg void OnGetContext();
afx_msg void OnGetData();
afx_msg void OnSetData();
afx_msg void OnSSConference();
afx_msg void OnSSTransfer();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

//}}AFX_INSERT_LOCATION}
// Microsoft Visual C++ will insert additional declarations immediately before
// the previous line.

#endif // !defined(AFX_EXERCISE3DLG_H__A89B260A_2BBE_49DC_86CD_132BD0D00CFE__INCLUDED_)

End "exercise3Dlg.h"  

Start "exercise3Dlg.cpp"  

// exercise3Dlg.cpp : implementation file

#include "stdafx.h"
#include "exercise3.h"
#include "exercise3Dlg.h"
#include <atlbase.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

static char THIS_FILE[] = __FILE__;

// nRetVal success value

#define CIL_OK1

// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
//}}AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
};
Appendix D  Sample COM for C++ Application

//}}AFX_DATA

// ClassWizard generated virtual function overrides
//}}AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
//{{AFX_MSG(CAboutDlg)
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
//{{AFX_DATA_INIT(CAboutDlg)
//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CAboutDlg)
//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
//{{AFX_MSG_MAP(CAboutDlg)
// No message handlers
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

void CExercise3Dlg::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//}}AFX_DATA_MAP
}

// CExercise3Dlg::CExercise3Dlg(CWnd* pParent /*=NULL*/) 
CExercise3Dlg::CExercise3Dlg(CWnd* pParent /*=NULL*/) 
: CDialog(CExercise3Dlg::IDD, pParent)
{
//{{AFX_DATA_INIT(CExercise3Dlg)
//}}AFX_DATA_INIT
// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CExercise3Dlg::DoDataExchange(CDataExchange* pDX)
{
CDIalog: :DoDataExchange(pDX);  
//\{AFX_DATA_MAP (CExercise3Dlg)
DDX_Control(pDX, IDC_CONFERENCE, m_btnCCConference);
DDX_Control(pDX, IDCTRANSFER, m_btnCCTransfer);
DDX_Control(pDX, IDGETCONTEXT, m_btnGetContext);
DDX_Control(pDX, IDGETDATA, m_btnGetData);
DDX_Control(pDX, IDSETDATA, m_btnSetData);
DDX_Control(pDX, IDSSCONFERENCE, m_btnSSConference);
DDX_Control(pDX, IDSTTRANSFER, m_btnSSTransfer);
DDX_Control(pDX, IDTRANSFER, m_btnTransfer);
DDX_Control(pDX, IDRETRIEVE, m_btnRetrieve);
DDX_Control(pDX, IDRELEASE, m_btnRelease);
DDX_Control(pDX, IDREADY, m_btnReady);
DDX_Control(pDX, IDNOTREADY, m_btnNotReady);
DDX_Control(pDX, IDMAKECALL, m_btnMakeCall);
DDX_Control(pDX, IDLOGOUT, m_btnLogout);
DDX_Control(pDX, IDLOGIN, m_btnLogin);
DDX_Control(pDX, IDHOLD, m_btnHold);
DDX_Control(pDX, IDDISCONNECT, m_btnDisconnect);
DDX_Control(pDX, IDC_DISCONNECT, m_btnDisconnect);
DDX_Control(pDX, IDC_CONNECTION, m_btnConnell);
DDX_Control(pDX, IDC_AGENTSTATE, m_btnAgentState);
DDX_Control(pDX, IDC_AGENTID, m_btnAgentID);
DDX_Control(pDX, IDC_CALLVARIABLE1, m_btnCallVariable1);
DDX_Control(pDX, IDC_INSTRUMENT, m_btnInstrument);
DDX_Control(pDX, IDC_PASSWORD, m_btnPassword);
DDX_Control(pDX, IDC_PERIPHERALID, m_btnPeripheralID);
DDX_Control(pDX, IDC_PHONEDESCRIPTION, m_btnPhoneDescription);
DDX_Control(pDX, IDC_PORTA, m_btnPortA);
DDX_Control(pDX, IDC_PORTB, m_btnPortB);
DDX_Control(pDX, IDC_HOSTA, m_btnHostA);
DDX_Control(pDX, IDC_HOSTB, m_btnHostB);
DDX_Control(pDX, IDC_MODE, m_btnMode);
//\}AFX_DATA_MAP

BEGIN_MESSAGE_MAP(CExercise3Dlg, CDialog)
//\{AFX_MSG_MAP (CExercise3Dlg)
ON_WM_SYSCOMMAND()  
ON_WM_PAINT()  
ON_WM_QUERYDRAGICON()  
ON_BN_CLICKED (IDCONNECT, OnConnect)  
ON_BN_CLICKED (IDDISCONNECT, OnDisconnect)  
ON_BN_CLICKED (IDLOGIN, OnLogin) 
ON_WM_DATAVALUE()
ON_BN_CLICKED(IDLOGOUT, OnLogout)
ON_BN_CLICKED(IDREADY, OnReady)
ON_BN_CLICKED(IDNOTREADY, OnNotready)
ON_BN_CLICKED(IDMAKECALL, OnMakecall)
ON_BN_CLICKED(IDANSWER, OnAnswer)
ON_BN_CLICKED(IDRELEASE, OnRelease)
ON_BN_CLICKED(IDRETRIEVE, OnRetrieve)
ON_BN_CLICKED(IDTRANSFER, OnTransfer)
ON_BN_CLICKED(IDHOLD, OnHold)
ON_BN_CLICKED(IDCONFERENCE, OnConference)
ON_BN_CLICKED(IDCLEARLOG, OnClearlog)
ON_WM_CLOSE()
ON_BN_CLICKED(IDCCTCONFERENCE, OnCCConference)
ON_BN_CLICKED(IDCCTRANSFER, OnCCTransfer)
ON_BN_CLICKED(IDGETCONTEXT, OnGetContext)
ON_BN_CLICKED(IDGETDATA, OnGetData)
ON_BN_CLICKED(IDSETDATA, OnSetData)
ON_BN_CLICKED(IDSSCONFERENCE, OnSSConference)
ON_BN_CLICKED(IDSSTRANSFER, OnSSTransfer)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
// CExercise3Dlg message handlers

BOOL CExercise3Dlg::OnInitDialog()
{
    HRESULT hRes;
    CDialog::OnInitDialog();
    // Add "About..." menu item to system menu.
    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenuIsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }
// Set the icon for this dialog. The framework does this automatically
// when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE); // Set big icon
SetIcon(m_hIcon, FALSE); // Set small icon

// Create SessionResolver and Session object
hRes = m_pSessionResolver.CreateInstance(OLESTR("CTIOSSessionResolver.SessionResolver"));

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

m_pAgent.CreateInstance("CTIOSClient.ComAgent");

// Initialize the event sink
m_EventSink.SetSessionPtr(m_pSession);
m_EventSink.SetAgentPtr(m_pAgent);
m_EventSink.SetDialogPtr(this);

// Add event sink as event listener
hRes = AtlAdvise(m_pSession, m_EventSink.GetIDispatch(FALSE), __uuidof(_IAllEvents), &m_dwEventSinkAdvise);

if (FAILED(hRes))
{
    char buffer[100];
sprintf(buffer, "AtlAdvise failed, hRes=(%d)", hRes);
    MessageBox(buffer, "Event Sink Error");
    ASSERT(FALSE);
}

// initialize edit boxes
m_edtConnection.SetWindowText("Offline");
m_edtHostA.SetWindowText("localhost");
m_edtPortA.SetWindowText("42028");
m_edtHostB.SetWindowText("localhost");
m_edtPortB.SetWindowText("42028");
m_edtAgentID.SetWindowText("5000");
m_edtPassword.SetWindowText("5000");
m_edtInstrument.SetWindowText("5000");
m_edtPeripheralID.SetWindowText("5000");
Appendix D      Sample COM for C++ Application

m_edtCallID.SetWindowText("no call");
m_edtMode.SetWindowText("A, B, or C");
SetSessionConnected(FALSE);
SetAgentState(eUnknown);

Log("Welcome to the VC++ CTIOS COM Softphone Sample\r\n");
Log("To start please connect\r\n");

// disable buttons
m_btnLogin.EnableWindow(FALSE);
m btnLogout.EnableWindow(FALSE);
m btnReady.EnableWindow(FALSE);
m btnNotReady.EnableWindow(FALSE);
m btnMakeCall.EnableWindow(FALSE);
m btnAnswer.EnableWindow(FALSE);
m btnRelease.EnableWindow(FALSE);
m btnHold.EnableWindow(FALSE);
m btnRetrieve.EnableWindow(FALSE);
m btnCCConference.EnableWindow(FALSE);
m btnConference.EnableWindow(FALSE);
m btnSSConference.EnableWindow(FALSE);
m btnCCTransfer.EnableWindow(FALSE);
m btnConference.EnableWindow(FALSE);
m btnSSTransfer.EnableWindow(FALSE);

return TRUE;  // return TRUE unless you set the focus to a control

void CExercise3Dlg::OnClose()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == TRUE)
    {
        m_pSession->Disconnect(NULL);
    }

    // unsubscribe the event sink
    HRESULT hRes = AtlUnadvise(m_pSession, __uuidof(_IAllEvents), m_dwEventSinkAdvise);
    if( !SUCCEEDED(hRes) )
    {
        AfxMessageBox("Sink Unadvise failed!");
        ASSERT(FALSE);
    }
}
m_pSessionResolver->RemoveSession(_bstr_t(""));
CDialog::OnClose();
}

void CExercise3Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.
void CExercise3Dlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting
        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}
Appendix D      Sample COM for C++ Application

// The system calls this to obtain the cursor to display while the user drags
// the minimized window.
HCURSOR CExercise3Dlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CExercise3Dlg::OnConnect()
{
    int nRetVal;

    // Check to see if we are already connected
    if (m_bSessionConnected == TRUE)
    {
        AddLogMessage("Already Connected!");
        return;
    }

    AddLogMessage("Connecting to CTIOS Server...");
    CString hostA;
    CString portA;
    CString hostB;
    CString portB;
    m_edtHostA.GetWindowText(hostA);
    m_edtPortA.GetWindowText(portA);
    m_edtHostB.GetWindowText(hostB);
    m_edtPortB.GetWindowText(portB);

    // Create Arguments array
    CtiOsComArgumentsPtr pArgs;
    pArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));
    pArgs->AddItem("CtiosA", hostA.GetBuffer(0));
    pArgs->AddItem("PortA", portA.GetBuffer(0));
    pArgs->AddItem("CtiosB", hostB.GetBuffer(0));
    pArgs->AddItem("PortB", portB.GetBuffer(0));
    pArgs->AddItem("Heartbeat", OLESTR("60"));

    // perform Connect
    nRetVal = m_pSession->Connect(pArgs);

    if (nRetVal != CIL_OK)
    {
        AddLogMessage("Session->Connect to CTIOS Server Failed");
    }
    else
void CExercise3Dlg::OnDisconnect()
{
    AddLogMessage("Disconnecting from CTIOS Server...");
    // perform Disconnect - no arguments required
    m_pSession->Disconnect(NULL);
    m_edtConnection.SetWindowText("Offline");
}

void CExercise3Dlg::OnLogin()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = (enumCTIOS_AgentState)
        m_pAgent->GetAgentState();
    if (currentState != eUnknown && currentState != eLogout)
    {
        AddLogMessage("Already logged in!");
        return;
    }

    CString agentID;
    CString password;
    CString instrument;
    CString peripheralID;

    // get agent info from dialog
    m_edtAgentID.GetWindowText(agentID);
    m_edtPassword.GetWindowText(password);
    m_edtInstrument.GetWindowText(instrument);
    m_edtPeripheralID.GetWindowText(peripheralID);

    // set the agent object
    m_pAgent->SetValue(&_variant_t("AgentID"), &_variant_t(agentID.GetBuffer(0)));
m_pAgent->SetValue(_variant_t("AgentPassword"),
&_variant_t(password.GetBuffer(0)));    
m_pAgent->SetValue(_variant_t("AgentInstrument"),
&_variant_t(instrument.GetBuffer(0)));    
m_pAgent->SetValue(_variant_t("PeripheralID"),
&_variant_t(peripheralID.GetBuffer(0)));    

AddLogMessage("Requesting SetAgent");    

// perform SetAgent    
m_pSession->SetAgent(m_pAgent);    

// In order to perform the actual login you must wait for the OnSetAgentMode() event    
// so the real Login is performed int the OnSetAgentMode event handler (EventSink.cpp)
}

void CExercise3Dlg::OnLogout()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = (enumCTIOS_AgentState)
        m_pAgent->GetAgentState();
    if ( currentState == eUnknown || currentState == eLogout )
    {
        AddLogMessage("Must login first!");
        return;
    }

    // Create Arguments array
    CTIOSCLIENTLib::IArgumentsPtr pArgs;
    pArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));

    // No arguments required
    m_pAgent->Logout(pArgs);

    AddLogMessage("Requesting Agent Logout");
}

void CExercise3Dlg::OnReady()


```cpp
{    
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = (enumCTIOS_AgentState)
    m_pAgent->GetAgentState();
    if ( currentState == eUnknown || currentState == eLogout )
    {
        AddLogMessage("Must login first!");
        return;
    }

    // Create Arguments array
    CTIOSCLIENTLib::IArgumentsPtr pArgs;
    pArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));

    // add arguments
    pArgs->AddItem(_variant_t("AGENTSTATE"), _variant_t((long) eAvailable));
    pArgs->AddItem(_variant_t("REASONCODE"), _variant_t("1234"));
    m_pAgent->SetAgentState(pArgs);
    AddLogMessage("Requesting Agent State = Ready");
}

void CXercise3Dlg::OnNotready()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = (enumCTIOS_AgentState)
    m_pAgent->GetAgentState();
    if ( currentState == eUnknown || currentState == eLogout )
    {
        AddLogMessage("Must login first!");
        return;
    }
```
Appendix D  Sample COM for C++ Application

// Create Arguments array
CTIOSCLIENTLib::IArgumentsPtr pArgs;
pArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));

// add arguments
pArgs->AddItem(_variant_t("AGENTSTATE"), _variant_t((long) eNotReady));
pArgs->AddItem(_variant_t("REASONCODE"), _variant_t("1234"));

m_pAgent->SetAgentState(pArgs);
AddLogMessage("Requesting Agent State = NotReady");
}

void CExercise3Dlg::OnMakecall()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = (enumCTIOS_AgentState)
    m_pAgent->GetAgentState();
    if ( currentState == eUnknown || currentState == eLogout )
    {
        AddLogMessage("Must login first!");
        return;
    }

    // get phone number and call variable (if present) from dialog
    CString dn;
    CString cv;
    m_edtPhoneNumber.GetWindowText(dn);
    m_edtCallVariable1.GetWindowText(cv);
    if (dn.IsEmpty() == TRUE)
    {
        AddLogMessage("Must provide phone number to dial!");
        return;
    }

    // Create Arguments array
    CTIOSCLIENTLib::IArgumentsPtr pArgs;
pArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));
// add arguments
pArgs->AddItem(&_variant_t("DIALEDNUMBER"), &_variant_t(dn.GetBuffer(0)));

if (cv.IsEmpty() != TRUE)
{
    pArgs->AddItem(&_variant_t("CALLVARIABLE1"), &_variant_t(cv.GetBuffer(0)));
}

m_pAgent->MakeCall(pArgs);
AddLogMessage("Requesting MakeCall");
}

void CExercise3Dlg::OnAnswer()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = (enumCTIOS_AgentState)
    m_pAgent->GetAgentState();
    if (currentState == eUnknown || currentState == eLogout)
    {
        AddLogMessage("Must login first!");
        return;
    }

    // Get pointer to the active call object
    CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t("CurrentCall"));
    if (pCall)
    {
        AddLogMessage("Requesting Call Answer");

        // Create arguments object
        CTIOSCLIENTLib::IArgumentsPtr pArgs;
        HRESULT hRes = pArgs.CreateInstance("CtiOsComArguments.ComArguments");

        // No arguments needed
        pCall->Answer(pArgs);
    }
}
void CExercise3Dlg::OnRelease()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = (enumCTIOS_AgentState)
    m_pAgent->GetAgentState();
    if (currentState == eUnknown || currentState == eLogout)
    {
        AddLogMessage("Must login first!");
        return;
    }

    // Get pointer to the active call object
    CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t("CurrentCall"));
    if (pCall)
    {
        AddLogMessage("Requesting Call Release");

        // Create arguments object
        CTIOSCLIENTLib::IArgumentsPtr pArgs;
        HRESULT hRes = pArgs.CreateInstance("CtiOsComArguments.ComArguments");

        // No arguments needed
        pCall->ClearConnection(pArgs);
    }
}

void CExercise3Dlg::OnHold()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = (enumCTIOS_AgentState)
    m_pAgent->GetAgentState();
    if (currentState == eUnknown || currentState == eLogout)
AddLogMessage("Must login first!");
return;
}

// Get pointer to the active call object
CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(_variant_t("CurrentCall"));
if (pCall)
{
    AddLogMessage("Requesting Call Hold");
    // Create arguments object
    CTIOSCLIENTLib::IArgumentsPtr pArgs;
    HRESULT hRes = pArgs.CreateInstance("CtiOsComArguments.ComArguments");
    // No arguments needed
    pCall->Hold(pArgs);
}

void CExercise3Dlg::OnRetrieve()
{
    // Check to see if we are already connected
    if (m_bSessionConnected == FALSE)
    {
        AddLogMessage("Must be connected first!");
        return;
    }

    // Check to see if we are already logged in
    enumCTIOS_AgentState currentState = (enumCTIOS_AgentState)
        m_pAgent->GetAgentState();
    if (currentState == eUnknown || currentState == eLogout)
    {
        AddLogMessage("Must login first!");
        return;
    }

    // Get pointer to the active call object
    CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(_variant_t("CurrentCall"));
    if (pCall)
    {
        AddLogMessage("Requesting Call Retrieve");
        // Create arguments object
Appendix D

Sample COM for C++ Application

```cpp
CTIOSCLIENTLib::IArgumentsPtr pArgs;
HRESULT hRes = pArgs.CreateInstance("CtiOsComArguments.ComArguments");

// No arguments needed
pCall->Retrieve(pArgs);
}
}

void CExercise3Dlg::OnCCConference()
{
    // Get Phone number from dialog
    CString dn;
m_edtPhoneNumber.GetWindowText(dn);

    if (dn.IsEmpty() == TRUE)
    {
        AddLogMessage("Must fill in phone number field for consult call.");
        return;
    }

    // Get Call Variable1 value
    CString cv1;
m_edtCallVariable1.GetWindowText(cv1);

    // Get pointer to the active call object
    CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t("CurrentCall"));

    if (pCall)
    {
        AddLogMessage("Requesting Consult Call");

        // Create Arguments array
        CTIOSCLIENTLib::IArgumentsPtr pArgs;
pArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));

        // add arguments
        pArgs->AddItem(&_variant_t("DIALEDNUMBER"), &_variant_t(dn.GetBuffer(0)));
pArgs->AddItem(&_variant_t("CALLVARIABLE1"), &_variant_t(cv1.GetBuffer(0)));
pArgs->AddItem(&_variant_t("ConsultType"), &_variant_t(2L));

        pCall->MakeConsultCall(pArgs);
    }
}

void CExercise3Dlg::OnConference()
{
    // Get pointer to the active call object
```
CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t("CurrentCall"));

if (pCall)
{
    AddLogMessage("Requesting Conference Complete");
    // Create Arguments array
    CTIOSCLIENTLib::IArgumentsPtr pArgs;
    pArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));
    // no arguments needed
    pCall->Conference(pArgs);
}

void CExercise3Dlg::OnSSConference()
{
    // Get Phone number
    CString dn;
    m_edtPhoneNumber.GetWindowText(dn);
    if (dn.IsEmpty() == TRUE)
    {
        AddLogMessage("Must fill in phone number field for conference");
        return;
    }

    // Get Call Variable1 value
    CString cv1;
    m_edtCallVariable1.GetWindowText(cv1);

    // Get pointer to the active call object
    CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t("CurrentCall"));
    if (pCall)
    {
        AddLogMessage("Requesting Single Step Conference");
        // Create Arguments array
        CTIOSCLIENTLib::IArgumentsPtr pArgs;
        pArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));
        // add arguments
        pArgs->AddItem(&_variant_t("DIALEDNUMBER"), &_variant_t(dn.GetBuffer(0)));
        pArgs->AddItem(&_variant_t("CALLVARIABLE1"),
                        &_variant_t(cv1.GetBuffer(0)));
    }
Appendix D      Sample COM for C++ Application

```cpp
void CExercise3Dlg::OnCCTransfer()
{
    // Get Phone number from dialog
    CString dn;
    m_edtPhoneNumber.GetWindowText(dn);

    if (dn.IsEmpty() == TRUE)
    {
        AddLogMessage("Must fill in phone number field for consult call.");
        return;
    }

    // Get Call Variable1 value
    CString cv1;
    m_edtCallVariable1.GetWindowText(cv1);

    // Get pointer to the active call object
    CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t("CurrentCall"));

    if (pCall)
    {
        AddLogMessage("Requesting Consult Call");

        // Create Arguments array
        CTIOSCLIENTLib::IArgumentsPtr pArgs;
        pArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));

        // add arguments
        pArgs->AddItem(&_variant_t("DIALEDNUMBER"), &_variant_t(dn.GetBuffer(0)));
        pArgs->AddItem(&_variant_t("CALLVARIABLE1"),
                        &_variant_t(cv1.GetBuffer(0)));
        pArgs->AddItem(&_variant_t("ConsultType"), &_variant_t(1L));

        pCall->MakeConsultCall(pArgs);
    }
}
```

```cpp
void CExercise3Dlg::OnTransfer()
{
    // Get pointer to the active call object
    CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t("CurrentCall"));

    if (pCall)
    {
```
AddLogMessage("Requesting Transfer Complete");

// Create Arguments array
CTIOSCLIENTLib::IArgumentsPtr pArgs;
pArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));

// no arguments needed
pCall->Transfer(pArgs);
}

void CExercise3Dlg::OnSSTransfer()
{
    // Get Phone number from dialog
    CString dn;
m_edtPhoneNumber.GetWindowText(dn);
    if (dn.IsEmpty() == TRUE)
    {
        AddLogMessage("Must fill in phone number field for transfer");
        return;
    }

    // Get Call Variable1 value
    CString cv1;
m_edtCallVariable1.GetWindowText(cv1);

    // Get pointer to the active call object
    CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t("CurrentCall"));
    if (pCall)
    {
        AddLogMessage("Requesting Single Step Transfer");

        // Create Arguments array
        CTIOSCLIENTLib::IArgumentsPtr pArgs;
pArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));

        // add arguments
        pArgs->AddItem(&_variant_t("DIALEDNUMBER"), &_variant_t(dn.GetBuffer(0)));
        pArgs->AddItem(&_variant_t("CALLVARIABLE1"),
                        &_variant_t(cv1.GetBuffer(0)));

        pCall->SingleStepTransfer(pArgs);
    }
}
void CExercise3Dlg::OnGetData()
{
    AddLogMessage("Get Data pressed.");

    // Get pointer to the active call object
    CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t("CurrentCall"));

    if (pCall)
    {
        CTIOSCLIENTLib::IArgumentsPtr pArgs;
        pArgs = pCall->GetCallData();
        LogEvent("GetCallData returns:", pArgs);
    }
    else
    {
        AddLogMessage("No active call to get data from.");
    }
}

void CExercise3Dlg::OnSetData()
{
    int nErrorCode;
    AddLogMessage("Set Data pressed.");

    // Get pointer to the active call object
    CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t("CurrentCall"));

    if (pCall)
    {
        CTIOSCLIENTLib::IArgumentsPtr pArgs;
        pArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));

        CTIOSCLIENTLib::IArgumentsPtr pECCArgs;
        pECCArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));

        // regular variables
        pArgs->AddItem(&_variant_t("ANI"), &_variant_t("MyANI"));
        pArgs->AddItem(&_variant_t("DNIS"), &_variant_t("MyDNIS"));
        pArgs->AddItem(&_variant_t("CED"), &_variant_t("MyCED"));
        pArgs->AddItem(&_variant_t("CallVariable1"), &_variant_t("MyCallVariable1"));

        // ECC variables
        pECCArgs->AddItem(&_variant_t("Scalar1"), &_variant_t("MyScalar1"));
        pECCArgs->AddItem(&_variant_t("Scalar2"), &_variant_t("MyScalar2"));
    }
pECCArgs->AddItem(&_variant_t("Array1[1]"), &_variant_t("FirstArrayValue"));
pECCArgs->AddItem(&_variant_t("Array1[2]"), &_variant_t("SecondArrayValue"));

VARIANT varECC;
varECC.vt = VT_DISPATCH;
varECC.pdispVal = pECCArgs;

// Add ECC arguments to args
pArgs->AddItem(&_variant_t("ECC"), &varECC);
pCall->SetCallData(pArgs, &nErrorCode);

LogEvent("SetData sending:", pArgs);

else
{
    AddLogMessage("No active call to set data into.");
}
}

void CExercise3Dlg::OnGetContext()
{
    AddLogMessage("Get Context pressed.");

    // Get pointer to the active call object
    CTIOSCLIENTLib::ICallPtr pCall = m_pSession->GetValue(&_variant_t("CurrentCall"));

    if (pCall)
    {
        CTIOSCLIENTLib::IArgumentsPtr pArgs;
        pArgs = pCall->GetCallContext();
        LogEvent("GetCallContext returns:", pArgs);
    }
    else
    {
        AddLogMessage("No active call to get context data from.");
    }
}

void CExercise3Dlg::OnClearlog()
{
    // clear text from log window
    m_edtLogWindow.SetWindowText(""");
}
Appendix D  Sample COM for C++ Application

// Convert agent state enum to string
//
char * CExercise3Dlg::AgentStateToString(enumCTIOS_AgentState eState)
{
    switch( eState )
    {
        case eLogin:
            return "Login";
        case eLogout:
            return "Logout";
        case eNotReady:
            return "NotReady";
        case eAvailable:
            return "Available";
        case eTalking:
            return "Talking";
        case eWorkNotReady:
            return "WorkNotReady";
        case eWorkReady:
            return "WorkReady";
        case eBusyOther:
            return "BusyOther";
        case eReserved:
            return "Reserved";
        case eUnknown:
            return "Unknown";
        case eHold:
            return "Hold";
        default:
            return "Invalid State";
    }
}

void CExercise3Dlg::LogEvent(const char *msg, CTIOSCLIENTLib::IArguments *pIArguments)
{
    CString str;

    // display event name
    str = msg;
    str += " Event received";
    Log(str.GetBuffer(0));

    // create and display event contents
    str = "\t";

str += pIArguments->DumpArgs();
Log(str.GetBuffer(0));
}

void CExercise3Dlg::Log(const char *newText)
{
    CString oldMsg;
    CString newMsg;
    m>EditLogWindow.GetWindowText(oldMsg);
    newMsg = oldMsg + "\r\n" + newText;
    m>EditLogWindow.SetWindowText(newMsg);
    m>EditLogWindow.SetSel(newMsg.GetLength(), -1);
}

========================= End "exercise3Dlg.cpp" ============================

========================= Start "stdafx.h" ===================================

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//
#if !defined(AFX_STDAFX_H__2848E5D9_3C1C_44F0_A041_A7A838FB8E8C__INCLUDED_)
#define AFX_STDAFX_H__2848E5D9_3C1C_44F0_A041_A7A838FB8E8C__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#pragma warning ( disable : 4146 )
#define VC_EXTRALEAN// Exclude rarely-used stuff from Windows headers
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC Automation classes
#include <afxdtctl.h>       // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h> // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
Appendix D      Sample COM for C++ Application

#import <ctiossessionresolver.dll>
using namespace CTIOSSESSIONRESOLVERLib;

#import <ctiosclient.dll>
using namespace CTIOSCLIENTLib;

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
#endif // !defined(AFX_STDAFX_H__2848E5D9_3C1C_44F0_A041_A7A838FB8E8C__INCLUDED_)

================================ End "stdafx.h" =================================

================================ Start "stdafx.cpp" ================================

// stdafx.cpp : source file that includes just the standard includes
// exercise3.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information
#include "stdafx.h"

================================ End "stdafx.cpp" =================================

================================ Start "resource.h" ==================================
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by exercise3.rc

#define IDCONNECT 3
#define IDDISCONNECT 4
#define IDLOGIN 5
#define IDREADY 6
#define IDLOGOUT 7
#define IDNOTREADY 8
#define IDCLEARLOG 9
#define IDMAKECALL 11
#define IDANSWER 13
#define IDRETRIEVE 14
#define IDHOLD 15
#define IDM_ABOUTBOX 0x0010
#define IDSSCONFERENCE 18
#define IDM_ABOUTBOX 100

Cisco ICM Software CTI OS Developer's Guide
#define IDS_ABOUTBOX 101
#define IDD_EXERCISE3_DIALOG 102
#define IDR_MAINFRAME 128
#define IDC_HOSTA 1000
#define IDC_PORTA 1001
#define IDC_HOSTB 1002
#define IDC_PORTB 1003
#define IDC_AGENTID 1004
#define IDC_PASSWORD 1005
#define IDC_INSTRUMENT 1006
#define IDC_PERIPHERALID 1007
#define IDC_CONNECTION 1008
#define IDC_AGENTSTATE 1009
#define IDC_MODE 1011
#define IDC_PHONENUMBER 1012
#define IDC_CALLVARIABLE1 1013
#define IDC_CURRENTCALLUID 1014
#define IDC_LOGWINDOW 1015
#define IDC_CCCONFERENCES 1016
#define IDC_CCCONFERENCES 1016
#define IDC_CTCOMMANDS 1017
#define IDC_CTCOMMANDS 1017
#define IDC_IDTRANSFERS 1019
#define IDC_IDTRANSFERS 1019
#define ID_IDGETDATA 1020
#define ID_IDSETDATA 1021
#define ID_IDGETCONTEXT 1022
#define ID_IDGETCONTEXT 1022

// Next default values for new objects

#ifdef APSTUDIO_INVOKED
#endif APSTUDIO_READONLY_SYMBOLS
#ifdef APSTUDIO_INVOKED
#else APSTUDIO_READONLY_SYMBOLS
#endif APSTUDIO_INVOKED

#define _APS_NEXT_RESOURCE_VALUE 129
#define _APS_NEXT_COMMAND_VALUE 32771
#define _APS_NEXT_CONTROL_VALUE 1023
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif
#endif

// End "resource.h"
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:

\texttt{HKEY\_LOCAL\_MACHINE\Software\Cisco\ Systems\Ctios\Logging\TraceMask}

\textbf{Warning} \textit{The default value for the trace masks is 0x07. 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.}
<table>
<thead>
<tr>
<th>A</th>
<th>StopMonitoringAgentTeam 9-42</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>StopMonitoringAllAgentTeam 9-43</td>
</tr>
<tr>
<td></td>
<td>SuperviseCall 9-44</td>
</tr>
<tr>
<td></td>
<td>AgentSelectCtl softphone control 5-8</td>
</tr>
<tr>
<td></td>
<td>agent state</td>
</tr>
<tr>
<td></td>
<td>CTI application control 1-2</td>
</tr>
<tr>
<td></td>
<td>AgentStateCtl softphone control 5-5</td>
</tr>
<tr>
<td></td>
<td>AgentStatisticsCtl softphone control 5-9</td>
</tr>
<tr>
<td></td>
<td>Alternate call object method 10-10</td>
</tr>
<tr>
<td></td>
<td>AlternateCtl softphone control 5-9</td>
</tr>
<tr>
<td></td>
<td>Answer call object method 10-11</td>
</tr>
<tr>
<td></td>
<td>AnswerCtl softphone control 5-10</td>
</tr>
<tr>
<td></td>
<td>Architecture 2-1</td>
</tr>
<tr>
<td></td>
<td>Arg class 12-2</td>
</tr>
<tr>
<td></td>
<td>Arg class methods</td>
</tr>
<tr>
<td></td>
<td>Clone 12-3</td>
</tr>
<tr>
<td></td>
<td>CreateInstance 12-4</td>
</tr>
<tr>
<td></td>
<td>DumpArg 12-5</td>
</tr>
<tr>
<td></td>
<td>GetType 12-6</td>
</tr>
<tr>
<td></td>
<td>GetType 12-6</td>
</tr>
<tr>
<td></td>
<td>SetValueType 12-7</td>
</tr>
<tr>
<td></td>
<td>SetValue 12-9</td>
</tr>
<tr>
<td></td>
<td>Arguments class methods</td>
</tr>
<tr>
<td></td>
<td>AddItem 12-13</td>
</tr>
<tr>
<td></td>
<td>Clear 12-15</td>
</tr>
<tr>
<td></td>
<td>Clone 12-15</td>
</tr>
</tbody>
</table>

| Accessing properties and parameters 3-9 |
| ActiveX Softphone Controls 5-1 |
| AddItem |
|   | Arguments class method 12-13 |
| Agent mode 2-6 |
| agent object methods 9-11 |
|   | GetAgentState 9-17 |
|   | GetMonitoredAgent 9-17 |
|   | GetMonitoredCall 9-18 |
|   | GetSkillGroups 9-19 |
| Login 9-19 |
| Logout 9-21 |
| MakeCall 9-23 |
| ReportBadCallLine 9-30 |
| RequestAgentTeamList 9-31 |
| RequestSupervisorAssist 9-32 |
| SendChatMessage 9-32 |
| SetAgentState 9-34 |
| StartMonitoringAgent 9-37 |
| StartMonitoringAgentTeam 9-38 |
| StartMonitoringAllAgentTeams 9-39 |
| StartMonitoringCall 9-40 |
| StopMonitoringAgent 9-41 |
Index

DumpArgs  12-17
GetElement  12-18
GetValue  12-19
IsValid  12-22
NumElements  12-22
RemoveItem  12-24
SetElement  12-24
SetValue  12-26
Arguments structure  3-8
asynchronous events  1-3

B
BadLineCtl softphone control  5-10
Button controls  5-4
ButtonCtl softphone control  5-10
Button enablement masks  3-14

C
CallAppearanceCtl softphone control  5-10
call object methods  10-7
Alternate  10-10
Answer  10-11
Clear  10-12
ClearConnection  10-13
Conference  10-14
Deflect  10-16
GetCallContext  10-17
GetCallData  10-19
Hold  10-20
MakeConsultCall  10-22
Reconnect  10-28
Retrieve  10-29
SendDTMFSignal  10-30
SetCallData  10-32
SingleStepConference  10-33
SingleStepTransfer  10-36
Snapshot  10-37
StartRecord  10-38
StopRecord  10-39
Transfer  10-40
CCtiosException class  12-29
ChatCtl softphone control  5-12
CIL error codes  3-4
CIL object model  2-4
CILRefArg class methods
  GetType  12-28
  GetValue  12-28
  SetValue  12-29
Clear
  Arguments class method  12-15
Clear call object method  10-12
ClearConnection call object method  10-13
Client Interface Library (CIL)  2-2
Clone
  Arg class method  12-3
  Arguments class method  12-15
Coding conventions 3-1
COM CIL
    using 4-12
COM CIL (C++) 4-15
COM error codes 3-7
Conference call object method 10-14
ConferenceCtl softphone control 5-12
Connecting to CTI OS Server 4-24
Connection Mode
    setting 4-25
Connect session object method 8-6
CreateInstance
    Arg class method 12-4
    Arguments class method 12-16
CTI-enabled applications 1-2
CTI OS
    advantages 1-6
    application architecture 1-6
    architecture 2-1
    benefits 1-7

D
Deflect call object method 10-16
DisableSkillGroupStatistics SkillGroup object method 11-25
Disconnecting from CTI OS Server 4-26
Disconnect session object method 8-7
DumpArg Arg class method 12-5
DumpArgs

Arguments class method 12-17

E
EmergencyAssistCtl softphone control 5-14
EnableSkillGroupStatistics SkillGroup object method 11-26
Event flow 1-3

G
GetAgentState agent object method 9-17
GetAllAgents session object method 8-8
GetAllCalls session object method 8-8
GetAllSkillGroups session object method 8-9
GetCallContext call object method 10-17
GetCallData call object method 10-19
GetElement Arguments class method 12-18
GetMonitoredAgent agent object method 9-17
GetMonitoredCall agent object method 9-18
GetObjectFromObjectID session object method 8-12
GetSkillGroups agent object method 9-19
GetType
    Arg class method 12-6
    CILRefArg class method 12-28
GetValue
    Arguments class method 12-19
    CILRefArg class method 12-28
GetValueType
Arg class method 12-7
Grid controls 5-4
GridControl softphone control 5-15

H
Hold call object method 10-20
HoldCtl softphone control 5-15

I
IAgentEvents interface 6-57
ICallEvents interface 6-10
Integrating an application 4-1
IsAgent session object method 8-13
ISessionEvents interface 6-2
ISKillGroupEvents interface 6-85
IsSupervisor session object method 8-14
IsValid
Arguments class method 12-22

M
MakeCall agent object method 9-23
MakeCallCtl softphone control 5-16
MakeConsultCall call object method 10-22
Monitor mode 2-6

N
Not Ready button 5-7
NumElements
  Arguments class method 12-22

O
Object Interface Framework 2-4
Object properties
  setting 3-10
  OnAgentPreCallAbortEvent event 6-37
  OnAgentPreCallEvent event 6-34, 6-35
  OnAgentStatistics event 6-59
  OnAlternateCallConf event 6-11
  OnAnswerCallConf event 6-11
  OnCallBegin event 6-12
  OnCallCleared event 6-23
  OnCallConferenced event 6-30
  OnCallConnectionCleared event 6-24
  OnCallDataUpdate event 6-15
  OnCallDelivered event 6-18
OnCallDequeuedEvent event 6-42
OnCallDiverted event 6-33
OnCallEnd event 6-15
OnCallEstablished event 6-20
OnCallFailed event 6-26
OnCallHeld event 6-22
OnCallOriginated event 6-25
OnCallQueuedEvent event 6-40
OnCallReachedNetworkEvent event 6-43
OnCallRetrieved event 6-22
OnCallServiceInitiatedEvent event 6-38
OnCallStartRecordingConf event 6-45
OnCallStopRecordingConf event 6-45
OnCallTransferred event 6-27
OnClearCallConf event 6-46
OnClearConnectionConf event 6-46
OnConferenceCallConf event 6-47
OnConnectionClosed message 6-3
OnConnection event 6-2
OnConnectionFailure event 6-3
OnConnectionRejected event 6-4
OnConsultationCallConf event 6-48
OnControlFailureConf event 6-47
OnCTIOSFailure event 6-4
OnCurrentAgentReset message 6-6
OnCurrentCallChanged message 6-7
OnDeflectCallConf event 6-49
OnGlobalSettingsDownloadConf event 6-7
OnHeartbeat event 6-7
OnHoldCallConf event 6-49
OnMakeCallConf event 6-71
OnMissingHeartbeat event 6-8
OnMonitorModeEstablished event 6-9
OnReconnectCallConf event 6-50
OnRetrieveCallConf event 6-50
OnSendDTMFConf event 6-51
OnSetAgentMode event 6-10
OnSetAgentStateConf event 6-75
OnSetCallDataConf event 6-51
OnSkillGroupStatisticsUpdated event 6-85
OnSnapshotCallConf event 6-52
OnSnapshotDeviceConf event 6-76
OnTransferCallConf event 6-54

R

Ready button 5-6
Reconnect call object method 10-28
ReconnectCtl softphone control 5-17
RecordCtl softphone control 5-20
Reference counting 2-7
Referencing COM components 4-13
Registering for events in Visual Basic 4-14
RemoveItem
Arguments class method 12-24
ReportBadCallLine agent object method 9-30
RequestAgentTeamList agent object method 9-31
Index

RequestDesktopSettings session object method 8-14
Request parameters
   setting 3-10
Request-response mechanism 1-4
RequestSupervisorAssist agent object method 9-32
Retrieve call object method 10-29

S

screen pop 1-2
SendChatMessage agent object method 9-32
SendDTMFSignal call object method 10-30
session object methods 8-4
   Connect 8-6
   Disconnect 8-7
   GetAllAgents 8-8
   GetAllCalls 8-8
   GetAllSkillGroups 8-9
   GetObjectFromObjectID 8-12
   IsAgent 8-13
   IsSupervisor 8-14
   RequestDesktopSettings 8-14
   SetAgent 8-15
   SetCurrentCall 8-16
   SetMessageFilter 8-17
   SetAgent session object method 8-15
   SetAgentState agent object method 9-34
   SetCallData call object method 10-32
   SetCurrentCall session object method 8-16
   SetElement
      Arguments class method 12-24
   SetMessageFilter session object method 8-17
   Setting object properties 3-10
   Setting request parameters 3-10
   SetValue
      Arg class method 12-9
      Arguments class method 12-26
   SingleStepConference call object method 10-33
   SingleStepTransfer call object method 10-36
   SkillGroup object methods 11-24
      DisableSkillGroupStatistics 11-25
      EnableSkillGroupStatistics 11-26
   SkillgroupStatisticsCtl softphone control 5-18
   Snapshot call object method 10-37
   softphone controls
      AgentSelectCtl 5-8
      AgentStateCtl 5-5
      AgentStatisticsCtl 5-9
      AlternateCtl 5-9
      AnswerCtl 5-10
      BadLineCtl 5-10
      ButtonCtl 5-10
      CallAppearanceCtl 5-10
      ChatCtl 5-12
      ConferenceCtl 5-12
      EmergencyAssistCtl 5-14
      GridControl 5-15
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
</table>

| **HoldCtl** | 5-15 |
| **MakeCallCtl** | 5-16 |
| **RecordCtl** | 5-20 |
| **SkillgroupStatisticsCtl** | 5-18 |
| **StatusBarCtl** | 5-18 |
| **SupervisorOnlyCtl** | 5-19 |
| **TransferCtl** | 5-21 |

| StartMonitoringAgent agent object method | 9-37 |
| StartMonitoringAgentTeam agent object method | 9-38 |
| StartMonitoringAllAgentTeams agent object method | 9-39 |
| StartMonitoringCall agent object method | 9-40 |
| StartRecord call object method | 10-38 |

| Static libraries | 4-18 |
| **StatusBarCtl** softphone control | 5-18 |
| **STLport** | 4-22 |
| **StopMonitoringAgent agent object method** | 9-41 |
| **StopMonitoringAgentTeam agent object method** | 9-42 |
| **StopMonitoringAllAgentTeams agent object method** | 9-43 |
| **StopRecord call object method** | 10-39 |
| **Subscribing for events in C++** | 4-22 |
| **SuperviseCall agent object method** | 9-44 |
| **SupervisorOnlyCtl softphone control** | 5-19 |

| **T** |
| Third-party call control | 1-3 |
| Transfer call object method | 10-40 |
| TransferCtl softphone control | 5-21 |

| **U** |
| UniqueObjectID | 3-11 |

| **V** |
| Visual Basic | 4-12 |

| **W** |
| Work Not Ready button | 5-8 |
| Work Ready button | 5-7 |