Cisco ICM Software CTI OS
Developer’s Guide
ICM Software Version 4.6

Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

Customer Order Number:
Text Part Number: OL-1392-01
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.

AccessPath, AtmDirector, Browse with Me, CCIP, CCSI, CD-PAC, CiscoLink, the Cisco Powered Network logo, Cisco Systems Networking Academy, the Cisco Systems Networking Academy logo, Fast Step, Follow Me Browsing, FormShare, FrameShare, GigaStack, IGX, Internet Quotient, IP/VC, iQ Breakthrough, iQ Expertise, iQ FastTrack, the iQ Logo, iQ Net Readiness Scorecard, MGX, the Networkers logo, Packet, RateMUX, ScriptBuilder, ScriptShare, SlideCast, SMARTnet, TransPath, Unity, Voice LAN, Wavelength Router, and WebViewer are trademarks of Cisco Systems, Inc.; Changing the Way We Work, Live, Play, and Learn, Discover All That’s Possible, and Empowering the Internet Generation, 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 Systems, Cisco Systems Capital, the Cisco Systems logo, Enterprise/Solver, EtherChannel, EtherSwitch, FastHub, FastSwitch, IOS, IP/TV, LightStream, MICA, Network Registrar, PIX, Post-Routing, Pre-Routing, Registrar, 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. (0106R)

Cisco ICM Software CTI OS Developer’s Guide
Copyright © 2001, Cisco Systems, Inc.
All rights reserved.
About This Guide  xxv
  Purpose  xxv
  Audience  xxv
  Conventions  xxv
  Organization  xxvi
  Other Publications  xxvii
  Obtaining Documentation  xxvii
    World Wide Web  xxvii
    Documentation CD-ROM  xxviii
    Ordering Documentation  xxviii
    Documentation Feedback  xxviii
  Obtaining Technical Assistance  xxix
    Cisco.com  xxix
    Technical Assistance Center  xxix
      Contacting TAC by Using the Cisco TAC Website  xxx
      Contacting TAC by Telephone  xxx

CHAPTER 1  Introduction  1-1
  Overview  1-2
  Extensibility and Ease of Use  1-5
  Accessing Properties with GetValue  1-7
  GetElement  1-9
  Property Names and Enumerated Constants  1-9
  Property Attributes  1-9
Platform Issues 1-11
Accessing Call and ECC Variables 1-12
Handling Errors 1-13
Event Driven Model 1-14
Event Cascade Model 1-15
Event Publication Model 1-16
Subscribing to an Event Interface 1-16
Special Values 1-16
Start Up, Connect, Snapshot 1-16

CHAPTER 2

CTI OS Client Interface Library Architecture  2-1

Object Interface Framework 2-1
  Session Object 2-2
  Session Manager 2-3
  Session Behavior 2-3
    Connection 2-3
    Connection Failure and Recovery 2-3
    Session Modes 2-3
  Object Manager and Event Passing 2-4
  Creating Objects 2-5
    Call Object Lifetime 2-5
    Agent Object Lifetime 2-6
    SkillGroup Object Lifetime 2-6
    Object Factory 2-6
  Event Publisher 2-7
    Exposed Interfaces 2-7
    Adding and Removing Subscribers 2-7
    Event Distribution 2-7
  Service Layer 2-8
Connection Layer 2-8
Multithreaded CTI OS Client Application 2-8
Support for Multithreaded Client in CIL 2-9
  Wait Object 2-9
    Properties 2-10
    Methods 2-10
    Usage at the Client 2-11
Multithreaded Application Example 2-12
Subclassing 2-14

CHAPTER 3

Handling Events 3-1
  Handling Events in C++ 3-2
  Creating a Subscriber Object Class 3-2
  Registering a C++ Subscriber Object 3-4
    Input Parameters 3-4
    Return Value 3-4
    Example 3-5
  Unregistering Subscriber 3-5
    Input Parameters 3-5
    Return Value 3-6
    Example 3-6
  Handling Events in COM 3-6
    Automation Based Applications 3-7
    ATL, MFC and COM SDK Based Applications 3-7

CHAPTER 4

Session Object 4-1
  Working With CTI OS Objects 4-2
  Methods 4-4
    Connect 4-5
Contents

Input Parameters  4-18
Return Values  4-18
RequestDesktopSettings  4-18
  Syntax  4-19
SetAgent  4-19
  Syntax  4-19
  Input Parameters  4-19
  Return Values  4-20
  Examples  4-20
SetCurrentCall  4-21
  Syntax  4-21
  Input Parameters  4-21
  Return Values  4-22
  Examples  4-22
SetMessageFilter  4-23
  Operator  4-23
  Expression  4-24
  Keyword  4-24
  Value  4-25
  Syntax  4-25
  Input Parameters  4-25
  Return Values  4-26
  Examples  4-26
ISessionEvents Interface  4-27
  OnConnection  4-27
  OnConnectionFailure  4-28
  OnHeartbeat  4-28
  OnMissingHeartbeat  4-28
  OnMonitorModeEstablished  4-29
  OnConnectionClosed  4-29
Contents

OnConnectionRejected 4-30
OnSetAgentModeEvent 4-30
OnCurrentCallChanged 4-30
OnCurrentAgentReset 4-30

CHAPTER 5

Agent Object 5-1

Methods 5-1

AgentTeamList 5-3
Syntax 5-3
Parameters 5-3
Return Values 5-3
Examples 5-4

BadCallLine 5-5
Syntax 5-5
Parameters 5-5
Return Values 5-5
Examples 5-6

Emergency 5-6
Syntax 5-7
Parameters 5-7
Return Values 5-7
Examples 5-7

EnableAgentStatistics/DisableAgentStatistics 5-8
Syntax 5-8
Parameters 5-9
Return Values 5-9
Examples 5-9

GetAgentState 5-10
Syntax 5-10
Output Parameters 5-10
Contents

Return Values 5-11
Examples 5-11
GetElement 5-12
GetMonitoredAgent 5-12
   Syntax 5-12
   Output Parameters 5-12
   Return Values 5-13
   Examples 5-13
GetMonitoredCall 5-14
   Syntax 5-14
   Output Parameters 5-14
   Return Values 5-14
   Examples 5-15
GetPropertyAttribute 5-16
   Syntax 5-16
GetSkillGroups 5-21
   Syntax 5-21
   Example 5-21
GetValue 5-21
   Syntax 5-22
Login 5-22
   Syntax 5-22
   Input Parameters 5-22
   Return Values 5-23
   Examples 5-23
Logout 5-24
   Syntax 5-24
   Input Parameters 5-25
   Return Values 5-25
   Examples 5-25
Contents

MakeCall 5-26
  Syntax 5-28
  Input Parameters 5-28
  Return Values 5-28
  Examples 5-29
MonitorAgentTeam 5-30
  Syntax 5-30
  Input Parameters 5-30
  Return Values 5-30
  Examples 5-31
MonitorAgentTeamAll 5-31
  Syntax 5-31
  Input Parameters 5-32
  Return Values 5-32
  Examples 5-32
OnEvent (C++ Only) 5-33
  Syntax 5-33
  Input Parameters 5-33
  Return Values 5-33
  Examples 5-33
SendUserMessage/SendChatMessage 5-34
  Syntax 5-34
  Input Parameters 5-35
  Return Values 5-35
  Examples 5-35
SetAgentState 5-36
  Syntax 5-36
  Input Parameters 5-37
  Return Values 5-37
  Examples 5-37
SetMonitoredAgent 5-38
   Syntax 5-38
   Input Parameters 5-38
   Return Values 5-39
   Examples 5-39
SetMonitoredCall 5-40
   Syntax 5-40
   Input Parameters 5-40
   Return Values 5-40
   Examples 5-41
SetValue 5-41
   Syntax 5-41
   Example 5-42
SuperviseCall 5-42
   Syntax 5-42
   Input Parameters 5-42
   Return Values 5-43
   Examples 5-43
SupervisorAssist 5-44
   Syntax 5-44
   Parameters 5-44
   Return Values 5-44
   Examples 5-45
IAgentEvents Interface 5-45
   OnAgentStateChange 5-46
   OnAgentStatistics 5-47
   OnUserMessage 5-47

CHAPTER 6
Call Object 6-1
   Active Call 6-1
Contents

Methods 6-2
   Alternate 6-3
      Syntax 6-3
      Parameters 6-3
      Return Values 6-4
      Examples 6-4
   Answer 6-5
      Syntax 6-5
      Parameters 6-5
      Return Values 6-5
      Examples 6-6
   Clear 6-6
      Syntax 6-7
      Parameters 6-7
      Return Values 6-7
      Examples 6-7
   ClearConnection 6-8
      Syntax 6-8
      Parameters 6-8
      Return Values 6-9
      Examples 6-9
   Conference 6-10
      Syntax 6-10
      Input Parameters 6-11
      Return Values 6-11
      Examples 6-11
   Deflect 6-12
      Syntax 6-12
      Input Parameters 6-13
      Return Values 6-13
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples</td>
<td>6-13</td>
</tr>
<tr>
<td>GetElement</td>
<td>6-14</td>
</tr>
<tr>
<td>Syntax</td>
<td></td>
</tr>
<tr>
<td>GetPropertyAttribute</td>
<td>6-14</td>
</tr>
<tr>
<td>Syntax</td>
<td>6-16</td>
</tr>
<tr>
<td>GetValue</td>
<td>6-16</td>
</tr>
<tr>
<td>Syntax</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>6-17</td>
</tr>
<tr>
<td>Hold</td>
<td>6-17</td>
</tr>
<tr>
<td>Syntax</td>
<td>6-17</td>
</tr>
<tr>
<td>Parameters</td>
<td>6-17</td>
</tr>
<tr>
<td>Return Values</td>
<td>6-18</td>
</tr>
<tr>
<td>Examples</td>
<td>6-18</td>
</tr>
<tr>
<td>MakeConsultCall</td>
<td>6-19</td>
</tr>
<tr>
<td>Syntax</td>
<td>6-20</td>
</tr>
<tr>
<td>Input Parameters</td>
<td>6-20</td>
</tr>
<tr>
<td>Return Values</td>
<td>6-21</td>
</tr>
<tr>
<td>Examples</td>
<td>6-21</td>
</tr>
<tr>
<td>OnEvent (C++ Only)</td>
<td>6-22</td>
</tr>
<tr>
<td>Syntax</td>
<td>6-22</td>
</tr>
<tr>
<td>Input Parameters</td>
<td>6-22</td>
</tr>
<tr>
<td>Return Values</td>
<td>6-23</td>
</tr>
<tr>
<td>Examples</td>
<td>6-23</td>
</tr>
<tr>
<td>Reconnect</td>
<td>6-23</td>
</tr>
<tr>
<td>Syntax</td>
<td>6-23</td>
</tr>
<tr>
<td>Parameters</td>
<td>6-24</td>
</tr>
<tr>
<td>Return Values</td>
<td>6-24</td>
</tr>
<tr>
<td>Examples</td>
<td>6-24</td>
</tr>
<tr>
<td>Record</td>
<td>6-25</td>
</tr>
<tr>
<td>Syntax</td>
<td>6-25</td>
</tr>
</tbody>
</table>
Parameters 6-38
Return Values 6-38
Examples 6-39

Transfer 6-40
Syntax 6-40
Input Parameters 6-40
Return Values 6-41
Examples 6-41

ICallEvents Interface 6-42
OnCallBegin 6-42
OnCallEnd 6-44
OnCallDataUpdate 6-45
OnCallDelivered 6-46
OnCallEstablished 6-48
OnCallHeld 6-51
OnCallRetrieved 6-52
OnCallCleared 6-53
OnCallConnectionCleared 6-53
OnCallOriginated 6-54
OnCallFailed 6-56
OnCallTransferConferenceInitiated 6-57
OnCallTransferred 6-57
OnCallConferenced 6-59
OnCallDiverted 6-61
OnTranslationRoute 6-62
OnCallEnterpriseAgent 6-63
OnCallPreEventAbort 6-65
OnCallRequestFailed 6-65
OnAgentPreCallEvent 6-65
OnAgentPreCallAbortEvent 6-67
SkillGroup Object 7-1
Methods 7-2
EnableSkillGroupStatistics/DisableSkillGroupStatistics 7-2
Syntax 7-2
Parameters 7-3
Return Values 7-3
Examples 7-3
GetPropertyAttribute 7-4
Syntax 7-22
Example 7-23
GetValue 7-23
Syntax 7-23
Example 7-23
OnEvent 7-24
Syntax 7-24
SkillGroupEvents Interface 7-24
OnSkillGroupStatisticsUpdated 7-24

Helper Classes 8-1
Arg Class 8-2
CreateInstance 8-3
Syntax 8-3
Parameters  8-4
Return Values  8-4
Examples  8-4
SetValue  8-5
Syntax  8-5
Input Parameters  8-6
Return Values  8-7
Examples  8-7
GetValueType  8-9
Syntax  8-9
Output Parameters  8-9
Return Values  8-10
Examples  8-11
Clone  8-12
Syntax  8-12
Output Parameters  8-13
Return Values  8-13
Examples  8-13
GetType  8-14
Syntax  8-14
Output Parameters  8-15
Return Values  8-15
Examples  8-15
GetClassID  8-17
Syntax  8-17
Output Parameters  8-17
Return Values  8-17
Examples  8-17
Arguments Class  8-19
AddItem  8-20
Contents

Syntax  8-21
Parameters  8-21
Returns  8-21
Clear  8-21
Syntax  8-21
Parameters  8-21
Clone  8-21
Syntax  8-21
Parameters  8-22
Returns  8-22
CreateInstance  8-22
Syntax  8-22
DumpArgs  8-22
Syntax  8-22
Parameters  8-22
GetClassID  8-23
Syntax  8-23
Parameters  8-23
Returns  8-23
GetElement  8-23
Syntax  8-23
GetValue  8-23
Syntax  8-24
Initialize  8-24
Syntax  8-24
Parameters  8-24
IsValid  8-24
Syntax  8-24
NumElements  8-24
Syntax  8-24
Contents

Parameters 8-25
RemovetItem 8-25
Syntax 8-25
SetElement 8-25
Syntax 8-25
SetValue 8-25
Syntax 8-26
CILRefArg Class 8-26
SetValue 8-26
Syntax 8-26
Input Parameters 8-27
Return Values 8-27
Examples 8-27
GetValue 8-27
Syntax 8-27
Parameters 8-27
Return Values 8-28
Examples 8-28
GetClassID 8-28
Syntax 8-28
Return Values 8-28
Examples 8-29
GetType 8-29
CCtiOsException Class 8-29
CCtiOsException 8-30
Syntax 8-30
Input Parameters 8-30
Return Values 8-31
Example 8-31
GetCode 8-31
Contents

Syntax 8-31
Parameters 8-31
Return Values 8-31
Example 8-31
GetStatus 8-32
Syntax 8-32
Parameters 8-32
Return Values 8-32
Example 8-32
GetString 8-32
Syntax 8-32
Parameters 8-33
Return Values 8-33
Example 8-33
What 8-33
Syntax 8-33
Parameters 8-33
Return Values 8-33
Example 8-34
CCtiOsObject Class 8-34
DumpProperties 8-35
Syntax 8-35
Parameters 8-35
Return Values 8-35
Example 8-36
GetPropertyname 8-36
Input Parameters 8-36
Return Values 8-36
Example 8-36
GetNumProperties 8-36
Syntax 8-36
Parameters 8-37
Return Values 8-37
Example 8-37
GetValueType 8-37
Syntax 8-37
Input Parameters 8-37
Return Values 8-38
Example 8-38
GetElement 8-38
Syntax 8-38
Input Parameters 8-38
Return Values 8-39
GetPropertyAttribute 8-39
Input Parameters 8-39
Return Values 8-39
SetValue 8-40
Syntax 8-40
Input Parameters 8-40
Return Values 8-40
Example 8-41
IsValid 8-41
Syntax 8-41
Input Parameters 8-41
Return Values 8-41
Example 8-41

APPENDIX A  CTI OS CIL Messages A-1
CIL CTI Server Message Equivalents A-1
Call Requests A-1
Agent Requests A-2
Skill Group Requests A-2
Supervisor Requests A-2
Generic CTI Server Confirmation Events A-3
Call Confirmation Events A-3
Agent Confirmation Events A-4
Supervisor Confirmation Events A-4
Error and Failure Events A-5
Call Events A-5
Call Recording Events A-6
Agent Events A-6
Skill Group Events A-6
Supervisor Events A-6
CIL CTI OS-specific messages A-7
Session Requests A-7
Single Step Transfer/Conference Requests A-7
Supervisor Requests A-7
Statistics Requests A-7
Timer Service Requests A-8
Team Maintenance Requests A-8
CTI OS Specific Confirmation Events A-8
CTI OS Specific Events A-8
Session Events A-8
Button Enablement Events A-9
Supervisor Events A-9
Filter Events A-10
Name Lookup Service Requests A-10
Name Lookup Service Event A-10
Timeout Events A-10
Supervised Calls A-10
Supervised Agent Events  A-11
Team Maintenance Events  A-11

Appendix B
CTI OS Keywords  B-1

Index
About This Guide

Purpose

This manual provides an overview of the Cisco CTI Object Server (CTI OS) product, describes the CTI OS Client Interface and CTI OS Architecture, and provides syntax and descriptions 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 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, “Handling Events”</td>
<td>Discusses CTI OS event handling.</td>
</tr>
<tr>
<td>Chapter 4, “Session Object”</td>
<td>Describes the methods and events associated with the CTI OS Session object.</td>
</tr>
</tbody>
</table>
About This Guide

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
- 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

CTI Object Server (CTI OS) is a high performance, scalable, fault-tolerant server-based solution to deploy CTI applications. CTI OS makes possible deployment of thin-client browser based applications that require no additional software to be installed at a desktop client. CTI OS serves as a single point of integration for third-party applications, including ERP systems, data mining and workflow solutions. Configuration and behavior information is managed at the server, simplifying customization, updates, and maintenance. Servers can be accessed and managed remotely.

Client applications interface to CTI OS using session, agent, and call interfaces. These interfaces are implemented in COM, C++, and C, allowing for a wide range of application development uses. The interfaces are used for control, to access to data values, and to receive event notifications.

The client interface was designed to support all of the following features:

- Extensible. The interface is designed so that it can be easily extended. New functionality in future revisions will not obsolete the current interfaces.
- Simplified and easy of use. Each method supports a variable number of arguments. This simplifies the use of optional arguments.
- No differentiation between Call Variables, ECC variables, or arguments stored in the Cisco Enterprise Data Store. The client interface simplifies access and use of ECC variables.
- Subclassable object architecture. Subclassing allows your custom defined call or agent object to be created instead of ours. This is a powerful mechanism that provides a client application complete control over all events and responses.
Automatic connection failover.

The architecture supports multiple threads and is thread safe. An application can spawn multiple threads using a single connection (session) object.

Applications can be deployed on virtually any platform and any host container, such as VB, PowerBuilder, or Siebel.

Overview

Clients connect to CTI OS through a client interface library. The library is responsible for establishing and automatically recovering connection failures, even if it must switch between CTI OS servers. It provides a set of wrapper classes to support COM, C++, and C applications.

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

- In Agent Connection, the client receives messages for a specific agent, such as calls and agent state changes;
- In Monitor Mode, the client provides a filter expression that selects the types of events that are forwarded to the application. These event messages can be used to provide notification of specific call events, agent state changes, or statistics.

The client application interacts with CTI OS through Session, Agent, Skill Group, and Call objects. The object model is shown in Figure 1-1.
Agent and Monitor Mode connections apply regardless of the type of customer interaction. They apply to any and all type of interactions and serve to select the type of messages sent to the client application. To connect in Agent Mode, the client application must login as an agent. To connect in Monitor Mode, the application specifies a message filter.

To connect:

```vbs
Dim m_sessionresolver As CTIOSSESSIONRESOLVERLib.SessionResolver
Dim WithEvents m_session As CTIOSCLIENTLib.Session
Dim m_Agent As CTIOSCLIENTLib.Agent

Sub ConnectToServer()

    ' Get a session to work with
    Set m_session = m_sessionresolver.GetSessionRef(""")

    ' Builds connect parameters
    Dim m_Args As New Arguments
```

Figure 1-1  CTI OS Client Interface Object Model
' Side A information
m_Args.AddItem "CtiOsA", machinename_A
m_Args.AddItem "PortA", 42028

' Side B information
m_Args.AddItem "CtiOsB", machinename_B
m_Args.AddItem "PortB", 43028
m_Args.AddItem "Heartbeat", 5

' Get ready to submit connection request
Dim vRequestParam As Variant
Set vRequestParam = m_Args
'sends connection request
m_session.Connect vRequestParam

End Sub

To set Agent Mode:

Sub EstablishAgentMode()
Dim m_Args As New Arguments

    m_Args.AddItem "Agentid", "23840"
m_Args.AddItem "Instrument", "23801"
m_Args.AddItem "PeripheralID", 5000

    ' Create a new agent
Set m_Agent = New Agent
' intialize the agent object using setValue
m_Agent.SetValue "Agentid", "23840"
m_Agent.SetValue "Instrument", "23801"
m_Agent.SetValue "PeripheralID", 5000

    ' Tell the session about the agent to establish agent mode
connection
m_session.SetAgent m_Agent

Dim vRequestParam As Variant
Set vRequestParam = m_Args
'send login request
m_Agent.Login vRequestParam

End Sub

To set Monitor Mode:

Sub EstablishMonitorMode()
Dim strFilter As String
Dim m_Args As New Arguments


Extensibility and Ease of Use

CTI OS class definitions are designed for extensibility. Each method definition supports a variable number of arguments. The design is intended to allow future revisions to add additional arguments when needed without requiring new methods or new classes to be defined.

For almost all arguments to method calls, the argument can be in one of the following forms:

- an integer value;
- a string value;
- a string containing one or more key/value pairs; or
- an array of key/value pairs using the Arguments class.

MakeCall is a typical method that takes advantage of optional arguments. The Arguments class is a helper class used to build key/value pairs. For example:

```vbscript
Dim agent As New ctios.agent
Dim CallArgs As New Arguments

' any of the following is valid
agent. MakeCall "23814" ' string
agent. MakeCall "DN=23814" ' key/value string
agent. MakeCall 23814 ' integer

' array of arguments is possible
CallArgs. AddItem "DN", "23814"
CallArgs. AddItem "CallVariable1", "Y"
CallArgs. AddItem "ExtDataArg", "Agent N"
CallArgs. AddItem "UserToUser", "from agent 23813"
agent. MakeCall CallArgs ' CallArgs cleared after call

' multiple key/values defined as a string
agent. MakeCall "DN=23814;CallVariable1='Y'; ExtDataArg='Agent N';UserToUser='from agent 23813'"
```
agent. MakeCall
"DN=" + agent_id + "; " + 
"CallVariable1=" + callvariable_item + "; " + 
"ExtDataArg=" + agent_identifier + "; " + 
"UserToUser=" + user_message

' this may be more efficient
CallArgs. AddItem eDN, "23814"
CallArgs. AddItem eCallVariable1, "Y"
CallArgs. AddItem "ExtDataArg", "Agent N"
CallArgs. AddItem eUserToUser, "from user 23813"
agent. MakeCall CallArgs

C++ has a slightly more flexible syntax:

Arguments &args;
Arguments::CreateInstance()
args[ eDN ] = "23814";
args[ eCallVariable1 ] = "Y";
args[ "ExtDataArg" ] = "Agent N";
args[ eUserToUser ] = "from user 23813";
agent. MakeCall( args );
args.Release()

The arguments to method calls can be in one of the following forms:

- An integer
- A string
- A string containing key/value pairs
- An array of key/value pairs

MakeCall is a typical method that takes advantage of optional arguments. The Arguments class is a helper class used to build key/value pairs.
Accessing Properties with GetValue

Properties are set by events and can be retrieved through the GetValue method. To retrieve one or more property values, the client application passes a name or an array of property names to the GetValue method. The method returns either a single value or an array of key/value pairs containing the requested properties. For example:

```vbnet
Dim argValue As Arg
Dim argAllCalls as Arguments
Dim nPeripheralId as Integer

'Returns extension number in an Argument object
Set argValue = m_Agent.GetValue("Extension") ' Using property name
Or
Set argValue = m_Agent.GetValue(eExtension) ' Using Property enumerated value
```

### Table 1-1  Client Interface Argument Types

<table>
<thead>
<tr>
<th>Argument Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>String or Key/value string</td>
<td>May contain a single or multiple values. Semi-colons separate multiple values. String may contain key/value pairs when in the form key=value. Semi-colons separate multiple key/value pairs. Values may contain spaces. All leading and trailing spaces are removed unless encapsulated in quoted string.</td>
</tr>
<tr>
<td>Integer</td>
<td>May only represent a single value</td>
</tr>
<tr>
<td>Array of key/value pairs</td>
<td>May be an Arguments object. Arguments is a collection of key/value pairs supplied as a helper class for CTI OS clients. May be an array of Variants. For C++, may use STL map.</td>
</tr>
</tbody>
</table>

---

**Cisco ICM Software CTI OS Developer’s Guide**

OL-1392-01

1-7
'GetValue from Arg returns the correct value using the appropriate
data type
MsgBox "Agent Extension = " & argValue.GetValue

Set argsAllCalls = m_session.GetValue("Calls")

The method returns a reference to an Argument or Arguments object. The
Arguments classes are helper classes that encapsulate strings, integers, and array
of values.

COM interface:
Dim argValue As Arg
Dim argAllCalls as Arguments

Set argValue = m_Agent.GetValue("Extension")
Set argsAllCalls = m_session.GetValue("Calls")

C++ interface:
Arg & argValue = m_Agent.GetValue(EXTENSION);
argValue.Release();
Arguments & argAllCalls = (Arguments &) m_session.GetValue("Calls") ;
argAllCalls.Release();

For clients that require a return value of a specific data type, use one of the
following methods: GetValueInt, GetValueString, GetValueArg, or GetValueObj.
They return integer, string, and argument array, respectively.

int n = GetValueInt( key );
string s = GetValueString( key );
args a = GetValueArg( key );
object obj = GetValueObj( key );
a.Release();

The signature(s) for the GetValue method are:

COM interface:
VARIANT GetValue ([in] LPVARIANT vProperty)

C++ interface:
Arg & GetValue(string & strProperty);
Arg & GetValue(char * strProperty);
Arg & GetValue(int strProperty);
GetElement

Some properties are arrays of values, such as Call Variables, ECC Array Variables, or number of call parties. Each element of an array is accessible using the GetElement method. The method is identical to GetValue, except that an additional index value argument is required.

```
Dim call As New CTIOSCLIENTLib.Call
Dim Properties As New Arguments
Dim CallVariable1 As String
Set Call = GetValue(“ActiveCall”)  
' any of the following is valid
CallVariable1 = call.GetElement(“CallVariables”, 1)  
CallVariable1 = call.GetElement(eCallVariables, 1)
```

Property Names and Enumerated Constants

Properties are identifiable by a unique name or enumerated constant. The list of properties and their format are listed in individual class definitions in the chapters that follow. The enumerated constant value for a property is equivalent to its name, except that it begins with the letter e, as in the following example:

```
' string versus enumerated constant
obj. GetValue(“CurrentServer”)  
obj. GetValue(eCurrentServer)
```

Property Attributes

It is possible to retrieve attribute information such as maximum length, format, and read-only status for any property at run time using the following syntax.

COM Interface:

```
VARIANT ICTiOSObjectClass::GetPropertyAttribute([in] LPVARIANT pPropName, [in] int nAttribute);
```

C++ Interface:

```
Arg& CICTiOSObject::GetPropertyAttribute(string& strPropName,  
enumCTIOSAttribute nAttribute);  
Arg& CICTiOSObject::GetPropertyAttribute(char * pstrzPropName,  
enumCTIOSAttribute nAttribute);
```
Chapter 1    Introduction

Property Attributes

Arg& CCIOSObject::GetPropertyAttribute( int nPropName,  
enumCTIOS_Attribute nAttribute );

' get type: string, integer, or boolean
nType = obj. GetPropertyAttribute( "property_name", "type" )

' if string, length is actual number of bytes, else returns 0
nLength = obj. GetPropertyAttribute( "property_name",  
CTIOS_ATTRIBUTE_LENGTH )

' size is valid regardless of type
' if boolean, 1. if integer, 4. if string, size is maximum number of
' bytes
nMaxBytes = obj. GetPropertyAttribute( "property_name",  
CTIOS_ATTRIBUTE_SIZE )

' can retrieve multiple attributes into an array
' but cannot get all attributes all at once
argArray = obj. GetPropertyAttribute( "AgentID;Extension;Instrument",  
CTIOS_ATTRIBUTE_SIZE )

Table 1-2    Property Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTIOS_ATTRIBUTE_TYPE = 0</td>
<td>One of following values: CTIOS_DATATYPE_STRING, CTIOS_DATATYPE_INTEGER,</td>
</tr>
<tr>
<td></td>
<td>CTIOS_DATATYPE_BOOLEAN, or CTIOS_DATATYPE_REFERENCE</td>
</tr>
<tr>
<td>CTIOS_ATTRIBUTE_LENGTH = 1</td>
<td>Current length of string type, otherwise 0.</td>
</tr>
<tr>
<td>CTIOS_ATTRIBUTE_SIZE = 2</td>
<td>Max size of string or size of data type. If string is of unlimited size,</td>
</tr>
<tr>
<td></td>
<td>returns the value CTIOS_DATASIZE_UNLIMITED = 32,767. Other defined types:</td>
</tr>
<tr>
<td></td>
<td>CTIOS_DATASIZE_BOOL = 1, CTIOS_DATASIZE_INTEGER = 4</td>
</tr>
</tbody>
</table>
Among the different platforms supported, not all of them support unsigned integers or shorts and there exist variations in how boolean values are handled. This section addresses how these differences are handled.

The CTI OS Client Interface Library does not use nor support unsigned shorts or unsigned integers. Any unsigned shorts or integers used or reported by CTI Server are converted to integers.

Booleans are represented as true and false and have a value of 1 or zero. COM applications represent booleans as type VT_BOOL and represent true and false as 0xFFFF and zero. There is good technical reason for expressing true as all ones, but it conflicts with the C/C++ definitions. When COM calls are made within the

---

**Table 1-2 Property Attributes (continued)**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTIOS_ATTRIBUTE_DEFINED_BY= 3</td>
<td>An identifier for which type of server connected to CTI OS has defined the properties of the constant. The only defined servers available at this time are: CTIOS_SERVERID_SYSTEM, CTIOS_SERVERID_CTISERVER</td>
</tr>
<tr>
<td>CTIOS_ATTRIBUTE_ISVALID = 4</td>
<td>Key. Returns true if the key is valid.</td>
</tr>
<tr>
<td>CTIOS_ATTRIBUTE_ISARRAY = 5</td>
<td>Returns true if value is an array or items.</td>
</tr>
<tr>
<td>CTIOS_ATTRIBUTE_NUMELEMENTS = 6</td>
<td>Number of elements in array or 1 if not array type.</td>
</tr>
</tbody>
</table>
CIL, boolean values are converted to (or from) VT_BOOL. COM applications such as VisualBasic, Internet Explorer, and other COM specific clients can treat TRUE as 0xFFFF.

The Client Interface Libraries for C and C++ maintain True and False as defined by the language, namely as 1 and 0.

**Accessing Call and ECC Variables**

Call, ECC variables, and arrays are passed as data with calls. These variables are accessible individually by name or as an array of values. Standard call variables 1 to 10 can be accessed as follows:

```vba
Dim org CallVariable as Arg
' access call variables
Set argCallVariable = call. GetValue( "CallVariable10" )
Set argCallVariable = call. GetValue( eCallVariable10 )

' ECC variables
Set argECCVar = call. GetValue( "ecc.acct_no" )' no equiv enumerated value
```

ECC variables in an object or an event parameter list are stored as an embedded arguments array that can be accessed in the following manner:

```vba
Dim argECCs As Arguments
Dim argECCVar as Arg
Dim argNamedArrayItem As Arg

Set argsECCs = call.GetValue("ECC")
Set argECCVar = argsECCs.GetValue("user.CustomerID")
Set argNamedArrayItem = argsECCs.GetValue("user.AccountNumber[1]")
```

ECC array variables are handled using special syntax. When the name of an ECC array variable is used, an array of values is returned. Variable and array names begin with the term “user” to distinguish ECC variable names from other CTI OS properties.

Adding a subscript to the reference can access each element of an ECC array variable as follows:
Chapter 1  Introduction

Handling Errors

'Accessing the n-th call variable
Set argCallVar = call.GetElement(eArrayCallVariables, n)

'Accessing ECC variable at offset "n"
Set argNamedArrayItem = call.GetElement("user.AccountNumber", n)
Set argNamedArrayItem = call. GetElement ("user.AccountNumber[\n]")

Special keywords are available to retrieve an array of all call variables, all ECC variables, or a specific named ECC array. Examples:
Dim argsCallVars As Arguments
Dim argsECCVars As Arguments
' access arrays
Set argsCallVars = call. GetValue( eArrayCallVariables )
Set argsECCVars = call. GetValue( "ECC" )
Set argsECCVars = call. GetValue( "user. name_arrayvariable" )

Handling Errors

The CTI OS Client Library provides a mechanism such that if any error occurs during the invocation of a method in an object; the method returns a numeric value that you can check using the macros CIL_SUCCESS (code) or CIL_FALIED (code). The method upon detecting the error will set the LASTERROR property in the object to the code that identifies the error. Any successful method invoked after the error will override the object’s property and will set it to CIL_OK. No exception is thrown, unless the method invoked returns a reference to an object and it failed to allocate memory for it.

try
{
    //Allocates arguments instance
    Arguments & arConnParams = Arguments::CreateInstance();

    arConnParams.AddItem(_T("PrimaryServer"), //Wrong Property Name
                         _T("MyCtiOSserver"));
    ...
    ...
    arConnParams.AddItem(_T("Heartbeat"), _T("MyCtiOSserver"));

    int nRetcode = m_Session.Connect(arConnParams);

    if(CIL_FAILED(nRetCode))
    {
        switch(nRetCode)
Event Driven Model

Most CTI OS Client Interface Library method calls send requests to a CTI OS server. In turn, these requests are forwarded to ACD, email or other servers. Any response can take from a few to several hundred milliseconds to several seconds. Waiting for a response would seriously impact the client’s performance.

Instead, method calls operate in asynchronous mode. The client application submits a request through a method call, then at some point in the future an event may be generated that indicates the success or failure of the request.

An event or response is not guaranteed. The operation may not result in any state change, may not create a telephony or other server response, may have failed outside in a link on the network outside of the CTI OS, or may have failed between the client application and CTI OS. The following guidelines and features will assist in diagnosing in real-time problems across servers:
Any request made that results in an error from CTI Server or other server are reported back to the client in the form of an OnError event.

The client system can request periodic heartbeats to CTI OS. When heartbeats are enabled, the library generates OnHeartbeat and OnMissingHeartbeat events that describe the quality and reliability of the connection. When heartbeats are missed, the connection to an active CTI OS server are automatically reestablished; CTI OS automatically monitors the quality and reliability of its connection to CTI Server and other servers using periodic heartbeats.

### Event Cascade Model

The CTI OS Client Interface Library uses an event cascade model. The model is designed to allow an application to override standard event handling for any event. Every event method uses the same signature: an event id code followed by a reference to an Arguments array containing all parameters passed.

The OnEvent method in the Session object receives every event. An application can override this method and replace it with its own implementation. It can then delegate to the standard behavior when and where appropriate. The standard implementation of the event handler in the Session object examines the event Id code, determines the event type, and forwards the event to the proper object's own OnEvent implementation. If the event is for an Agent object, it locates the proper Agent object and calls that object's OnEvent method. If the correct agent object does not exist, a new Agent object is created and the event is forwarded to that object's OnEvent implementation. The same logic is used for Call and SkillGroup objects.

The standard implementation of the OnEvent method for Agent, Call, and SkillGroup objects is to identify the event. Client applications can override the default implementation of the OnEvent method and thus intercept any event, giving it substantial control over flow, timing, and content. One benefit of this architecture is that it permits subclassing, as shown below. It permits the use of application specific custom objects to override the standard behavior where necessary. This model cascades events across object instances.
Event Publication Model

Clients can subscribe to an event interface for each class of objects. The published event interfaces are ISessionEvent, IAgentEvents, ICallEvents, ISkillGroupEvents, and IAllEvents. When an object needs to fire an event, it calls the Session’s FireEvent method, which then publishes the event to all subscribers. For example, as part of its OnEvent handler, the Call object fires an OnCallDelivered event. (This can also be overridden through subclassing; see the section entitled “Subclassing” in Chapter 2, “CTI OS Client Interface Library Architecture.”)

Subscribing to an Event Interface

The Session object manages all event subscription lists and the adding and removing of event subscribers. In C++ and COM, connection points are used (often implemented under the covers by the IDE).

Special Values

Throughout this document, special reference is made to values such as CallPlacementType, DeviceIDType, Reason code, and many others. These values are defined by CTI Server or other server. CTI OS passes these values through.

Start Up, Connect, Snapshot

A client receives call and agent events when it establishes a connection mode: Agent or Message filter. Unless one of these modes is selected, the client will not receive any events from any CTI OS server. The client establishes the connection using a two step process.

- The client connects to a CTI OS server using the Session object Connect() method.
- The client then establishes the connection mode. For Agent Mode, the client uses the Session object SetAgent() method. For Monitor Mode, the client uses the Session object SetMessageFilter() method.
The CTI OS server responds to the client’s SetAgent or SetMessageFilter request by sending an exact copy of the current agent and call states, using the following events:

- **eQueryAgentStateConf**: This event contains the current state of the agent in all the skill groups it belongs to.

- **eSnapshotDeviceConf**: This event contains a list of all the calls the agent is or was working on.

This information download allows a client application to have the immediate control over calls and the phone via software in the following situations:

- The agent logged in to the system using a hard phone and then launched the softphone.

- The softphone reconnected to the system after a failure.
CTI OS Client Interface Library Architecture

The CTI OS Client Interface Library (CIL) consists of the following components:

- **Object Interface Framework (OIF)** – The Object Interface Framework is the interface between the CIL and the client application.
- **Service** – The Service object (or Service layer) is isolates the OIF object from the Connection object so that neither object directly impacts the performance of the other. The Service object also converts message formats between that used by the Session (and its clients) and the format sent across the network.
- **Connection** – The Connection object (or Connection layer) monitors the CIL’s connection with the server. It also sends and receives server messages.

This chapter describes each layer of the CIL architecture.

Object Interface Framework

The CTI Object Interface Framework is the topmost layer on the CIL architecture. It consists of a group of object classes that enable an application developer 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.
Session Object

The Session object class provides the client with access to CTI Services offered by CTI OS. It is responsible for event distribution, object creation and connection maintenance. From an architecture perspective, the Session object is really a wrapper for four subobjects, which handle all the work.

- Session Manager
- Session Behavior
- Object Manager
- Event Publisher

Figure 2-1 depicts the Session object with all its components and relationships.

Figure 2-1 Object Interface Framework Block Diagram
Session Manager

The Session Manager is responsible for the internal state of the Session object. This state functionality includes making the connection, recovering from link failure, and specifying the Session mode.

Session Behavior

The session behavior is the set of public and private methods of the class that allows iteration between layers.

Connection

The Session Manager services requests to the Session object for connecting and disconnecting from the CTI OS Server. In addition, the Session Manager maintains the state of the connection based on events arriving in its OnEvent method.

Connection Failure and Recovery

The Session Manager object will be notified of any Connection Failure messages. In the event of a connection failure, the Session Manager will execute the reconnect algorithm to attempt to reconnect to an alternate CTI OS Server.

Session Modes

A Session object can be set to work one of two modes: Agent Mode or Monitor Mode. The Session Manager is responsible for maintaining the state of the Session mode. Once the client connects to the CTIOS server it cannot change its mode without closing the session.

Agent Mode

The client specifies its intent to work as an Agent or a Supervisor by creating an Agent object (or subclassing from one) and then calling SetAgent (Agent). The server is informed of the client’s mode selection via the
eSetSessionModeRequest. The Agent object contains a reference to the Session object for the purpose of sending requests to CTI OS Server. The client may not change agents or change modes once Connect is called.

**Monitor Mode**

The client specifies its intent to work in Monitor mode by calling SetMessageFilter. The server is not informed of the client’s mode selection, however, until the client calls Connect. From this point on, the Session can accommodate multiple agent, and calls for multiple agents. (Creation of object for multiple agents is explained in the next section.) The client may not change message filters or change modes once Connect is called.

**Object Manager and Event Passing**

The Object Manager is responsible for the creation and destruction of CTI OS objects. The main event entry into each of these objects (Session, Agent, Call, SkillGroup) is the OnEvent method. At startup, when the client creates a Session object, the Session creates the Service Layer object and passes a reference back to itself to the Service Layer. From that point forward, when new events arrive at the Message Receiver (in the Service layer), the Service Layer will deliver the event to the session via its OnEvent method. The arguments passed to OnEvent are EventID and an Argument array of key-value pairs. Each event is targeted to a particular object (Session, Agent, Call, SkillGroup, etc.). The event arguments contain a field UniqueObjectID that identifies the object that the event is directed to. Event arguments correspond to properties on the target object.

Within the Session’s OnEvent, the Object Manager performs the following steps using the EventID and UniqueObjectID that came in the event parameters:

- It determines whether the object identified by UniqueObjectID (Session, Call, Agent or SkillGroup) exists or not. If it exists, the object manager will return a reference to the target object. Otherwise, it will create a new object (see Creating Objects) for that UniqueObjectID and return a reference to the new object.
- Using the reference to the targeted object the Session object forwards the event to by invoking that object’s OnEvent method.
Creating Objects

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

An instance of an object is created when Object Manager determines that the object to which the UniqueObjectID in the event refers does not exist in any of the object lists it maintains. The creation process starts by determining what object type is required in order to invoke the appropriate object factory. Once the object is instantiated and initialized using the event arguments, the object is added to the collection it belongs to.

Lifetime of objects is controlled using reference counting. Any application that will hold a reference to an object it is required to perform and AddRef() on the object to gain ownership and when it does not longer needs the object it must execute Release(). This way the object manager will not release the object before any client is done with it. Next it is explained the lifetime for each of the standard CTI OS object types.

Call Object Lifetime

A Call object is created and its reference count is incremented in the handler for the OnCallBegin event. Any data available for the call that is passed in the OnCallBegin event is used to set up the Call objects initial state and properties. Upon receipt of the OnCallEnd event, the Session object forwards the event to the Call object via its OnEvent method in the usual manner. The Call object forwards the event to any subscribers. Call event subscribers should release their references to the Call object within their event handler for OnCallEnd to allow the Call object to be deleted. When the Call object’s OnEvent method returns after handling OnCallEnd, the object manager decrements the Call object’s reference count. If the reference count does not decrement to zero, then some client has not released its reference to the Call object. In that case, the Object Manager will log an error and remove the object from the calls collection. After the object was removed, no client will be able to access it from Session. Finally the object will destroy itself when the last client releases the reference it had.
Agent Object Lifetime

In Agent mode, the client will create an Agent Object (which causes its reference count to be incremented) and passes it to the Session in the `SetAgent` method. Since the Agent object already exists at the time that the `OnEvent` is first triggered, the existing Agent Object is updated with the arguments accompanying the `OnAgentStateChange` event.

In Monitor Mode objects are created and their reference counts incremented in the handler for an `OnAgentStateChange` event. When `OnEvent` receives an `OnAgentStateChange` event for an unrecognized Agent, that new Agent is added to the Session’s collection of agents. This new Agent will be created with the arguments accompanying the `OnAgentStateChange` event set as properties, and its initial state will be the state passed in the `OnAgentStateChange` event.

In Agent mode the `Session` object decrements the reference count on the agent. In Monitor mode the `Session` object cycles through its Agent collection and decrements the reference count of each Agent in the collection.

SkillGroup Object Lifetime

The SkillGroup Object is simply a container for SkillGroup statistics. A SkillGroup Object is created and its reference count is incremented the first time an `OnNewSkillGroupStatisticsEvent` event occurs for that Skill Group, and is subsequently updated by `OnNewSkillGroupStatisticsEvent` events.

In both Agent mode and Monitor mode the `Session` object decrements the reference count on the SkillGroup object when the `Session` object is destroyed. In its destructor, the `Session` cycles through its SkillGroup collection and decrements the reference count of each SkillGroup in the collection.

Object Factory

An object factory is a special method in CIL for C++ that has to be declared in any class derived from `CCtiOsObject` class. The factory is declared in the class definition using `DECLARE_CIL_OBJECT_FACTORY()` and implemented in the class module using `IMPLEMENT_CIL_OBJECT_FACTORY(Classname)`. By default CIL provides standard object factories for `CCall`, `CAgent`, `CSkillGroup` and `CWaitObject`. For more details on the use of object factories, see the section entitled Subclassing.
Event Publisher

The Session’s Event Publisher component is based on the publisher-subscriber design pattern, and is responsible for firing events to subscribing clients of the Session.

Exposed Interfaces

The following event interfaces are exposed to the client.

- ISessionEvents (e.g. OnConnection, OnConnectionClosed)
- IAgentEvents (e.g. OnAgentStateChange, OnNewAgentStatisticsEvent)
- ICallEvents (e.g. OnCallBegin, OnCallDelivered, OnCallDataUpdate)
- ISkillGroupEvents (e.g. OnNewSkillGroupStatisticsEvent)
- IAllInOne (all events)

Adding and Removing Subscribers

The Event Publisher manages lists of all clients to notify for each category of event. To subscribe for the events of a particular interface, a client must call Session’s AddXYZEventListener(...) method. The argument passed to this method is a callback handle to the subscriber’s own implementation of the interface. Now, a client to drop its subscription to events it registered before. It must call Session’s RemoveXYZEventListener(...) method. The XYZ is the name of the event interface to be removed.

Event Distribution

When a CTI OS Object receives its OnEvent, it will first update its internal state, and decide whether to fire the event to clients. If the object decides to fire the event, it will call the Session’s FireEvent. This last method forwards the request to the Event Publisher. The Event Publisher then determines what kind of event this event is and fires the event on the appropriate interface(s).
Service Layer

The service layer sits between the OIF and the connection layer. Its main purpose is to translate between the high-level command/event message structure of the upper CIL and the low-level network message structure of the connection layer. 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.

Connection Layer

The purpose of the connection layer is to provide a low-level connection management mechanism between the CIL and CTI OS. It sits as the bottom tier of the CIL’s layered architecture. A layer at this level allows the CIL to decouple the higher-level event and message architecture from the handling and specifics of a low-level communication link such as TCP/IP sockets.

The Connection Layer provides basic communication and connection recovery facilities. This layer has no knowledge about the format or meaning of CTI OS messages. It simply receives a buffer and sends it to the other end.

Multithreaded CTI OS Client Application

A CTI OS Client Application can be designed to run with multiple threads, such that, one thread can be use to process a single event type while the remaining threads process all the other types.

To write a multithreaded client, the application programmer will need to adhere to the programming model described in this document.
Support for Multithreaded Client in CIL

To accommodate multithreaded applications, the CTI OS Client Interface library provides a wait object per thread that will be signaled whenever a new event is sent to the client application.

The architecture shown in Figure 2-2 guarantees consistency with the server because all CTI OS objects are updated before the client knows about the event.

**Figure 2-2  Multithreaded Application Support**

A Wait Object is responsible for signaling a waiting thread when one of the events specified on the mask arrived from the server. The object class maintains the event mask and the operating system object on which the thread waits on. The system object type will depend on the platform. It can be an Event or Mutex under Win32 and a Semaphore or Signal under UNIX. The following sections outline the properties and methods exported by the class.
Properties

Table 2-1 lists the Wait Object properties. All the properties listed are private to the class.

**Table 2-1  Wait Object Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThreadId</td>
<td>Double word type variable to store the system assigned thread ID.</td>
</tr>
<tr>
<td>EventMask</td>
<td>Integer type variable to store the event mask for which the thread will be waiting on</td>
</tr>
<tr>
<td>WaitObject</td>
<td>Handle type variable to store the handle to the system’s wait object.</td>
</tr>
</tbody>
</table>

Methods

Table 2-2 lists the Wait Object methods. All the methods listed belong to the public user interface.

**Table 2-2  Wait Object Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InMask(int iEventId)</td>
<td>Returns true if iEventId is part of the event mask.</td>
</tr>
<tr>
<td>GetMask()</td>
<td>Returns the Event Mask currently in use</td>
</tr>
<tr>
<td>SetMask(int iEventMask)</td>
<td>Sets the list of events for which the synchronization object will signal the awaiting thread.</td>
</tr>
<tr>
<td>SignalEvent (int iEventId)</td>
<td>Causes the system object to be signaled if iEventId is part of the event mask.</td>
</tr>
<tr>
<td>WaitOnMultipleEvents()</td>
<td>Makes the current thread to go to sleep until one of the events in the mask arrives and the system object is signaled.</td>
</tr>
</tbody>
</table>
Usage at the Client

On a multithreaded application, you must create a Wait Object for each thread that uses CTI OS Objects.

In order to create an instance, the application must call CreateWaitObject at the beginning of the thread, then call SetEventMask to specify the events to wait for. To make the thread stop for events, then call WaitOnMultipleEvents via the Wait Object. Before the thread is terminated, to destroy the object, call DestroyObject.

```cpp
DWORD WINAPI WorkerThread(LPVOID lpParameter)
{
    CAgent * pAgent = (CAgent *) lpParameter;
    Arguments & argsWaitParams = Arguments::CreateInstance();

    CWaitObject * pWaitCall = pSession->CreateWaitObject(argsWaitParams);
    pWaitCall.SetMask(eCallDeliveredMask | eCallEstablished | eCallControlFailureMask);

    pAgent->MakeCall("DN=1234");
    pWaitCall->WaitOnMultipleEvents();
    pSession->DestroyObject(pWaitCall);
    argsWaitParams.Release();

    return TERMINATED_OK;
}

Other example:

DWORD WINAPI WorkerThread(LPVOID lpParameter)
{
    CAgent * pAgent = (CAgent *) lpParameter;
    Arguments & argsWaitParams = Arguments::CreateInstance();
    int nEventMask = eCallDeliveredMask | eCallEstablishedMask | eCallControlFailurecaseMask;

    argsWaitParams.AddItem(EVENTMASK, nEventMask);
```
```c
CWaitObject * pWaitCall = pSession->CreateWaitObject(argsWaitParams);
pAgent->MakeCall("DN=1234");
pWaitCall->WaitOnMultipleEvents();
pSession->DestroyObject(pWaitCall);
argsWaitParams.Release();
return TERMINATED_OK;
}
```

### Multithreaded Application Example

**Figure 2-3** shows a client application with two threads. The main client’s thread deals with agent activity and call management. The second, updates a corporate database used for account billing.
This example shows clearly how an application can take advantage of workload distribution by using different threads to do the work. CIL’s programming model is easy to use and does not compromise extensibility. Moreover, the architecture guarantees that all CTI OS objects remain consistent during the lifetime of the application. CIL will only signal Wait Objects after the event owners (CTI OS Objects) processed the event(s).

Although CIL is very robust, the application programmer is still responsible for executing the following steps in order.
## Subclassing

Subclassing is a mechanism that allows an application to replace CIL’s standard implementation of an object with its own. Without subclassing, an application is required to contain our object within their object and it is also required to define a separate object to receive events. Subclassing eliminates this additional overhead. Subclassing is particularly useful for the Call or Agent object. Subclassing is only supported under C++.

To use subclassing an application programmer must follow the following steps:

### Step 1
Create a Wait Object at the beginning of each thread of the application that will use references to CTI OS Objects.

```cpp
CWaItObject * pWaitCall = pSession->CreatWaitObject(argsWaitParams);
```

### Step 2
Before calling WaitOnMultipleEvents, set the mask of events for which the thread will wait for.

```cpp
pWaitCall->SetMask(eCallDeliveredMask);
```

### Step 3
Execute action on CTI OS Object that will result on events.

```cpp
pAgent->MakeCall("DN=1234");
```

### Step 4
Use the Wait Object created at the beginning of the thread to call WaitOnMultipleEvents.

```cpp
pWaitCall ->WaitOnMultipleEvents();
```

### Note
Never use wait objects other than those created in the current thread.

### Step 5
Destroy all Wait Objects created on a thread before it is terminated.

```cpp
pSession->DestroyObject(oWaitCall);
argsWaitParams.Release();
```
Step 1  Create a class that derives from any of the standard OIF classes, as shown in the following example.

class CSupportCall : public CCall
{
    //Class members
    private:
        long m_tTalkingTime;
        short m_nPriority;
        CAduitTrail m_objAuditInfo;
    protected:
        public:
    //Constructors and Destructors
        CSupportCall();
        ~CSupportCall();
    //Class Methods
        private:
        protected:
    //Class public interface
        public:
            virtual long etTalkingTime();
            virtual void SetPriority();

    //Events
        public:
            virtual void OnEvent(int _iEventID, Arguments & rEventParam); //Overides CCall::OnEvent

    //Event Handlers
        protected:
            void OnCallBegin(Arguments & rEventParam);
            void OnEnd(Arguments & rEventParam);
    //Object Factory declaration
        DECLARE_CIL_OBJECT_FACTORY();
};

Step 2  Declare an object factory for this class and implement the factory in the class module, as shown in the following example.

#include "CIL.h"

//Implement Class factory
IMPLEMENT_CIL_OBJECT_FACTORY(CSupportCall)

CSupportCall::CSupportCall()
{
    //members initialization goes here
} 
CSupportCall::~CSupportCall() 
{ 
    //cleanup goes here 
} 

////////////////////////////////////////////////////////////////////// 
// Process Events Received from CTI OS Server 
////////////////////////////////////////////////////////////////////// 
void CSupportCall::OnEvent(int iEventID, Arguments & rEventParam) 
{ 
    switch(iEventID) 
    { 
    case eCallBeginEvent: 
        OnCallBegin(rEventParam); 
        break; 
    case eCallEndEvent: 
        OnCallEnd(rEventParam); 
        break; 
    //other events 
    } 
    //Let default processing at the CIL to happen 
    CCall::OnEvent(rEventParam); 
} 

////////////////////////////////////////////////////////////////////// 
void CSupportCall::OnCallBegin(Arguments & rEventParam) 
{ 
    CILRefArg & rCILRefArg = (CILRefArg &) GetValue(CURRENTAGENT); 
    CAgent * pCurrAgent = (CAgent *) rCILRefArg.GetValue(); 
    m_objAuditInfo.TrackServiceRep(*pCurrAgent); 
    m_objAuditInfo.TrackStartTime(GetTime()); 
    m_objAuditInfo.TrackPriority(GetPriority()); 
    rCILRefArg .Release(); 
    pCurrAgent->Release(); 
}
### Step 3

Register the class factory for your new subclass with the instance of the Session object that will use this subclass instead of the default. It is important to remember that only the Session where the factory was registered knows about it. If the application uses more than one instance of a Session object it is required to register the new class on all the other Sessions so they can use it. It is recommended that all factories registrations be performed before any connection is open. This guarantees that CIL will use the new classes to create new object instances.

```cpp
#include <iostream.h>
#include "CIL.h"
#include "SupportCall.h"

void main(void)
{
    CCtiOsSession ctiSession;

    //Register Subclassed object factories
    string strFactoryTypeName = _T("Call");
    tCilObjectFactory newClassFactory = GET_CIL_OBJECT_FACTORY(CSupportCall);
    ctiSession.SetFactory(strFactoryTypeName, newClassFactory);

    //Continue with program
}
```

In the example, after SetFactory() was invoked, CIL registered the new factory for calls. When a call event arrives and a new object is needed, the object manager in Session will use the specified factory and instantiate a CCSupportCall object. Session will treat this object as a if it was a CCall. To deliver incoming events it will invoke OnEvent() on the object as usual. To illustrate, consider the implementation of OnEvent and assume that the event passed by Session to the object is eCallBeginEvent. In this case, CCSupportCall maintains an audit trail of who is handling the call, the time the call was assigned to the agent and sets the priority of the call based on company business rules. After that code is executed the object lets CIL to continue with the standard processing.
Subclassing
Handling Events

The CTI OS Client Library makes events available to applications using a publisher-subscriber design pattern. In this pattern, the client subscribes with CIL and specifies the type of events it is interested in. After the subscription is established in CIL, it will start forwarding events to the client application.

The subscription mechanism, in essence, is the same in all the environments supported by CTI OS. The few differences that exist are explained later in this chapter.

In order for a client application to be notified of events, it must first implement an object or objects that derive from any other CIL interfaces:

- IAllInOne, which contains event handlers for all the events that are processed in the CIL, the interface.
- IAgentEvents, which contains handlers for agent events.
- IButtonEnablementEvents, which contains handlers for GUI enablement events.
- ICallEvents, which contains handlers for call events.
- ISkillGroupEvents, which contains handlers for skill group events
- ISessionEvents, which contains handlers for session events.

Finally, the application subscribes the new interface objects with CIL such that messages can be forwarded.
Handling Events in C++

In order for a CIL C++ application to receive events, it must create a derivative class from one of the event interfaces and then register the object or objects with the session event publisher.

Creating a Subscriber Object Class

The event handlers in all the event interface classes in CIL for C++ are declared as pure virtual functions, such that a derived class will have to provide an implementation for each handler.

To facilitate this work CIL provides a set of adapter classes (Table 3-1) that implement event handlers as virtual functions with an empty body. It is preferable that application programmers create subclasses of the adapter classes rather than of the interfaces themselves.

Table 3-1  CIL Adapter Classes

<table>
<thead>
<tr>
<th>Adapter Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllInOneEventsAdapter</td>
<td>Provide the default implementation for the message handlers in IAllEvents</td>
</tr>
<tr>
<td>AgentEventsAdapter</td>
<td>Provide the default implementation for the message handlers in IAgentEvents</td>
</tr>
<tr>
<td>ButtonEnablementEventsAdapter</td>
<td>Provide the default implementation for the message handlers in IButtonEnablementEvents</td>
</tr>
<tr>
<td>CallEventsAdapter</td>
<td>Provide the default implementation for the message handlers in ICallEvents</td>
</tr>
<tr>
<td>SkillGroupEventsAdapter</td>
<td>Provide the default implementation for the message handlers in ISkillGroupEvents</td>
</tr>
<tr>
<td>SessionEventsAdapter</td>
<td>Provide the default implementation for the message handlers in ISessionEvents</td>
</tr>
</tbody>
</table>
Creating a Subscriber Object Class

The following example shows a class, which derives from CallEventsAdapter and implements a few message handlers.

```cpp
#include "CIL.h"

class MyCallEventSink : public CallEventsAdapter
{
public:
// Constructor
// Destructor

    // We are interested only on these events
    virtual void OnCallRequestFailed(Arguments & rArguments);
    virtual void OnCallQueuedEvent(Arguments & rArguments);
    virtual void OnCallDequeuedEvent(Arguments & rArguments);
    virtual void OnCallReachedNetworkEvent(Arguments & rArguments);
};

    // Notify the user and undo database transaction
    void MyCallEventSink :: OnCallRequestFailed(Arguments & rArguments)
    { pWallBoard->NotifyUI(rArguments);
      pDataBase->RollBack(m_CurTransaction);
    }

    // Registers arrival to queue and starts tracking of mean wait until service
    // For this call
    void MyCallEventSink :: OnCallQueuedEvent(Arguments & rArguments)
    { pAuditTrail->StartTrackingMeanWait(rArguments);
    }

    // Registers departure from queue and stops tracking of mean wait until service
    // For this call
    void MyCallEventSink :: OnCallDequeuedEvent(Arguments & rArguments)
    { pAuditTrail->StopTrackingMeanWait(rArguments);
    }

    // Keeps track of the calls that end up leaving the call center
    void MyCallEventSink :: OnCallReachedNetworkEvent(Arguments & rArguments)
    { pAuditTrail->RegisterDeparture(rArguments);
    }
```
Registering a C++ Subscriber Object

CIL only forwards events to a client application after an event subscriber object is registered. The registration of subscribers in CIL for C++ takes place via the Session object. The methods available from CCtiOsSession are as follows:

```c
int AddAllInOneEventListener(IAllInOne * pAllInOneEvents);
int AddAgentEventListener(IAgentEvents * pAgentEvents);
int AddButtonEnablementEventListener(IButtonEnablementEvents * pButtonEvents);
int AddCallEventListener(ICallEvents * pCallEvents);
int AddSkillGroupEventListener(ISkillGroupEvents * pSkillGroupEvents);
int AddSessionEventListener(ISessionEvents * pSessionEvents);
```

### Input Parameters

- **pAllInOneEvents**
  - Points to an IAllInOne subscriber object.
- **pAgentEvents**
  - Points to an IAgentEvents subscriber object.
- **pButtonEvents**
  - Points to an IButtonEnablementEvents subscriber object.
- **pCallEvents**
  - Points to an ICallEvents subscriber object.
- **pSkillGroupEvents**
  - Points to an ISkillGroupEvents subscriber object.
- **pSessionEvents**
  - Points to an ISessionEvents subscriber object.

### Return Value

If successful it returns CIL_OK, otherwise it returns E_CTIOS_INVALID_ARGUMENT.
Example

CMyCtiApplication::StartSubscriptionToEvents(void)
{
    //Creates subscriber object
    m_pCallSubscriber = new MyCallEventSink();

    //Register Subscriber
    m_ctiSession.AddCallEventListener ((ICallEvents *)
    m_pCallSubscriber);

    //More things to initialize
}

Unregistering Subscriber

In order to stop the flow of events to the client, the application has to remove the
subscriber from the session event publisher. The C++ methods provided by the
CCtiOsSession object are as shown:

int RemoveAllInOneEventListener(IAllInOne * pAllInOneEvents);
int RemoveAgentEventListener(IAgentEvents * pAgentEvents);
int RemoveButtonEnablementEventListener(IButtonEnablementEvents * pButtonEvents);
int RemoveCallEventListener (ICallEvents * pCallEvents);
int RemoveSkillGroupEventListener (ISkillGroupEvents * pSkillGroupEvents);
int RemoveSessionEventListener (ISessionEvents * pSessionEvents);

Input Parameters

pAllInOneEvents
    Points to an IAllInOne subscriber object to be removed.
pAgentEvents
    Points to an IAgentEvents subscriber object to be removed
pButtonEvents
    Points to an IButtonEnablementEvents subscriber object to be removed
pCallEvents
   Points to an ICallEvents subscriber object to be removed
pSkillGroupEvents
   Points to an ISkillGroupEvents subscriber object to be removed
pSessionEvents
   Points to an ISessionEvents subscriber object to be removed

Return Value

If successful it returns CIL_OK, otherwise it returns
E_CTIOS_INVALID_ARGUMENT.

Example

CMyCtiApplication::RevokeSubscriptionToEvents(void)
{
   //UnRegister Subscriber
   m_ctiSession.RemoveCallEventListener ((ICallEvents *)
   m_pCallSubscriber);
   //More things to initialize

Handling Events in COM

COM CIL client applications can be classified in two different groups:
   • Automation based Applications.
   • COM (ATL, MFC, COM SDK) based Applications.
Automation Based Applications

The clients in this group include those applications built with MS Visual Basic, MS Access, MS Fox Pro, MS Office, and other development environments that make use of ActiveX controls and COM objects via Automation (IDispatch interface). By definition a COM Server Object that supports Automation will fire events to its Automation clients through one interface only. For this reason CIL for COM exposes _IAllEvents for all its automation clients to use. The following Visual Basic example shows how an Automation client application will subscribe for events with the session object.

' 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

Private Sub m_Session_OnCallBegin(ByVal pDispParam As Object)
    LogEvent "OnCallBegin", pDispParam
End Sub

Private Sub m_Session_OnCallCleared(ByVal pDispParam As Object)
    LogEvent "OnCallCleared", pDispParam
End Sub

Private Sub m_Session_OnCallConferenced(ByVal pDispParam As Object)
    LogEvent "OnCallConferenced", pDispParam
End Sub

ATL, MFC and COM SDK Based Applications

The clients in this group are all those applications built using ATL, MFC, or Common object Model SDK. These applications are usually written in C++. Client applications are usually created this way to take advantage of the performance improvements that are obtained by using directly the v-tables and IUnknown objects. Applications of this type tend to be more complex but faster.
In this model client applications subscribe for events with a COM server object by registering an event sink instance in the client with the connection point at the event interface from which events are expected. This is the standard protocol defined in MS COM SDK. If a COM server provides more than one event interface a client is allowed to connect to any of them.

The COM Session object publishes the following interfaces:

- IAllEvents
- _IAgentEvents
- _IButtonEnablementEvents
- _ICallEvents
- _ISkillGroupEvents
- _ISessionEvents

When building a COM client application, an application programmer is free to choose what interface or interfaces it wants the application to subscribe to. Then, a CTI OS client application will be able to connect any of the interfaces published by the COM session. After the application executes, it must drop its registration.

The following example shows how an ATL application will receive events for call and Agents only.

```cpp
//Call Event Sink
class ATL_NO_VTABLE CSinkCallEvents :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CSinkCallEvents, &__uuidof(_ICallEvents)>,
    public IDispatchImpl<_ICallEvents,
                        &__uuidof(_ICallEvents),
                        &LIBID_CTIOSCLIENTLib>,
    public IDispEventImpl<1,
                        CSinkCallEvents,
                        &__uuidof(_ICallEvents),
                        &LIBID_CTIOSCLIENTLib, 1, 0>
{    //Class definition
};

//Agent Event Sink
class ATL_NO_VTABLE CSinkAgentEvents :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CSinkAgentEvents, &__uuidof(_IAgentEvents)>,
```
public IDispatchImpl<_IAgentEvents, 
&_uuidof(_IAgentEvents), 
&LIBID_CTIOSCLIENTLib>, 
public IDispEventImpl<1, 
CSinkAgentEvents, 
&_uuidof(_IAgentEvents), 
&LIBID_CTIOSCLIENTLib, 1, 0>
{ 
  //Class definition
};

Registering Event Sink

//Register with the Session Object for Call Events 
hr = CComObject<CSinkCallEvents>::CreateInstance( &m_pSinkCallEvents );
_ASSERTE(SUCCEEDED(hr));
m_pSinkCallEvents->AddRef();
m_pSinkCallEvents->RegisterCallbackClass( this );
hr = m_pSinkCallEvents->DispEventAdvise(m_pSession);
_ASSERTE(SUCCEEDED(hr));

//Register with the Session Object for Agent Events 
hr = CComObject<CSinkAgentEvents>::CreateInstance( 
&m_pSinkAgentEvents );
_ASSERTE(SUCCEEDED(hr));
m_pSinkAgentEvents->AddRef();
m_pSinkAgentEvents->RegisterCallbackClass( this );
hr = m_pSinkAgentEvents->DispEventAdvise(m_pSession);
_ASSERTE(SUCCEEDED(hr));

Unregistering Event Sink

if (m_pSession)
{
  // Stop event notification for ICallEvents 
  hr = m_pSinkCallEvents->DispEventUnadvise( m_pSession );
  _ASSERTE(SUCCEEDED(hr));
m_pSinkCallEvents->Release();
m_pSinkCallEvents = NULL;

  // Stop event notification for IAgentEvents 
hr = m_pSinkAgentEvents->DispEventUnadvise( m_pSession );
  _ASSERTE(SUCCEEDED(hr));
}
m_pSinkAgentEvents->Release();
m_pSinkAgentEvents = NULL;
}
The session object establishes and maintains a connection to an active CTI OS server. The object provides two main functions: automatically recovery from connection failures and filtering and distributing events. The session object acts as a class factory for agent call and skill group objects. The session object receives all events from a CTI OS connection. It interprets the event, determines whether it applies to an agent, an existing call, or whether it must create a new call object.

Typically, an application has a single instance of the session object. However, there are no restrictions on the number or types of session connections from either a single application or workstation. It is possible, and sometimes desirable, to establish and manage multiple independent sessions. A restriction on agent login applies. An agent may only be logged in once.

If there is more than one session object monitoring the same agent or call, each session object will receive its own events. There is no guarantee on the order of event receipt when there are multiple session objects.

The session object creates new objects upon receipt of an event only if the targeted object does not exist. The session object maintains a list of agents, calls, subgroups, and wait objects. It is important that the client application not delete the objects; that makes the object reference invalid and can lead to unpredictable results. In COM and C++ applications, all CTI OS objects support reference counting. An application must release the reference count instead of deleting the object.

If the session is released and terminated, the connection to CTI OS server is dropped. Each object maintained by the session object (that is, any Agent, Call, Skill Group, or Wait object) will be released. As long as no other references are used by the client application, all objects are automatically removed from memory.
Working With CTI OS Objects

A client application can access the objects maintained in the session using the GetValue method. The application can obtain a list containing references to all the objects, a list of a objects of particular type and a reference to one single object.

First, to get a list of all objects in a session, the client application would include code similar to the following:

```vbnet
Dim argsAllObjects As Arguments
Dim argsAgents As Arguments
Dim argsCalls As Arguments
Dim argsSkills As Arguments
Dim strUOID As String

'Gets all the references from Session grouped by object type
Set argsAllObjects = m_session.GetValue("ObjectReferences")
'Access each reference collection
Set argsAgents = argsAllObjects.GetValue("Agents")
Set argsCalls = argsAllObjects.GetValue("Calls")
Set argsSkills = argsAllObjects.GetValue("SkillGroups")

'Loop through Calls
Dim ctiCall As CTIOSCLIENTLib.Call
Dim sCallData As new Arguments
For nI = 1 To argsCalls.NumElements
    Set ctiCall = argsCalls.GetElement(nI)
    sCallData.AddItem "CallVariable1", Format(Time, "Long Time")
    sCallData.AddItem "user.CustSrvAgent", "CtiOSClient"
    ctiCall. SetCallData sCallData
    Set ctiCall = Nothing
Next
```

This example shows that Session returns the reference to all the objects in an embedded arguments array whose keys are “Agents”, “Calls” and “SkillGroups”. The values corresponding to those keys are arguments arrays that use the UniqueObjectId of each object as key and for value the actual reference to the object.

If the client application is interested only on all the calls available at a given point in time, it retrieves them from session as follows:

```vbnet
Dim argsCalls As Arguments
Dim strUOID As String
```
Chapter 4  Session Object

Working With CTI OS Objects

'Gets all calls from Session
Set argsCalls = m_session.GetValue("Calls")

'Loop through Calls
Dim ctiCall As CTIOSCLIENTLib.Call
Dim sCallData As New Arguments
For nI = 1 To argsCalls.NumElements
    strUOID = argsCalls.GetKey(nI)
    Set ctiCall = argsCalls.GetElement(nI)
    sCallData.AddItem "CallVariable1", Format(Time, "Long Time")
    sCallData.AddItem "user.CustSrvAgent", "CtiOSClient"
    ctiCall. SetCallData sCallData
    'Releases the reference to the call
    Set ctiCall = Nothing
Next
'Releases reference to collection
Set argsCalls = Nothing

The collection of calls is returned as an arguments array in which the key is the UniqueObjectID of the object whose reference is returned as value when the argument array is accessed either with GetElement(index) or GetValue(key). The same procedure can be employed to retrieve Agents or Skill Groups.

Last, when the application only knows the UniqueObjectID of an object for which it needs a reference. It uses GetValue from session as described in the following example:

Private Sub m_session_OnCallDelivered(ByVal argEventParams As Object)
    Dim ctiCall As CTIOSCLIENTLib.Call
    Dim strUOID As String
    strUOID = argEventParams.GetValue("UniqueObjectID")
    'Compose string "UniqueObjectID=XXXXXX" to retrieve a Call reference
    strUOID = m_session.GetValue("UniqueObjectid")
    Dim strObjToRetrieve As String
    strObjToRetrieve = "UniqueObjectID=" & strUOID
    'Retrieve Call Reference and use it
    Set ctiCall = m_session.GetValue(strObjToRetrieve)
    'Pick up the call
    CtiCall.Aswer
    Set ctiCall = Nothing
End Sub
The application developer should note that the type of reference returned by GetValue() and GetElement() variates from environment to environment as follows:

**Table 4-1  Returned References**

<table>
<thead>
<tr>
<th>Environment</th>
<th>Reference Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>CILRefArg</td>
<td>Any method returning a reference to a CTI OS object will return a reference to the helper class named CILRefArg (See Chapter 8, “Helper Classes”) that contains the actual reference to a CAgent, CCall or CSkillGroup object instance.</td>
</tr>
<tr>
<td>COM</td>
<td>Idispatch</td>
<td>Any method returning a reference to a CTI OS COM object will return a reference to IAgent, ICall and ISkillGroup interfaces.</td>
</tr>
</tbody>
</table>

**Methods**

Table 4-2 lists the available session object methods.

**Table 4-2  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>GetPropertyAttribute</td>
<td>Retrieves attribute information for a specified session property.</td>
</tr>
<tr>
<td>GetValue (also GetValueInt, GetValueString)</td>
<td>Retrieves the value of a specified session property.</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>
</tbody>
</table>
Chapter 4  Session Object

Table 4-2  Session Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakeRequest</td>
<td>Sends a message request to the CTI OS Server</td>
</tr>
<tr>
<td>OnEvent</td>
<td>Specifies an action to take on occurrence of a specified event.</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>
</tbody>
</table>

Connect

The Connect method establishes a connection to a CTI OS server. The application must provide the name or TCP/IP address of at least one CTI OS server.

Syntax

C++

```cpp
int Connect(Arguments& rArguments);
```

COM

```cpp
HRESULT Connect( [in] VARIANT * pVariantArgs );
```

Input Parameters

rArguments

Reference to an arguments array containing the connection parameters listed in Table 4-3.
Table 4-3  Connect Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctios, or ctiosA</td>
<td>Name or tcp/ip address of a CTI OS server</td>
</tr>
<tr>
<td>CtiosB</td>
<td>Name or tcp/ip address of alternate CTI OS server.</td>
</tr>
<tr>
<td>Port, or portA</td>
<td>(optional) Tcp/ip port for ctiosA.</td>
</tr>
<tr>
<td>PortB</td>
<td>(optional) Tcp/ip port for ctiosB</td>
</tr>
<tr>
<td>Hb, or heartbeat</td>
<td>(optional) Heartbeat time, expressed in seconds.</td>
</tr>
</tbody>
</table>

pVariantArgs

Pointer to a variant that contains a pointer to an IArguments object containing the connection parameters listed on Table 4-3.

Return Values

C++

<table>
<thead>
<tr>
<th>Return Code</th>
<th>When Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIL_OK</td>
<td>If Successful</td>
</tr>
<tr>
<td>E_CTIOS_IN_FAIOVER</td>
<td>If the Session object has started the connection recovery algorithm</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_PROPERTY</td>
<td>When a parameter passed in the connection parameters is invalid</td>
</tr>
</tbody>
</table>

COM

If the method was able to connect it returns S_OK, otherwise it returns E_FAIL. To determine the error cause you can access the object’s LASTERROR property.
Examples

C++

```
try {
  Arguments &rArgSessionConn = Arguments::CreateInstance();
  rArgSessionConn.AddItem(CTIOSA, "localhost");
  rArgSessionConn.AddItem(PORTA, 42028);
  rArgSessionConn.AddItem(CTIOSB, "localhost");
  rArgSessionConn.AddItem(PORTB, 42028);
  rArgSessionConn.AddItem(HEARTBEAT, 5);

  nRetVal = ctiSession.Connect(rArgSessionConn);

  rArgSessionConn.Release();
  if (CIL_FAILED(nRetVal))
  {
    switch(nRetVal)
    {
      case E_CTIOS_IN_FAIlOVER:
        printf("Session is recovering from failure…");
        break;
      case E_CTIOS_INVALID_PROPERTY:
        printf("Failed to connect. Invalid parameter…");
    }
  }
}
```

COM C++

```
HRESULT hr = S_OK;
IArgumentsPtr arConnectArgs;
VARIANT vParam;
VariantInit(&vParam);
hr = arConnectArgs.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));
vParam.vt = VT_DISPATCH;
h = arConnectArgs->QueryInterface(IID_IDispatch,
(void **) &vParam.pdispVal);
m_pSession->Connect(&vParam);
```
arConnectArgs = NULL;

VB

' Prepares the connect request
Dim m_Args As New Arguments
m_Args.AddItem "CtiOsA", CStr(m_SideA)
m_Args.AddItem "PortA", m_PortA
m_Args.AddItem "CtiOsB", CStr(m_SideB)
m_Args.AddItem "PortB", m_PortB
m_Args.AddItem "Heartbeat", m_Heartbeat

Dim vRequestParam As Variant
Set vRequestParam = m_Args
m_Session.Connect vRequestParam

Set m_Args = Nothing

Disconnect

The Disconnect method disconnects the open connection to the CTI OS server. If there is more than one session object connected, only the object that requested the connection to be closed will be disconnected. The session object will not receive any additional events after disconnect.

Syntax

C++

void Disconnect ();

COM

HRESULT Disconnect ();

Parameters

None.
Return Values

C++

None.

COM

Always returns S_OK.

Examples

C++

ctiSession.Disconnect();

COM C++

m_pSession-> Disconnect();

VB

m_Session. Disconnect

GetPropertyAttribute

The GetPropertyAttribute method retrieves attribute information for any of the session properties listed in Table 4-4. For additional information on GetValue and GetPropertyAttributes, see Chapter 1, “Introduction.”
### Table 4-4  Session Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 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>CurrentPort</td>
<td>integer</td>
<td>Tcp/ip address of the current connected CTI OS server. May be port A or B.</td>
</tr>
<tr>
<td>ConnectedSince</td>
<td>integer</td>
<td>Time of day in milliseconds when connected.</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>
<tr>
<td>ConnectionMode</td>
<td>integer value</td>
<td>eAgentConnection, eMonitorConnection, or eNotConnected.</td>
</tr>
<tr>
<td>LastError</td>
<td>integer value</td>
<td>Last error code, if any. Otherwise this value is 0.</td>
</tr>
<tr>
<td>TryingServer</td>
<td>string</td>
<td>Contains the name or tcp/ip address of the server where a connection is being attempted. The value is blank if no connection is being attempted (see CurrentServer). The name of the server should be the name of CTI OS server A or B.</td>
</tr>
<tr>
<td>TryingPort</td>
<td>integer</td>
<td>Tcp/ip address of the server where a connection is being attempted. May be port A or B.</td>
</tr>
</tbody>
</table>
### Table 4-4  Session Properties (continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TryingSince</td>
<td>integer</td>
<td>Time of day in milliseconds when try began.</td>
</tr>
<tr>
<td>MessageFilter</td>
<td>string</td>
<td>Message expression.</td>
</tr>
<tr>
<td>Ctios, or 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>Port, or portA</td>
<td>integer</td>
<td>Tcp/ip port for ctiosA.</td>
</tr>
<tr>
<td>PortB</td>
<td>integer</td>
<td>Tcp/ip port for ctiosB.</td>
</tr>
<tr>
<td>Hb, or heartbeat</td>
<td>integer</td>
<td>Heartbeat time, expressed in seconds. If not set, default heartbeats are configurable on CTI OS server.</td>
</tr>
<tr>
<td>MaxHeartbeats</td>
<td>integer</td>
<td>Max heartbeats that can be missed before switching CTI OS servers. Default is 3 missed heartbeats.</td>
</tr>
<tr>
<td>ActiveCall</td>
<td>object reference</td>
<td>Valid only if in Agent Connect mode. When there is more than one call, this references the current active call. The current active call is the call just answered. For additional information, refer to ActiveCall in the Call object.</td>
</tr>
<tr>
<td>ObjectReferences</td>
<td>argument list</td>
<td>Array of object references maintained by the session object. Typically includes AgentReferences, CallReferences, and SkillGroupReferences. Can also include EmailReferences or Chat References.</td>
</tr>
<tr>
<td>AgentReference Array</td>
<td>agent object reference</td>
<td>Reference to Agent Array.</td>
</tr>
<tr>
<td>CallReferenceArray</td>
<td>call object reference</td>
<td>Reference to Call Array.</td>
</tr>
</tbody>
</table>
Syntax

GetPropertyAttribute( propertyname, attribute requested )

GetValue

The GetValue method retrieves the value for any property listed in Table 4-4. For additional information on GetValue and GetPropertyAttributes, see Chapter 1, “Introduction.”

Syntax

GetValue( key )

isAgentMode

The isAgentMode method checks if the current connection mode is agent mode.

Syntax

C++

bool isAgentMode ()

COM

HRESULT isAgentMode ([out, retval]
VARIANT_BOOL * pbAgentMode);

Call Skill Group Reference Array

Reference to Skill Group Array.

Call Object Reference Array

Reference to Call Array.
Parameters

None.

Return Values

**C++**

If the current session is on agent mode it returns true, otherwise it returns false.

**COM**

If the current session is on agent mode it returns VARIANT_TRUE, otherwise it returns VARIANT_FALSE.

Examples

**C++**

```cpp
if(m_pSession->isAgentMode())
{
    fnAutoLogin();
}
```

**COM C++**

```cpp
VARIANT_BOOL bRet = VARIANT_FALSE;

m_pSession->isAgentMode(&bRet);

    if(bRet == AVRIANT_TRUE)
    {
        fnAutoLogin();
    }
```

**VB**

```vb
bRet = m_Session.isAgentMode

    if bRet = True Then
        fnAutoLogin
    End If
```
The isSupervisorMode method checks if the current connection mode is supervisor mode.

**Syntax**

**C++**

```cpp
bool isSupervisorMode();
```

**COM**

```cpp
HRESULT isSupervisorMode ([out, retval] VARIANT_BOOL * pbAgentMode);
```

**Parameters**

None.

**Return Values**

**C++**

If the current session is on supervisor mode it returns true, otherwise it returns false.

**COM**

If the current session is on supervisor mode it returns VARIANT_TRUE. Otherwise it returns VARIANT_FALSE.
Examples

C++

```c++
if(m_pSession->isSupervisorMode())
{
    fnAutoLogin();
}
```

COM C++

```c++
VARIANT_BOOL bRet = VARIANT_FALSE;
m_pSession->isSupervisorMode(&bRet);
    if(bRet == AVARIANT_TRUE)
    {
        fnAutoLogin();
    }
```

VB

```vb
bRet = m_Session.isSupervisorMode
    if bRet = True Then
        fnAutoLogin
    End If
```

MakeRequest

The MakeRequest method sends a message request to the CTI OS Server. Any application or object that make use of the method is required to include as part or the parameters a UniqueObjectID that identifies what object is requiring the action. The objects created using CIL (e.g. Calls, Agents and Skill Groups) always include their UniqueObjectID as part of the message request by default.

Syntax

C++

```c++
void MakeRequest( int nRequestId, Arguments & rReqParam);
```
COM

HRESULT MakeRequest([in] int nRequestId,
[in] VARIANT *pReqParams)

Input Parameters

nRequestId
Enumerated value that identifies the command request to be executed by CTI
OS Server.

rReqParam
Reference to an arguments array that contains the request parameters.

pReqParams
Pointer to a variant that contains a pointer to an IArguments object that that
contains the request parameters.

Return Values

C++
None.

COM
Always returns S_OK.

Examples

C++

Args. AddItem("DialNumber", "1234");
Args. AddItem("UniqueObjectID",
ctiAgent.GetValueString("UniqueObjectID");

m_pSession->MakeRequest(eMakeCallRequest, Args);
Args.Release();
**COM C++**

```c++
HRESULT hr = S_OK;
IArgumentsPtr Args;
VARIANT vParam;
VariantInit(&vParam);
hr = Args.CreateInstance(OLESTR("CtiOsComArguments.ComArguments"));
Args. AddItem("DialNumber", "1234");
Args. AddItem("UniqueObjectID",
ctiAgent->GetValueString("UniqueObjectID"));
vParam.vt = VT_DISPATCH;
hr = Args ->QueryInterface(IID_IDispatch, (void **)&vParam.pdispVal);
m_pSession->MakeRequest(eMakeCallRequest, & vParam);
Args = NULL;
```

**VB**

```vb
' Prepares the connect request
Dim m_Args As New Arguments
m_Args. AddItem "DialNumber", "1234"
m_Args. AddItem "UniqueObjectID",
ctiAgent.GetValueString("UniqueObjectID")
Dim vRequestParam As Variant
Set vRequestParam = m_Args
m_pSession.MakeRequest eMakeCallRequest, Args
```
OnEvent (C++ Only)

The OnEvent method specifies an action to take on occurrence of a specified event. The session object receives every event. The default behavior of the session object is to examine the event type and route the event to the proper corresponding object: agent, call, or skill group object. If this represents a new agent, new call, or new skill group, it will create a new object and pass the event to the new object.

Syntax

virtual void OnEvent(int iEventID, Arguments & rEventParam);

Input Parameters

iEventID

Enumerated value that identifies the event received by CIL. Appendix CIL Messages.

rEventParam

Reference to an Arguments object that that contains the event parameters.

Return Values

None.

RequestDesktopSettings

The RequestDesktopSettings method sends a request to the CTI OS Server to download the configuration settings defined for a desktop application. A client application can request the download for either an Agent or a Supervisor desktop. The possible values for desktop_type are eDesktopTypeUnknown (-1), eAgentDesktop (0), and eSupervisorDesktop (1). The CTI OS Server responds to a successful request with an eGlobalSettingsDownloadConf event.
Chapter 4    Session Object

Syntax

RequestDesktopSettings( desktop_type )

SetAgent

The SetAgent method assigns an agent to the session. This is necessary to establish an Agent Mode connection. If no agent is set, the agent associated with the Agent object will be unable to login.

When the Agent is connected to the session object, the agent object's reference count is increased by one. The reference will be decreased by one when any of the following occurs: the session object is deleted; a new agent is set using SetAgent, or ResetAgent is called.

The current agent object reference can be retrieved using GetValue(“CurrentAgent”). The value is only set when a reference was set by SetAgent(). If no agent reference was set, the value returned in null.

Syntax

C++

int SetAgent(CAgent & ctiAgent);

COM

HRESULT SetAgent ( [in] LPDISPATCH pDispParam );

Input Parameters

ctiAgent

Reference to a CAgent object instance of the agent to be set.

pDispParam

Pointer to an IAgent object instance of the agent to be set.
Return Values

C++

If the agent was set in the session it returns CIL_OK, otherwise it returns CIL_FAIL.

COM

If the agent was set in the session it will return S_OK, otherwise E_FAIL.

Examples

C++

```cpp
CAgent ctiAgent;
ctiAgent.SetValue(AGENTID, "23840");
ctiAgent.SetValue(PASSWORD, "23840");
ctiAgent.SetValue(PERIPHERALID, 5000);
ctiAgent.SetValue(INSTRUMENT, "23801");

nRetVal = ctiSession.SetAgent(ctiAgent);

if (CIL_FAILED(nRetVal))
{
    printf(" Failed to set Agent. RetVal = %d...
", nRetVal);
    exit(1);
}
```

COM C++

```cpp
IAgentPtr m_pAgent;

m_pAgent.CreateInstance(OLESTR("CTIOSClient.ComAgent"));

m_pAgent->SetValue(AGENTID, "23840");
m_pAgent->SetValue(PASSWORD, "23840");
m_pAgent->SetValue(PERIPHERALID, 5000);
m_pAgent->SetValue(INSTRUMENT, "23801");

m_pSession->SetAgent(m_pAgent);
```
Chapter 4  Session Object

Methods

VB

Dim ctiAgent As New CTIOSClientLib.Agent
ctiAgent.SetValue "AgentID", "23840"
ctiAgent.SetValue "Password", "23840"
ctiAgent.SetValue "PeripheralID", 5000
ctiAgent.SetValue "Instrument", "23801"

nRetVal = ctiSession.SetAgent(ctiAgent);
If nRetVal = CIL_FAIL Then
    Print " Failed to set Agent. RetVal = " & nRetVal & Chr$(10) & Chr$(13)
End
End If

SetCurrentCall

The SetCurrentCall method associates the current call to a session object. It lets you set any call to the current call.

Syntax

C++

void SetCurrentCall(CCall * pCall);

COM

HRESULT SetCurrentCall ([in] LPDISPATCH pCallParam);

Input Parameters

pCall
    Pointer to the CCall to make current.
pCallParam
    Pointer to an ICall to make current.
Return Values

C++

None.

COM

Always returns S_OK

Examples

C++

```cpp
void CMyListBox::OnSelChange(...) {
    int nCurSel = this->GetCurSel();
    CCall * pCall = GetItemDataPtr(nCurSel);
    m_pSession->SetCurrentCall(pCall);
}
```

COM C++

```cpp
void CMyActiveListBox::OnSelChange(...) {
    int nCurSel = this->GetCurSel();
    ICall * pCall = GetItemDataPtr(nCurSel);
    m_pSession->SetCurrentCall((LPDISPATCH)pCall);
}
```

VB

```vb
Private Sub m_session_OnCallBegin(ByVal pDispParam As Object)
    ' Get UniqueObjectID to get the actual call and answer right away

    Dim m_uid As String
    m_uid = pDispParam.GetValue("UniqueObjectID")
```
Dim tmp As String
 tmp = "UniqueObjectID=" + m uid

' Get call object
Set m_call = m_session.GetValue(tmp)

m_session.SetCurrentCall(m_call)

m_call.Answer();

End Sub

SetMessageFilter

The SetMessageFilter method sets a message filter that the client application must pass in order to connect in Monitor Mode; the message filter is a string value that selects which agents and/or which system events to monitor. It is possible to monitor events on one or more agents, skill groups, or call events. To receive pre-route events, you must set up a Monitor Mode connection and filter on OnCallPreRoute.

A filter is a set of conditions that an event must meet in order to be sent to the client. It consists of zero or more logical operations performed on one or more expressions.

Operator

Logical operators allow a filter to specify more than one expression in one single filter. Table 4-5 lists the logical operators supported by CTI OS.

Table 4-5 CTI OS Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Logical AND</td>
</tr>
<tr>
<td>;</td>
<td>Logical AND</td>
</tr>
<tr>
<td>OR</td>
<td>Logical OR</td>
</tr>
<tr>
<td>NOT</td>
<td>Logical NOT</td>
</tr>
</tbody>
</table>
Expression

An expression is a single condition within a filter. An expression can consist of an event (such as OnCallBegin or OnCallEnd), or a keyword and its required value separated by “=”.

For example,

AgentID=222
CallVariable1=mike
SkillGroupNumber=22,23
Extension=23*

Keyword

A keyword is a string that is recognized by the server. Table 4-6 lists the most common CTI OS keywords and events.

Table 4-6  Common CTI OS Keywords and Events

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>SkillGroupNumber, SkillGoupPriority</td>
<td>OnNewSkillGroupStatistics</td>
</tr>
<tr>
<td>CallID, CallState, CallVariable1, ..., CallVariable10, <code>user.NamedVariable</code>, <code>user.NamedArray[Offset]</code>, UserToUser, WrapUp</td>
<td>OnCallPreRoute, OnCallBegin, OnCallDelivered, OnCallEstablished, OnCallHeld, OnCallReceived, OnCallTransferInit, OnCallTransferComplete, OnCallCleared, OnCallEnd</td>
</tr>
</tbody>
</table>

For a complete list see Appendix B, “CTI OS Keywords.”
Value

The value of a keyword may be expressed in several manners. It may be expressed explicitly as a single value:

222

It may be expressed as a list of allowed values separated by commas:

222,333,444

It may be expressed using the wildcard “*” to represent any character

22*, which means any character string beginning with 22.

It may be a combination of the preceding:

222,333,44*

Syntax

C++

int SetMessageFilter(string strFilter)

COM

HRESULT SetMessageFilter ([in] BSTR bstrFilter);

Input Parameters

strFilter

String text containing the message filter.

bstrFilter

String text containing the message filter.
Return Values

C++

If successful it returns CIL_OK. Otherwise it returns E_CTIOS_MODE_CONFLICT, indicating that the session cannot be set on monitor mode because the session is already in a mode different than monitor.

COM C++

Always returns S_OK. To determine the error cause, access the object’s LASTERROR property.

Examples

```cpp
string strFilter1, strFilter2, strFilter3, strFilter4, strFilter5;

//Filters events for all agents that start with 2381
strFilter1 = "AgentID=2381*";

//Filters events for skills groups 23 and 24
strFilter2 = "SkillGroupNumber=23,24";

//Filters only OnCallBegin events for the entire contact center
strFilter3 = "OnCallBegin"

//Filters messages SkillGroups and OnCallbegin
strFilter4 = "SkillGroupNumber=23,24;OnCallBegin"

//Filters messages all agents that start with 2381 or the bigining and end of a call strFilter5 = "(AgentID=2381*) OR (OnCallBegin AND OnCallEnd)"

C++

m_pSession->SetMessageFilter(strFilter4);
```
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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeOfDay</td>
<td>Integer value with time of day expressed in milliseconds.</td>
</tr>
<tr>
<td>CurrentServer</td>
<td>Name or tcp/ip address of the current connected CTI OS server.</td>
</tr>
</tbody>
</table>
### OnConnectionFailure

The OnConnectionFailure event is generated when an already established. 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 when connection has been successfully reestablished. The client application need not take any special action but may use it to display connection status.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeOfDay</td>
<td>Integer value with time of day expressed in milliseconds.</td>
</tr>
<tr>
<td>FailedServer</td>
<td>Name or tcp/ip address of the server that has failed to respond. See reason code.</td>
</tr>
<tr>
<td>ReasonCode</td>
<td>CTOIS_SERVER_CONNECTIONBROKEN, CTOIS_SERVER_MISSINGHEARTBEATS</td>
</tr>
</tbody>
</table>

### OnHeartbeat

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeOfDay</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.
Chapter 4  Session Object

### ISessionEvents 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 `CTIOS_SERVER_MONITORMODE`, or `CTIOS_SERVER_MONITORMODE_EXIT`.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeOfDay</td>
<td>Integer value with time of day expressed in milliseconds.</td>
</tr>
<tr>
<td>Mode</td>
<td><code>CTIOS_SERVER_MONITORMODE</code> or <code>CTIOS_SERVER_MONITORMODE_EXIT</code></td>
</tr>
</tbody>
</table>

#### OnConnectionClosed

The `OnConnectionClosed` message is generated when a connection is terminated.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageID</td>
<td>Integer, Message ID</td>
</tr>
</tbody>
</table>
OnConnectionRejected

The OnConnectionRejected message is generated when a connection is rejected.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventTime</td>
<td>Integer, Time of day</td>
</tr>
<tr>
<td>ReasonCode</td>
<td>Integer, Reason code</td>
</tr>
<tr>
<td>MessageID</td>
<td>Integer, Message ID</td>
</tr>
</tbody>
</table>

OnSetAgentModeEvent

The OnSetAgentModeEvent message is generated when the agent mode is set. This message has no fields.

OnCurrentCallChanged

The OnCurrentCallChanged message is generated when the current call has changed to another call. This message has no fields.

OnCurrentAgentReset

The OnCurrentAgentReset message is generated when the current agent is removed from the session. This message has no fields.
Agent Object

The Agent object provides an abstraction of Agent behavior. It incorporates all information necessary to login to various devices and servers, such as an ACD, and email and collaboration servers. The object stores specific information, which can include agent id, password, instrument, extension, email address and skill group(s). The Login/Logout method is used to login or out of specific servers.

If the agent is logged into an ACD, the agent object will receive agent state events. It will also receive other agent information such as Agent Statistics and Skill Group information. Skill Group information is encapsulated in a separate SkillGroup object, discussed in Chapter 1. An agent can be a member of more than one skill group and methods exist to access each skill group. The Skill Group object receives skill group statistics.

Methods

Table 5-1 lists the supported Agent object methods.

Table 5-1  Agent Object Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentTeamList</td>
<td>Retrieves the current agent team list.</td>
</tr>
<tr>
<td>BadCallLine</td>
<td>Informs the CTI OS Server of a bad line.</td>
</tr>
<tr>
<td>DisableAgentStatistics</td>
<td>Disables agent statistic messages.</td>
</tr>
<tr>
<td>DisableSkillGroupStatistics</td>
<td>Disables skill group statistic messages.</td>
</tr>
</tbody>
</table>
Table 5-1  Agent Object Methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emergency</td>
<td>Lets an agent makes an emergency call to the supervisor.</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>GetElement</td>
<td>Retrieves a property from the Agent object based on the property’s index value.</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>GetPropertyAttribute</td>
<td>Retrieves attribute information for a specified agent property.</td>
</tr>
<tr>
<td>GetSkillGroups</td>
<td>Returns an array of SkillGroups objects</td>
</tr>
<tr>
<td>GetValue (also contains GetValueInt, GetValueString)</td>
<td>Retrieves a property from the Agent object based on the property’s name key.</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>MonitorAgentTeam</td>
<td>Enables monitoring of a specified agent team.</td>
</tr>
<tr>
<td>MonitorAgentTeamAll</td>
<td>Enables monitoring of all agent teams.</td>
</tr>
<tr>
<td>OnEvent</td>
<td>Specifies an action to take when an agent object receives agent state, agent statistic and associated skill group events.</td>
</tr>
<tr>
<td>SendChatMessage, SendUserMessage</td>
<td>Send asynchronous messages between CTI clients</td>
</tr>
<tr>
<td>SetAgentState</td>
<td>Requests a new agent state.</td>
</tr>
<tr>
<td>SetMonitoredAgent</td>
<td>Sets an agent object to be monitored.</td>
</tr>
<tr>
<td>SetMonitoredCall</td>
<td>Sets a call object to be monitored.</td>
</tr>
</tbody>
</table>
AgentTeamList

The AgentTeamList method retrieves the current agent team list.

Syntax

C++

```c++
int RequestAgentTeamList ( );
```

COM (standard COM API)

```plaintext
HRESULT AgentTeamList ( );
```

Parameters

None.

Return Values

C++

If the method was able to get AgentTeamList, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

---

**Table 5-1 Agent Object Methods (continued)**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetValue</td>
<td>Sets a value for a property name key.</td>
</tr>
<tr>
<td>SuperviseCall</td>
<td>Enables monitoring a call of an agent on your team.</td>
</tr>
<tr>
<td>SupervisorAssist</td>
<td>Lets an agent makes a request for assistance call to the supervisor.</td>
</tr>
</tbody>
</table>
If the method was able to get AgentTeamList, it returns S_OK. Otherwise, it returns E_FAIL.

**Examples**

**C++**

```cpp
CAgent * pAgent = NULL;
Int nRet = 0;

// Get a valid agent pointer
if (pAgent)
{
    nRet = pAgent->RequestAgentTeamList();
    // Check if the RequestAgentTeamList method failed
}
```

**COM C++**

```cpp
HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;

// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    hr = pAgent->AgentTeamList();
    if (FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pAgent = NULL;
}

return hr;
```
BadCallLine

The BadCallLine method lets the CTI OS server know that the quality of a line is bad.

Syntax

C++

int BadCallLineRequest ( );

COM

HRESULT BadCallLine ( );

Parameters

None.

Return Values

C++

If the method was able to send BadCallLine request, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to send BadCallLine request, it returns S_OK. Otherwise, it returns E_FAIL.
Examples

C++

CAgent * pAgent = NULL;
Int nRet = 0;

// Get a valid agent pointer
if (pAgent)
{
    nRet = pAgent->BadCallLineRequest () ;
    // Check if the BadCallLineRequest method failed
}

COM C++

HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;

// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    hr = pAgent->BadCallLine () ;
    if (FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pAgent = NULL;
}

return hr;

Emergency

The Emergency method lets an agent makes an emergency call to the supervisor.
Chapter 5      Agent Object

Syntax

C++

int EmergencyCallRequest ( );

COM

HRESULT Emergency ( );

Parameters

None.

Return Values

C++

If the method was able to send Emergency request, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to send Emergency request, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

CAgent * pAgent = NULL;
Int nRet = 0;

// Get a valid agent pointer
if (pAgent)
{
    nRet = pAgent->EmergencyCallRequest ();
    // Check if the EmergencyCallRequest method failed
}
COM C++

HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;

// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    hr = pAgent->Emergency();
    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pAgent = NULL;
}

return hr;

EnableAgentStatistics/DisableAgentStatistics

These methods enable or disable agent statistic messages. You should supply an empty Arguments array for these methods.

When statistics are enabled, the client will periodically receive Agent statistics as an OnAgentStatistics message. The frequency at which statistics messages are sent is configured on the CTI OS Server. See Table 5-3 for a description of the individual agent statistics and how to access them.

Syntax

C++

int EnableAgentStatistics (Arguments & rArguments);
int DisableAgentStatistics (Arguments & rArguments);

COM

HRESULT EnableSkillGroupStatistics ( [in] VARIANT * pVariantArgs );
HRESULT DisableSkillGroupStatistics ( [in] VARIANT * pVariantArgs );
Parameters

pVariantArgs
Pointer to variant that wraps an empty argument

Return Values

C++

If the methods EnableAgentStatistics/DisableAgentStatistics succeed, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the EnableAgentStatistics and DisableAgentStatistics methods succeed, they return S_OK. Otherwise, they return E_FAIL.

Examples

C++

CAgent * pAgent = NULL;
Int nRet = 0;

// Get a valid agent pointer
if (pAgent)
{
    // Create an empty argument
    Argument & rArgs;
    nRet = pAgent->EnableAgentStatistics(rArgs);

    // Check if the EnableAgentStatistics method failed
}
COM C++

HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;
IArguments * pArgs = NULL;
// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    // Create and empty argument
    hr = pAgent->DisableAgentStatistics (pArgs);
    if (FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pAgent = NULL;
}

return hr;

GetAgentState

The GetAgentState method returns the current agent state as an enumCTIOSAgentState.

Syntax

C++

virtual enumCTIOS_AgentState GetAgentState();

COM (standard COM API)

HRESULT GetAgentState ([out, retval] long * pAgentStateCode );

Output Parameters

pAgentStateCode

    Pointer that returns agent state
Return Values

C++

Returns the agent state.

COM

If the method succeeds, it returns S_OK and the agent state. Otherwise, it returns E_FAIL.

Examples

C++

```c++
CAgent * pAgent = NULL;
enumCTIOS_AgentState AgentState;

// Get a valid agent pointer
if (pAgent)
{
    AgentState = pAgent->GetAgentState();
    // Check if the GetAgentState method failed
}
```

COM C++

```c++
HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;
Long AgentState;
// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    hr = pAgent->GetAgentState (&AgentState);
    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pAgent = NULL;
}
return hr;
```
VB

Dim myState As ctios.enumCTIOSAgentState
MyState = agent.GetAgentState

GetElement

The GetElement method retrieves an element from an array property, see Table 5-2. For additional information see Chapter 1, “Introduction.”

GetMonitoredAgent

The GetMonitoredAgent method returns the agent object that is currently being monitored.

Syntax

C++

virtual CAgent * GetMonitoredAgent();

COM

HRESULT GetMonitoredAgent ( [out ] IAgent ** pIAgent );

Output Parameters

pIAgent

The returned agent object
Chapter 5  Agent Object

Methods

Return Values

C++

An agent object.

COM

If the method succeeds, it returns S_OK and an agent object. Otherwise, it returns E_FAIL.

Examples

C++

CAgent * pAgent = NULL;

// Get a valid agent pointer
if (pAgent)
{
    CAgent * pMonitoredAgent = pAgent->GetMonitoredAgent();

    // Check if the pMonitoredAgent is a good pointer
}

COM C++

HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;
IAgentPtr pMonitoredAgent = NULL;
// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    hr = pAgent->GetMonitoredAgent (&pMonitoredAgent);
    if (FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pAgent = NULL;
}

return hr;
GetMonitoredCall

The GetMonitoredCall method returns the call object that is being monitored.

Syntax

C++

virtual CCall * GetMonitoredCall();

COM (standard COM API)

HRESULT GetMonitoredCall ( [out ] ICall ** pICall );

Output Parameters

pIAgent
The returned call object

Return Values

C++

Returns a call object.

COM

If the method succeeds, it returns S_OK and a call object. Otherwise, it returns E_FAIL.
Examples

C++

CAgent * pAgent = NULL;

// Get a valid agent pointer
if (pAgent)
{
    CCall * pMonitoredCall = pAgent->GetMonitoredCall();

    // Check if the pMonitoredCall is a good pointer
}

COM C++

HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;
ICallPtr pMonitoredCall = NULL;
// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    hr = pAgent->GetMonitoredCall (&pMonitoredCall);
    if (FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pAgent = NULL;
}

return hr;
GetPropertyAttribute

Syntax

GetPropertyAttribute( propertyname, attribute requested )

The GetPropertyAttribute method retrieves attribute information for any of the properties listed in Table 5-2. For additional information on GetPropertyAttribute, see Chapter 1, “Introduction.”

Table 5-2 Agent Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentID</td>
<td>string</td>
<td>Can be set prior to Login or after Logout.</td>
</tr>
<tr>
<td>Instrument</td>
<td>string</td>
<td>Instrument associated by ACD to agent.</td>
</tr>
<tr>
<td>Extension</td>
<td>string</td>
<td>Extension associated by ACD to agent.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>integer</td>
<td>ID of peripheral.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>integer</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>AgentState</td>
<td>integer</td>
<td>Agent State value reported by ACD.</td>
</tr>
<tr>
<td>LastError</td>
<td>integer</td>
<td>Last error code, if any. Otherwise this value is 0.</td>
</tr>
<tr>
<td>Statistics</td>
<td>arguments array</td>
<td>An arguments array containing the statistics listed in Table 5-3.</td>
</tr>
</tbody>
</table>

Statistics can be accessed by first using GetValue on the Agent object to obtain the “Statistics” arguments array and then using GetValue on the “Statistics” arguments array to obtain the desired value:

```vbnet
' First get the statistics arguments
Dim args As Arguments
args = agent.GetValue("Statistics")

' Then get the desired statistics
Dim availTimeSession As Integer
Dim loggedOnTimeSession As Integer
availTimeSession = args.GetValue("AvailTimeSession")
bargeInCallsToday = args.GetValue("BargeInCallsToday")
```
Note: Not all the statistics values listed below are present in every system configuration. Whether or not a particular statistic value is available depends both on the protocol version of CTIServer with which CTIOS connects and on the peripheral on which the agent resides.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailTimeSession</td>
<td>Total time, in seconds, the agent was in the Available state for any skill group.</td>
</tr>
<tr>
<td>LoggedOnTimeSession</td>
<td>Total time, in seconds, the agent has been logged on.</td>
</tr>
<tr>
<td>NotReadyTimeSession</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>Total number of completed outbound ACD calls made by agent.</td>
</tr>
<tr>
<td>AgentOutCallsTalkTimeSession</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>AgentOutCallsTimeSession</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>The total number of completed outbound ACD calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeSession</td>
<td>Total number of seconds outbound ACD calls were placed on hold.</td>
</tr>
</tbody>
</table>
### Table 5-3  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandledCallsSession</td>
<td>The number of inbound ACD calls handled by the agent.</td>
</tr>
<tr>
<td>HandledCallsTalkTimeSession</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>HandledCallsAfterCall</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.</td>
</tr>
<tr>
<td>TimeSession</td>
<td></td>
</tr>
<tr>
<td>HandledCallsTimeSession</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>IncomingCallsHeldSession</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>Total number of seconds completed inbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>InternalCallsSession</td>
<td>Number of internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCallsTimeSession</td>
<td>Number of seconds spent on internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCallsRcvdSession</td>
<td>Number of internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeSession</td>
<td>Number of seconds spent on internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCallsHeldSession</td>
<td>The total number of internal calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCallsHeldTimeSession</td>
<td>Total number of seconds completed internal calls were placed on hold.</td>
</tr>
</tbody>
</table>
### Table 5-3  Agent Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Meaning</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>
</tr>
<tr>
<td>LoggedOnTimeToday</td>
<td>Total time, in seconds, the agent has been logged on.</td>
</tr>
<tr>
<td>NotReadyTimeToday</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>Total number of completed outbound ACD calls made by agent.</td>
</tr>
<tr>
<td>AgentOutCallsTalkTimeToday</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCallsTimeToday</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCallsHeldToday</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>Total number of seconds outbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>HandledCallsToday</td>
<td>The number of inbound ACD calls handled by the agent.</td>
</tr>
<tr>
<td>HandledCallsTalkTimeToday</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by the agent. Includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
**Table 5-3  Agent Statistics (continued)**

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandledCallsAfterCallTimeToday</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.</td>
</tr>
<tr>
<td>HandledCallsTimeToday</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by the agent. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>IncomingCallsHeldToday</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>Total number of seconds completed inbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>InternalCallsToday</td>
<td>Number of internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCallsTimeToday</td>
<td>Number of seconds spent on internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCallsRcvdToday</td>
<td>Number of internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeToday</td>
<td>Number of seconds spent on internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCallsHeldToday</td>
<td>The total number of internal calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCallsHeldTimeToday</td>
<td>Total number of seconds completed internal calls were placed on hold.</td>
</tr>
</tbody>
</table>
GetSkillGroups

The GetSkillGroups method returns an array of SkillGroups Objects. If one or more skill groups are specified in the method, the returned array of skill group objects will be limited to the requested skill groups. The skill group list is generated by the ACD (or other servers) when the agent performs a Login or when detected in Monitor Mode. The purpose of the method is to provide a list of objects to iterate.

Syntax

GetSkillGroups( optionalSkillGroup )

Example

' assume login performed
Dim args As ctios.Arguments
args = agent.GetSkillGroups()
for each skill in args do
  ...
Next

GetValue

The GetValue method retrieves a property from the Agent object based on the property’s name key (see Table 5-2). GetValue takes either a single key name or an array of key names as its required argument, and returns the value associated with that key.

Dim myExt As String
myExt = agent.GetValue("Extension")

You can use GetValue to retrieve agent statistics. Statistics are read-only. Any attempt to write to a statistic will fail.

' get any one statistic
Dim nAvgCallDuration As Integer
nAvgCallDuration = agent.GetValue( "AvgCallDuration" )

' iterate over all statistics
Dim agent As CTIOSCLIENTLIB.Agent
Dim allStatistics As Arguments
allStatistics = agent.GetValue( "Statistics" )
For Each stat In allStatistics
  ...
Next

Syntax

GetValue (key)

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 Instrument. Often, based on customer configuration, the minimum requirement includes an ACD password. Optional arguments include Extension or PositionID and Password.

Syntax

C++

virtual int Login(Arguments & rArguments);

COM (standard COM API)

HRESULT Login ( [in] VARIANT * pVariantArgs );

Input Parameters

rArguments
  Arguments array that contains login parameters.

pVariantArgs
  Arguments array that contains login parameters.
Return Values

C++

If the method was able to login the agent, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to login the agent, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```cpp
CAgent * pAgent = NULL;
Arguments & rArgs;
int nRet = 0

// Get a valid agent pointer
if (pAgent)
{
    // Fill out the argument with agent information
    nRet = pAgent->Login(rArgs);

    // Check if the Login method failed
}
```

COM C++

```cpp
HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;
IArguments pArgs = NULL;

// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    // Fill out the argument with agent information
    hr = pAgent->Login(pArgs);
    if FAILED(hr)
    {
        // You might want to log an error description HERE
    }`
VB

Dim argsLogin Param As New Arguments
Dim WithEvents session As ctios. Session
Dim WithEvents agent As ctios. Agent

session.SetAgent( agent )
agent.Login( "agentid=23814;instrument=23815;password=1234" )

Logout

The Logout method logs out the agent from the ACD. Optionally, if the ACD configuration supports logout reason codes (currently only the Avaya Definity ECS does), the Logout method accepts a reason code passed by the client as an integer.

Syntax

C++

virtual int Logout(Arguments & rArguments);

COM (standard COM API)

HRESULT Logout ( [in] VARIANT * pVariantArgs );
Chapter 5  Agent Object

Methods

Input Parameters

rArguments
Arguments array that contains login parameters.
pVariantArgs
Arguments array that contains login parameters.

Return Values

C++
If the method was able to Logout agent, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM
If the method was able to Logout agent, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

CAgent * pAgent = NULL;
Arguments & rArgs;
int nRet = 0

// Get a valid agent pointer
if (pAgent)
{
  // Fill out the argument with agent information
  nRet = pAgent->Logout (rArgs);

  // Check if the Logout method failed
}
COM C++

HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;
IArgumentptr pArgs = NULL;

// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    // Fill out the argument with agent information
    hr = pAgent->Logout(pArgs);
    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pAgent = NULL;
}
return hr;

VB

Dim argsLoginForm As New Arguments
Dim WithEvents session As ctios.Session
Dim WithEvents agent As ctios.Agent

argsLoginForm.AddItem( "agentid=23814;instrument=23815;password=1234"
agent.Login(argsLoginForm)
agent.Logout(argsLoginForm)

MakeCall

The MakeCall method initiates a call to a device or agent. The simplest form of the request only requires a dialed number. The request may also include all of the parameters in Table 5-4.
### Table 5-4 Make Call Optional Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallPlacementType</td>
<td>One of the values specifying how the call is to be placed identified in the Call Placement Type Table.</td>
</tr>
<tr>
<td>CallMannerType</td>
<td>One of the values specifying additional call processing options identified in the Call Manner Type Table.</td>
</tr>
<tr>
<td>AlertRings</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>One of the values specifying additional peripheral-specific call options.</td>
</tr>
<tr>
<td>FacilityType</td>
<td>One of the values indicating the type of facility to be used.</td>
</tr>
<tr>
<td>AnsweringMachine</td>
<td>One of the values specifying the action to be taken if the call is answered by an answering machine.</td>
</tr>
<tr>
<td>Priority</td>
<td>This field should be set to TRUE if the call should receive priority handling.</td>
</tr>
<tr>
<td>PostRoute</td>
<td>When this field is set to TRUE, the Post Routing capabilities of the Intelligent Call Manager are to be used to determine the new call destination.</td>
</tr>
<tr>
<td>UserToUserInfo</td>
<td>The ISDN user-to-user information.</td>
</tr>
<tr>
<td>Call and ECC Variables</td>
<td>Up to 10 call variables and any of the defined ECC variables or array variables.</td>
</tr>
<tr>
<td>CallWrapupData</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>FacilityCode</td>
<td>A trunk access code, split extension, or other data needed to access the chosen facility.</td>
</tr>
</tbody>
</table>
### Syntax

**C++**

```cpp
virtual int MakeCall(Arguments & rArguments);
```

**COM (standard COM API)**

```cpp
HRESULT MakeCall ([in] VARIANT * pVariantArgs );
```

### Input Parameters

- **rArguments**  
  Arguments array that holds parameters from Table 5-4.

- **pVariantArgs**  
  Arguments array that holds parameters from Table 5-4.

### Return Values

**C++**

If the method succeeds, it returns CIL_OK. Otherwise, it returns the error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

**COM**

If the method succeeds, it returns S_OK. Otherwise, it returns E_FAIL.
Examples

C++

CAgent * pAgent = NULL;
Arguments & rArgs;
int nRet;
// Get a valid agent pointer
if (pAgent)
{
  // Fill out the argument with valid parameters
  nRet = pAgent->MakeCall (rArgs);
  // Check if the method fails
}

COM C++

HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;
IArgumentsPtr & rArgs;
int nRet;
// First, get agent object and make sure it is valid pointer
if (pAgent)
{
  // Fill out the argument with valid parameters
  hr = pAgent->MakeCall (&rArgs);
  if (FAILED(hr))
  {
    // You might want to log an error description HERE
  }
  pAgent = NULL;
}
return hr;
MonitorAgentTeam

The MonitorAgentTeamAll method enables monitoring of the agent teams specified by the TeamID parameter.

Syntax

C++

virtual int MonitorAgentTeam( bool bStart, int TeamId );

COM (standard COM API)

HRESULT MonitorAgentTeam ( [in] VARIANT_BOOL bStart, [in] ULONG TeamId );

Input Parameters

.bStart

Specifies whether to start or stop monitoring

TeamId

Team ID for the agent teams to be monitored

Return Values

C++

If the method succeeds, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method succeeds, it returns S_OK. Otherwise, it returns E_FAIL.
Examples

C++

```cpp
CAgent * pAgent = NULL;
Arguments & rArgs;
int nRet;
// Get a valid agent pointer
if (pAgent)
{
    nRet = pAgent->MonitorAgentTeam ((0 == bStart) ? false : true, TeamId);
    // Check if the method fails
}
```

COM C++

```cpp
HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;
// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    hr = pAgent->MonitorAgentTeam ((bStart, TeamId);
    if (FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pAgent = NULL;
}
return hr;
```

MonitorAgentTeamAll

The MonitorAgentTeamAll method enables monitoring of all agent teams.

Syntax

C++

```cpp
virtual int MonitorAgentTeamAll ( bool bStart);
```
COM (standard COM API)

HRESULT MonitorAgentTeamAll ([in] VARIANT_BOOL bStart);

Input Parameters

bStart

Specifies whether to start or stop monitoring all agent teams.

Return Values

C++

If the method succeeds, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method succeeds, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

CAgent * pAgent = NULL;
Arguments & rArgs;
int nRet;
// Get a valid agent pointer
if (pAgent)
{
    nRet = pAgent->MonitorAgentTeamAll ((0 == bStart) ? false : true);
    // Check if the method fails
}

COM C++

HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;
// First, get agent object and make sure it is valid pointer
if (pAgent)
{
hr = pAgent->MonitorAgentTeamAll ((bStart);
if (FAILED(hr))
{
    // You might want to log an error description HERE
}
pAgent = NULL;
}
return hr;

**OnEvent (C++ Only)**

The OnEvent method specifies an action to take when an agent object receives agent state, agent statistic and associated skill group events. The default behavior of the object is to update its internal state based on the event, and fire events to clients via the IAgentEvents interface. Applications are free to override the OnEvent method and add their own logic, where applicable.

**Syntax**

```
virtual void OnEvent(int iEventID, Arguments & rEventParam);
```

**Input Parameters**

- **iEventID**
  
  A unique number that points to a unique event.

- **rEventParam**
  
  Parameters for an event

**Return Values**

None.

**Examples**

```c++
CAgent * pAgent = NULL;
Arguments & rArgs;
int nRet;
// Get a valid agent pointer
```
if (pAgent)
{
    // Fill out the rEventParam with valid parameters, and set the
    // event that you need to process
    pAgent -> OnEvent(iEventID, rEventParam);
}

SendUserMessage/SendChatMessage

The SendUserMessage and SendChatMessage methods send asynchronous chat-like messages between CTI clients. These methods let the user specify a distribution of one or more clients, and attach a text message. The recipient receives the message via the OnUserMessage event. The request may also include all of the parameters from Table 5-5.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution</td>
<td>Specifies one of the values of eDistributeToClient, eDistributeToSupervisor, eDistributeToTeam, or eDistributeToAll indicating to whom this message is to be sent.</td>
</tr>
<tr>
<td>ClientID (optional)</td>
<td>When the Distribution is set to DistributeToClient, this field must be included with the ClientID of the intended recipient.</td>
</tr>
<tr>
<td>Text (optional)</td>
<td>The text of the user message. Maximum message size is 255 bytes.</td>
</tr>
</tbody>
</table>

Syntax

C++

```cpp
virtual int SendUserMessage(Arguments & rArguments);
virtual int SendChatMessage(Arguments & rArguments);
```
Chapter 5  Agent Object

Methods

**COM (standard COM API)**

```
HRESULT SendUserMessage ( [in] UINT nDistribution, [in] VARIANT *pVariantArgs );
HRESULT SendChatMessage ( [in] VARIANT * pVariantArgs );
```

**Input Parameters**

- **rArguments**
  Array that contains parameters from Table 5-5.
- **nDistribution**
  Indicates to whom this message is to be sent.
- **pVariantArgs**
  Array that contains parameters from Table 5-5.

**Return Values**

**C++**

If the method succeeds, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

**COM**

If the method succeeds, it returns S_OK. Otherwise, it returns E_FAIL.

**Examples**

**C++**

```cpp
CAgent * pAgent = NULL;
Arguments & rArgs;
int nRet;
// Get a valid agent pointer
if ( pAgent )
{
  // Fill out the argument with valid parameters
  nRet = pAgent->SendUserMessage( rArgs );
} 
```
SetAgentState

The SetAgentState method requests a new agent state. The work states are provided by agent state model of the ACD and is vendor-specific. When a client tries to SetAgentState to an invalid state, such as no functional equivalent is supported on the ACD, or illegal transition, an exception will be returned explaining the error.

Login and Logout are valid states and can be set using the SetAgentState method. The Login method provides shorthand that helps to organize the required arguments.

Syntax

C++

virtual int SetAgentState(Arguments & rArguments);
Chapter 5      Agent Object

Methods

COM (standard COM API)

HRESULT SetAgentState ( [in] VARIANT * pVariantArgs );

Input Parameters

rArguments
Contains the new agent state.
pVariantArgs
Contains the new agent state.

Return Values

C++
If the method succeeds, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM
If the method succeeds, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

CAgent * pAgent = NULL;
Arguments & rArgs;
int nRet;
// Get a valid agent pointer
if (pAgent)
{
    // Fill out the argument with valid parameters
    nRet = pAgent->SetAgentState (rArgs);

    // Check if the method fails
}
COM C++

HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;
IArgumentsPtr rArg = NULL;
// First, get agent object and make sure it is valid pointer
if (pAgent)
{
// Fill out the argument with valid parameters
hr = pAgent->SetAgentState(rArg);
if (FAILED(hr))
{
    // You might want to log an error description HERE
}
}
pAgent = NULL;
return hr;

SetMonitoredAgent

The SetMonitoredAgent method sets a agent object to be monitored.

Syntax

C++

virtual int SetMonitoredAgent( CAgent * pAgent, bool bMonitor );
virtual int SetMonitoredAgent( CAgent * pAgent );

COM (standard COM API)

HRESULT SetMonitoredAgent ( [in] IAgent * pIAgent, [in] VARIANT_BOOL bMonitor );

Input Parameters

pAgent / pIAgent

    Pointer to the agent object to be monitored.

bMonitor / bMonitor

    Whether to enable or disable agent monitoring.
Return Values

C++

If the method succeeds, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method succeeds, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```cpp
CAgent * pAgent = NULL;
int nRet;
// Get a valid agent pointer
if (pAgent)
{
    nRet = pAgent->SetMonitoredAgent( pAgent, vbMonitor);
    // Check if the method fails
}
```

COM C++

```cpp
HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;
// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    hr = pAgent->SetMonitoredAgent( pAgent, vbMonitor);
    if (FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pAgent = NULL;
}
return hr;
```
SetMonitoredCall

The SetMonitoredCall method sets a call object to be monitored.

Syntax

C++

virtual int SetMonitoredCall( CCall * pCall );

COM (standard COM API)

HRESULT SetMonitoredCall ( [in] ICall * pICall );

Input Parameters

pCall

Pointer to the call object that needs to be monitored.

pICall

Pointer to the com call object that needs to be monitored.

Return Values

C++

If the method succeeds, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method succeeds, it returns S_OK. Otherwise, it returns E_FAIL.
Examples

C++

```c++
CAgent * pAgent = NULL;
CCall * pCall = NULL;
int nRet;
// Get a valid agent pointer
if (pAgent)
{
    nRet = pAgent->SetMonitoredCall (pCall);
    // Check if the method fails
}
```

COM C++

```c++
HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;
ICallPtr pCall = NULL;
// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    hr = pAgent->SetMonitoredCall (pCall);
    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pAgent = NULL;
}
return hr;
```

SetValue

The SetValue method sets a value for a key. A True return value means the entry was located. Keys are not case sensitive. In C++ version, there is SetValue method that takes a string which contains more then one key and one value.

Syntax

```
SetValue (key)
```
Example

```cpp
string kvstr = "Key1 = Val1; Key2 = Val2; Key3 = Val3.1, Val3.2, Val3.3";
Arguments1.SetValue(kvstr); // Leading and trailing spaces are always removed from the key.
```

SuperviseCall

The SuperviseCall method lets you start monitoring a call of an agent on your team. You must be in Supervisor mode to use this method.

Syntax

**C++**

```cpp
virtual int SuperviseCallRequest( int nSuperviseMode, CAgent * pAgent, CCall * pCall = 0 );
```

**COM (standard COM API)**

```cpp
HRESULT SuperviseCall ( [in] UINT nSuperviseMode, [in] IAgent * pIAgent, [in] ICall * pICall );
```

Input Parameters

- **nSuperviseMode**
  Supervisor or agent mode.

- **pAgent / pIAgent**
  Pointer to COM agent object.

- **pCall / pICall**
  Pointer to call object.
Chapter 5      Agent Object

Return Values

C++

If the method succeeds, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method succeeds, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

    CAgent * pAgent = NULL;
    CAgent * pAnotherAgent = NULL;
    CCall * pCall = NULL;
    int nRet;
    // Get a valid agent pointer
    if (pAgent)
    {
        nRet = pAgent->SuperviseCallRequest(nSuperviseMode, pAnotherAgent, pCall);
        // Check if the method fails
    }

COM C++

    HRESULT hr = S_OK;
    IAgentPtr pAgent = NULL;
    IAgentPtr pAnotherAgent = NULL;
    ICallPtr pCall = NULL;
    // First, get agent object and make sure it is valid pointer
    if (pAgent)
    {
        hr = pAgent->SuperviseCall(nSuperviseMode, pAnotherAgent, pCall);
        if (FAILED(hr))
        {
            // You might want to log an error description HERE
        }
        pAgent = NULL;
    }
    return hr;
SupervisorAssist

The Supervisor Assist method allows the agent to make a call back to the supervisor requesting for assistance.

Syntax

C++

    virtual int SupervisorAssistRequest();

COM (standard COM API)

    HRESULT SupervisorAssist();

Parameters

None.

Return Values

C++

    If the method succeeds, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

    If the method succeeds, it returns S_OK. Otherwise, it returns E_FAIL.
IAgentEvents Interface

The Agent object fires events on the IAgentEvents interface. The following events are published to subscribers of the IAgentEvents interface.

Examples

C++

```c++
CAgent * pAgent = NULL;
int nRet;
// Get a valid agent pointer
if (pAgent)
{
    nRet = pAgent->SupervisorAssistRequest ();
    // Check if the method fails
}
```  

COM C++

```c++
HRESULT hr = S_OK;
IAgentPtr pAgent = NULL;

// First, get agent object and make sure it is valid pointer
if (pAgent)
{
    hr = pAgent->SupervisorAssist();
    if (FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pAgent = NULL;
}
return hr;
```
OnAgentStateChange

The OnAgentStateChange event is generated when the agent state at the ACD changes. This may be a response to a Login, Logout, or SetAgentState request. The message is generated when CTI OS receives an AGENT_STATE_EVENT message.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the change occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>SkillGroupState</td>
<td>Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.</td>
</tr>
<tr>
<td>StateDuration</td>
<td>The number of seconds since the agent entered this state (typically 0).</td>
</tr>
<tr>
<td>SkillGroupNumber</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>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>SkillGroupPriority</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>AgentState</td>
<td>One of the values representing the current overall state of the associated agent.</td>
</tr>
<tr>
<td>EventReasonCode</td>
<td>A peripheral-specific code indicating the reason for the state change.</td>
</tr>
<tr>
<td>CTIClientSignature (optional)</td>
<td>The Client Signature of the CTI Client that is associated with this agent.</td>
</tr>
</tbody>
</table>
**OnAgentStatistics**

The `OnAgentStatistics` event is generated when an agent statistic is reported.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentID (optional)</td>
<td>The agent’s ACD login ID.</td>
</tr>
<tr>
<td>AgentExtension (optional)</td>
<td>The agent’s ACD teleset extension.</td>
</tr>
<tr>
<td>AgentInstrument (optional)</td>
<td>The agent’s ACD instrument number.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>SkillGroupState</td>
<td>Values representing the current state of the associated agent with respect</td>
</tr>
<tr>
<td></td>
<td>to the indicated Agent Skill Group.</td>
</tr>
<tr>
<td>ICMCentralControllerTime</td>
<td>The current ICM Central Controller date and time.</td>
</tr>
<tr>
<td>Distribution</td>
<td>One of value of <code>eDistributeToClient</code>, <code>eDistributeToSupervisor</code>, <code>eDistributeToTeam</code>, or <code>eDistributeToAll</code>.</td>
</tr>
<tr>
<td>ClientID</td>
<td>The ClientID of the message sender.</td>
</tr>
<tr>
<td>Text</td>
<td>The text message provided by the sender.</td>
</tr>
</tbody>
</table>

**OnUserMessage**

The `OnUserMessage` event is generated when a user message (asynchronous text from another user) is received.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMCentralControllerTime</td>
<td>The current ICM Central Controller date and time.</td>
</tr>
<tr>
<td>Distribution</td>
<td>One of value of <code>eDistributeToClient</code>, <code>eDistributeToSupervisor</code>, <code>eDistributeToTeam</code>, or <code>eDistributeToAll</code>.</td>
</tr>
<tr>
<td>ClientID</td>
<td>The ClientID of the message sender.</td>
</tr>
<tr>
<td>Text</td>
<td>The text message provided by the sender.</td>
</tr>
</tbody>
</table>
Call Object

The Call object controls calls, receives call events, and manages call context. The Call object is created when the Session object receives an OnCallBegin event. Once a call has been created, it will receive call events.

Client applications do not have to distinguish between CallVariables and ECC variables. Both types of variables are accessible as any other property through the GetValue method. Applications can access each variable by name or iterate through call variables, ECC variables, or ECC arrays.

Client applications do not have to track calls by ConnectionCallID, ConnectionDeviceID, or ConnectionDeviceIDType. Although these properties are accessible, clients should track calls by call identifier or call reference. The call identifier is a unique string used to identify the call. The call reference is a pointer to a call object. The reference is only valid in the current execution context and while the call is active.

Active Call

The session object maintains a reference to the current active call object. The active call is the call the agent has currently selected. The ActiveCall property is set in the session object by the default implementation of Call object. The active call is maintained to facilitate some types of CTI applications.

The rules for determining the current active object are as follows:

- The active call is set when the call is answered;
- The active call is set when another call is retrieved from hold;
When the active call clears, the previous call in the stack is selected as the active call.

**Methods**

Table 6-1 lists the available 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 then either retrieves a previously held call or answers another 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.</td>
</tr>
<tr>
<td>ClearConnection</td>
<td>Clears a call connection.</td>
</tr>
<tr>
<td>Conference</td>
<td>Conferences a third party to a call.</td>
</tr>
<tr>
<td>Deflect</td>
<td>Dialed number</td>
</tr>
<tr>
<td>GetElement</td>
<td>Retrieves a property from the Agent object based on the property’s index value.</td>
</tr>
<tr>
<td>GetPropertyAttribute</td>
<td>Retrieves attribute information for a specified call property.</td>
</tr>
<tr>
<td>GetValue (also GetValueInt, GetValueString)</td>
<td>Retrieves a property from the Agent object based on the property’s name key.</td>
</tr>
<tr>
<td>Hold</td>
<td>Places an active call on hold.</td>
</tr>
<tr>
<td>MakeConsultCall</td>
<td>Places an active call on hold and makes a new call.</td>
</tr>
<tr>
<td>OnEvent</td>
<td>Specifies an action to take when a call variable or call state changes.</td>
</tr>
<tr>
<td>Reconnect</td>
<td>Clears the current call and then retrieves a specified call.</td>
</tr>
<tr>
<td>Record</td>
<td>Lets the user record a call.</td>
</tr>
<tr>
<td>Retrieve</td>
<td>Retrieves a held call.</td>
</tr>
</tbody>
</table>
Alternate

The Alternate method combines the action of placing the current call on hold and then either retrieving a previously held call or answering an alerting call at the same device. The reference to the call to be retrieved or answered is passed in the method. If successful, the method will result in generating an OnCallHold event on this call.

Syntax

C++

```cpp
int Alternate ( );
```

COM (standard COM API)

```cpp
HRESULT Alternate ( );
```

Parameters

None.
Return Values

**C++**

If the method was able to alternate the call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

**COM**

If the method was able to alternate the call, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

**C++**

```cpp
CCall * pCall = NULL;
Int nRet = 0;

if (pCall)
{
    nRet = pCall->Alternate();
    // Check if the alternate call method failed
}
```

**COM C++**

```cpp
HRESULT hr = S_OK;
ICallPtr pCall = NULL;

// First, get call object and make sure it is valid pointer
if (pCall)
{
    hr = pCall->Alternate();
    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pCall = NULL;
}

return hr;
```
Answer

The Answer method answers a call that is in the alerting or ringing state. No arguments are required. The state of the call is accessible using GetValue and retrieving either the “CurrentState” or “IsAlerting” properties. Once the call status has changed to established, this method will return an error. An OnCallRequestFailed event may be generated if the request fails.

Syntax

C++

int Answer ( );

COM (standard COM API)

HRESULT Answer ( );

Parameters

None.

Return Values

C++

If the method was able to answer the call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to answer the call, it returns S_OK. Otherwise, it returns E_FAIL.
Examples

C++

    CCall * pCall = NULL;
    Int nRet = 0;
    
    if (pCall)
    {
        nRet = pCall->Answer();
        // Check if the Answer call method failed
    }
    // Release the call after you are done with it
    pCall->Release();

COM C++

    HRESULT hr = S_OK;
    ICall * pCall = NULL;
    
    // First, get call object and make sure it is valid pointer
    if (pCall)
    {
        hr = pCall->Answer();
        if (FAILED(hr))
        {
            // You might want to log an error description HERE
        }
        // Release the call after you are done with it
        pCall->Release();
    }
    return hr;

Clear

The Clear method clears a call. No arguments are required. The call must be in Established (talking) or Active state, otherwise the method returns an exception. An OnCallRequestFailed event may be generated if the request fails.
Chapter 6      Call Object

Methods

Syntax

C++

    int Clear ( );

COM (standard COM API)

    HRESULT Clear ( );

Parameters

None.

Return Values

C++

    If the method was able to Clear the call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

    If the method was able to Clear the call, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```c
CCall * pCall = NULL;
Int nRet = 0;

if (pCall)
{
    nRet = pCall->Clear();
    // Check if the Clear call method failed
}
// Release the call after you are done with it
pCall->Release();
```
COM C++

HRESULT hr = S_OK;
ICall * pCall = NULL;

// First, get call object and make sure it is valid pointer
if ( pCall )
{
    hr = pCall->Clear();
    if (FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    // Release the call after you are done with it
    pCall->Release();
}

return hr;

ClearConnection

The ClearConnection method clears a call connection. No arguments are required. The call must be in Established (talking) or Active state, otherwise the method returns an exception. An OnCallRequestFailed event may be generated if the request fails.

Syntax

C++

int ClearConnection ( );

COM (standard COM API)

HRESULT ClearConnection ( );

Parameters

None.
Return Values

C++

If the method was able to clear the call connection, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to clear the call connection it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```cpp
CCall * pCall = NULL;
Int nRet = 0;

if (pCall)
{
    nRet = pCall-> ClearConnection ();
    // Check if the ClearConnection call method failed
}

// Release the call after you are done with it
pCall->Release();
```

COM C++

```cpp
HRESULT hr =S_OK;
ICall * pCall = NULL;

// First, get call object and make sure it is valid pointer
if ( pCall )
{
    hr = pCall-> ClearConnection ();
    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }
```
// Release the call after you are done with it
pCall->Release();
}

return hr;

Conference

The Conference method conferences a third party to a call. If the optional call reference is not provided, CTI OS will automatically locate the consultative call initiated on the current agent device and join the call to the conference.

If the optional call reference is provided, the current call is joined with the referenced call. It does not matter whether either call is a consultative call, or which of the two calls is the consultative call. A consultative call can be dialed using MakeCall or MakeConsultCall.

To support Conference/Dial/Conference, use the following sequence:

- Place the current call on hold;
- Initiate a consultative call using MakeConsultCall l("conference"), or optionally MakeCall;
- Join the two calls with Conference.

Syntax

C++

int Conference(CCall * pCall = NULL);

COM (standard COM API)

HRESULT Conference ( [in] VARIANT * pVariantArgs );
Input Parameters

PCall
   Pointer to a Call object. The default value is NULL.

pVariantArgs
   Pointer to a variant to holds a pointer to a call object.

Return Values

C++

If the method was able to conference a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to conference a call, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```cpp
CCall * pCall = NULL;
CCall * pAnotherCall = NULL;
Int nRet = 0;

if (pCall)
{
   //
   // You need a second call to make the conference, it also could be null
   nRet = pCall->Conference (pAnotherCall);
   // Check if the Conference call method failed
}
// Release the call after you are done with it
pCall->Release();
```
Deflect

The Deflect method allows a client to take an alerting call from a known device and move it to another device. Note that since there is no formal connection between a call and an alerting device, the ConnectionDeviceID of the calling connection is used here, as given in the OnCallDelivered event.

The default argument is the instrument id where the alerting call should be deflected.

Syntax

C++

```cpp
int Deflect(string & strDialeledNumber);
```

COM (standard COM API)

```csharp
HRESULT Deflect ( [in] VARIANT * pVariantArgs );
```
Input Parameters

strDialedNumber
   String that contains a dialed number.

pVariantArgs
   Pointer to a variant that wraps a dialed number.

Return Values

C++

If the method was able to Deflect a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode or E_CTIOS_INVALID_ARGUMENT if the string is empty.

COM

If the method was able to Deflect a call, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```cpp
CCall * pCall = NULL;
CCall * pAnotherCall = NULL;
Int nRet = 0;

if (pCall)
{
   // Get a valid dialed number from somewhere
   nRet = pCall->Deflect( strDialedNumber );
   // Check if the Deflect call method failed
}

// Release the call after you are done with it
pCall->Release();
```
**COM C++**

```c++
HRESULT hr = S_OK;
ICall * pCall = NULL;
VARIANT * vDialedNumber = NULL;
// First, get call object and make sure it is valid pointer
if (pCall)
{
  // Wrap a valid dialed number into variant
  hr = pCall->Deflect(vDialedNumber);
  if(FAILED(hr))
    {
      // You might want to log an error description HERE
    }
  // Release the call after you are done with it
  pCall->Release();
}
return hr;
```

**GetElement**

The GetElement method is identical to the GetValue method, 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 the Arguments array is never guaranteed. This method is useful for sequentially iterating over all items in Arguments. The index is one (1) based. The index parameter should never be less than one or equal to the number of elements in Arguments.

**Syntax**

```c++
GetElement (index )
```

**GetPropertyAttribute**

The GetPropertyAttribute method retrieves attribute information for any of the properties listed in Table 6-2. You can access properties with theGetValue Method. For additional information on GetPropertyAttribute, see Chapter 1, “Introduction.”
### Table 6-2 Call Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>string</td>
<td>Unique reference generated for a call at a client.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td></td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>string, maximum length 64</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td></td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>Call and ECC Variables</td>
<td></td>
<td>Array of call and ecc variables.</td>
</tr>
<tr>
<td>CallType</td>
<td></td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>ANI</td>
<td>string, maximum length 64</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>UserToUserInfo</td>
<td>unspecified, up to 131 bytes.</td>
<td>The ISDN user-to-user information element.</td>
</tr>
<tr>
<td>DNIS</td>
<td>string, maximum length 32</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>string, maximum length 40</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits</td>
<td>string, maximum length 40</td>
<td>The digits entered by the caller in response to IVR prompting.</td>
</tr>
<tr>
<td>NumCTIClients</td>
<td>integer</td>
<td>The number of CTI Clients previously associated with this call. This value also indicates the number of CTI Client signatures and timestamps that are present in the floating part of the message.</td>
</tr>
</tbody>
</table>
### GetValue

The `GetValue` method retrieves a property from the Agent object based on the property’s name key. The list of properties appears in Table 6-2. `GetValue` takes either a single key name or an array of key names as its required argument, and returns the value associated with that key.

You can use the `GetValue` method to retrieve agent statistics. Statistics are read-only. Any attempt to write to a statistics will fail.

**Syntax**

```java
GetValue( key )
```

### Table 6-2 Call Properties (continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RouterCallKeyDay</td>
<td>integer</td>
<td>Together with the <code>RouterCallKeyCallID</code> 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>integer</td>
<td>The call key created by the ICM. The ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>CallWrapupData</td>
<td>string, maximum length 40</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>integer</td>
<td>The status of a call at a client.</td>
</tr>
</tbody>
</table>
Example

Dim myExt As String
MyExt = agent.GetValue( eExtension )

' get any one statistic
Dim nAvgCallDuration As Integer
nAvgCallDuration = agent.GetValue( eAvgCallDuration )

' iterate over all statistics
Dim agent As ctios.Agent
Dim allStatistics As ctios.Arguments
allStatistics = agent.GetValue( eStatistics )
For Each stat In allStatistics
  ...  
  Next

Hold

The Hold method places an active call on hold. No arguments are required. This method only works when the call is in Established (talking) or Active state. All other times, this method will return an exception. An OnCallRequestFailed event may be generated if the request fails.

Syntax

C++

int Hold();

COM (standard COM API)

HRESULT Hold();

Parameters

None.
Return Values

C++

If the method was able to hold a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to hold a call, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```c++
CCall * pCall = NULL;
Int nRet = 0;

if (pCall)
{
    nRet = pCall -> Hold();
    // Check if the Hold call method failed
}

// Release the call after you are done with it
pCall->Release();
```

COM C++

```c++
HRESULT hr = S_OK;
ICall * pCall = NULL;

// First, get call object and make sure it is valid pointer
if (pCall)
{
    hr = pCall->Hold();
    if (FAILED(hr))
    { // You might want to log an error description HERE
    }
    // Release the call after you are done with it
    pCall->Release();
```
The MakeConsultCall method initiates the combined action of placing an active call on hold and then making a new call. By default, the call context data of the active call is used to initialize the context data of the 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. The request may also include the parameters in Table 6-3.

**Table 6-3 MakeConsultCall Optional Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallPlacementType</td>
<td>A value specifying how the call is to be placed identified in the Call Placement Type Table.</td>
</tr>
<tr>
<td>CallMannerType</td>
<td>A value specifying additional call processing options identified in the Call Manner Type Table.</td>
</tr>
<tr>
<td>AlertRings</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>A value specifying additional peripheral-specific call options.</td>
</tr>
<tr>
<td>FacilityType</td>
<td>A value indicating the type of facility to be used.</td>
</tr>
<tr>
<td>AnsweringMachine</td>
<td>A value specifying the action to be taken if the call is answered by an answering machine.</td>
</tr>
<tr>
<td>Priority</td>
<td>This field should be set to TRUE if the call should receive priority handling.</td>
</tr>
</tbody>
</table>
Table 6-3 MakeConsultCall Optional Parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostRoute</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</td>
<td>The ISDN user-to-user information.</td>
</tr>
<tr>
<td>Call and ECC Variables</td>
<td>Up to 10 call variables and any of the defined ECC variables or array variables.</td>
</tr>
<tr>
<td>CallWrapupData</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>FacilityCode</td>
<td>A trunk access code, split extension, or other data needed to access the chosen facility.</td>
</tr>
<tr>
<td>AuthorizationCode</td>
<td>An authorization code needed to access the resources required to initiate the call.</td>
</tr>
<tr>
<td>AccountCode</td>
<td>A cost-accounting or client number used by the peripheral for charge-back purposes.</td>
</tr>
</tbody>
</table>

Syntax

C++

```cpp
int MakeConsultCall(Arguments & rArguments);
```

COM (standard COM API)

```csharp
HRESULT MakeConsultCall ( [in] VARIANT * pVariantArgs );
```

Input Parameters

rArguments

Arguments array that holds optional parameters from Table 6-3.

pVariantArgs

Pointer of variant that wraps an Arguments array that holds parameters from Table 6-3.
Return Values

C++

If the method was able to MakeConsultCall a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to MakeConsultCall a call, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```cpp
CCall * pCall = NULL;
Arguments * pArguments = NULL;

int nRet = 0;

if (pCall)
{
    // Fill out the arguments with valid parameters
    nRet = pCall -> MakeConsultCall(*pArguments);
    // Check if the MakeConsultCall call method failed
}

// Release the call after you are done with it
pCall->Release();
```

COM C++

```cpp
HRESULT hr = S_OK;
ICall * pCall = NULL;
VARIANT * vArguments = NULL;
// First, get call object and make sure it is valid pointer
if ( pCall )
{
    // Fill out the arguments with valid parameters and then use variant to wrap it
    hr = pCall->MakeConsultCall(vArguments);
```
if(FAILED(hr))
{
    // You might want to log an error description HERE
}
// Release the call after you are done with it
pCall->Release();
}
return hr;

OnEvent (C++ Only)

Call events are received any time the call variables or call state changes. Every call receives an OnCallBegin just after the call object is created and receives an OnCallEnd just before the call object is destroyed. The call itself may not have terminated upon receipt of an OnCallEnd, just an agent's association with the call.

The standard behavior for the call object OnEvent method is to update is internal state and call context data, and then to fire the event to subscribers of the ICallEvents interface.

Applications are free to override the OnEvent method and add their own logic, where applicable. For additional information on the event model, see Chapter 1, “Introduction.”

Syntax

virtual void OnEvent(int iEventID, Arguments & rEventParam);

Input Parameters

iEventID

A unique number that points to a unique event.

rEventParam

Arguments array that contains parameters for an event.
Return Values

None.

Examples

C++

CCall * pCall = NULL;
Arguments & rEventParam;

if (pCall)
{
    // Fill out the rEventParam with valid parameters , and set the
eventID with the event
    //that you need to process
    pCall -> OnEvent(iEventID, rEventParam);
}

// Release the call after you are done with it
pCall->Release();

Reconnect

The Reconnect method combines the action of clearing the current call and then
retrieving the referenced call. If successful, the method will result in generating
OnCallCleared followed by OnCallEnd events on this call.

Syntax

C++

    int Reconnect ( );

COM (standard COM API)

    HRESULT Reconnect ( );
Parameters

None.

Return Values

C++

If the method was able to Reconnect a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to Reconnect a call, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```cpp
CCall * pCall = NULL;
Int nRet = 0;
if (pCall)
{
    nRet = pCall->Reconnect();
    // Check if the Reconnect call method failed
}
// Release the call after you are done with it
pCall->Release();
```

COM C++

```cpp
HRESULT hr = S_OK;
ICall * pCall = NULL;
// First, get call object and make sure it is valid pointer
if (pCall)
{
    hr = pCall->Reconnect();
```
if(FAILED(hr))
{
    // You might want to log an error description HERE
}
// Release the call after you are done with it
pCall->Release();
}
return hr;

Record

The Record method lets the user record a call. Arguments such as call recording media features should take default values if not specified by the client.

Syntax

C++

int Record( bool bEnable, Arguments& rArguments );

COM (standard COM API)

HRESULT Record ( [in] VARIANT_BOOL bEnable, [in] VARIANT *pVariantArgs );

Input Parameters

bEnable
   Boolean value, true for start recording and false for stop recording.

rArguments
   Information that is sent to the server. This parameter may be empty.

pVariantArgs
   Pointer to a variant that wraps an argument array.
Return Values

C++

If the method was able to Record a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to Record a call, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```cpp
CCall * pCall = NULL;
Arguments & rArgs;
Int nRet = 0;

if (pCall)
{
    // Start recording
    nRet = pCall -> Record (true, rArgs);
    // Check if the Record call method failed
}

// Release the call after you are done with it
pCall->Release();
```

COM C++

```cpp
HRESULT hr =S_OK;
ICall * pCall = NULL;
// First, get call object and make sure it is valid pointer
if ( pCall )
{
    // Stop recording
    bEnable = VARIANT_FALSE;

    // Send empty Arguments
    VARIANT vParam;
```
hr = pCall->Record(bEnable, &vParam);
VariantClear(&vParam);
if(FAILED(hr))
{
  // you might need to log an error HERE
}

// Release the call after you are done with it
pCall->Release();
}

return hr;

Retrieve

The Retrieve method takes a held call and retrieves it from hold. No arguments are required. This method only works while the call is in a Held state. All other times, this method will return an exception. An OnCallRequestFailed event may be generated if the request fails.

Syntax

C++

int Retrieve();

COM (standard COM API)

HRESULT Retrieve();

Parameters

None.
Return Values

C++

If the method was able to Retrieve a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to Retrieve a call, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```cpp
CCall * pCall = NULL;

Int nRet = 0;
if (pCall)
{
    nRet = pCall -> Retrieve ();
    // Check if the Retrieve call method failed
}

// Release the call after you are done with it
pCall->Release();
```

COM C++

```cpp
HRESULT hr =S_OK;
ICall * pCall = NULL;
// First, get call object and make sure it is valid pointer
if ( pCall )
{
    hr = pCall-> Retrieve ();
    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    // Release the call after you are done with it
```
pCall->Release();
}

return hr;

SendDTMFSignal

The SendDTMFSignal method requests the ACD transmit to send a sequence of DTMF tones.

Syntax

C++

int SendDTMFSignal(Arguments & rArguments);

COM (standard COM API)

HRESULT SendDTMFSignal ( [in] VARIANT * pVariantArgs ) ;

Input Parameters

rArguments

Arguments array that can contain the parameters listed in Table 6-4.

pVariantArgs

Pointer to variant that wraps an argument array that can contain the parameters listed in Table 6-4.
Return Values

C++

If the method was able to SendDTMFSignal a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to SendDTMFSignal a call, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```cpp
CCall * pCall = NULL;
Arguments & rArguments;
Int nRet = 0;

if (pCall)
{
    // fill the argument with the parameters that you are interested in

    nRet = pCall -> SendDTMFSignal (rArguments);
    // Check if the SendDTMFSignal call method failed
}
```
// Release the call after you are done with it
pCall->Release();

COM C++

HRESULT hr = S_OK;
ICall * pCall = NULL;
VARIANT * pVariantArgs;
// First, get call object and make sure it is valid pointer
if ( pCall )
{
    // fill the argument with the parameters that you are interested in
    hr = pCall->SendDTMFSignal (pVariantArgs);
    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    // Release the call after you are done with it
    pCall->Release();
}
return hr;

SetCallData

The SetCallData method sets Call variables, ECC variables, and other variables such as ANI, and/or user-to-user information. The method requires an array of key/value pairs describing the data to be updated.

Syntax

C++

int SetCallData(Arguments & rArguments);

COM (standard COM API)

HRESULT SetCallData ( [in] VARIANT * pVariantArgs );
Input Parameters

rArguments
Arguments array that contains call data parameters.

pVariantArgs
Pointer to a variant that wraps an argument array that contains call data parameters.

Return Values

C++
If the method was able to SetCallData a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM
If the method was able to SetCallData a call, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

```cpp
CCall * pCall = NULL;
Arguments & rArguments;
Int nRet = 0;

if (pCall)
{
    // fill the argument with the parameters that you are interested in

    nRet = pCall -> SetCallData (rArguments);
    // Check if the SetCallData call method failed
}

// Release the call after you are done with it
pCall->Release();
```
COM  C++

HRESULT hr = S_OK;
ICall * pCall = NULL;
VARIANT * pVariantArgs;
// First, get call object and make sure it is valid pointer
if ( pCall )
{
    // fill the argument with the parameters that you are interested in
    hr = pCall->SetCallData (pVariantArgs);

    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }

    // Release the call after you are done with it
    pCall->Release();
}
return hr;

VB

' update call variables using one call to SetCallData

call.SetCallData(
    "CallVariable1='1200';ecc.acct_no='45';ecc.array_var[9]='Y'"
)'
' update call variables using Arguments array and SetCallData

Dim WithEvents args As New ctios.Arguments

args.AddItem( "CallVariable1", "1200" )

args.AddItem( "ecc.acct_no", "45" )

args.AddItem( "array_var[9]", "Y" )

call.SetCallData( args )
**SingleStepConference**

The SingleStepConference method initiates a blind conference without the intermediate consultative call. When the called party answers the call, they will be joined with the current call. The method requires a DialedNumber to conference argument.

**Syntax**

C++

```c++
int SingleStepConference(Arguments & rArguments);
```

COM (standard COM API)

```c
HRESULT SingleStepConference ( [in] VARIANT * pVariantArgs );
```

**Input Parameters**

`rArguments`

Arguments array that contains a consult call.

`pVariantArgs`

Pointer to a variant that contains a consult call

**Return Values**

C++

If the method was able to SingleStepConference a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to SingleStepConference a call, it returns S_OK. Otherwise, it returns E_FAIL.
Examples

C++

CCall * pCall = NULL;
Arguments & rArguments;
Int nRet = 0;

if (pCall)
{
    // fill the argument with the consult call
    nRet = pCall -> SingleStepConference (rArguments);
    // Check if the SingleStepConference call method failed
}

// Release the call after you are done with it
pCall->Release();

COM C++

HRESULT hr = S_OK;
ICall * pCall = NULL;
VARIANT * pVariantArgs;
// First, get call object and make sure it is valid pointer
if ( pCall )
{
    // fill the argument with the consult call
    hr = pCall-> SingleStepConference (pVariantArgs);

    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }

    // Release the call after you are done with it
    pCall->Release();
}
return hr;
SingleStepTransfer

The SingleStepTransfer method initiates a blind transfer without the intermediate consultative call. When the called party answers the call, the other party to this call will be joined with the dialed party. This call receives an OnCallCleared event. The method requires a DialedNumber argument.

Syntax

C++

int SingleStepTransfer (Arguments & rArguments);

COM (standard COM API)

HRESULT SingleStepTransfer ( [in] VARIANT * pVariantArgs

Input Parameters

rArguments
Arguments array that contains a consult call.
pVariantArgs
Pointer to a variant that contains a consult call.

Return Values

C++

If the method was able to SingleStepTransfer a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to SingleStepTransfer a call, it returns S_OK. Otherwise, it returns E_FAIL.
Examples

C++

CCall * pCall = NULL;
Arguments & rArguments;
Int nRet = 0;

if (pCall)
{
    // fill the argument with the consult call
    nRet = pCall -> SingleStepTransfer (rArguments);
    // Check if the SingleStepTransfer call method failed
}

// Release the call after you are done with it
pCall->Release();

COM C++

HRESULT hr = S_OK;
ICall * pCall = NULL;
VARIANT * pVariantArgs;
// First, get call object and make sure it is valid pointer
if ( pCall )
{
    // fill the argument with the consult call
    hr = pCall->SingleStepTransfer (pVariantArgs);
    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    // Release the call after you are done with it
    pCall->Release();
} 
return hr;
The Snapshot method issues a server request to update the current call information. If values are passed in the optional ArgumentList, the snapshot record will return the server's current call information for the requested arguments; otherwise all call information is returned. The server returns the requested values as an OnCallDataUpdate event with the following key/value pair "snapshot=true".

**Syntax**

**C++**

```cpp
int Snapshot();
```

**COM (standard COM API)**

```cpp
HRESULT Snapshot();
```

**Parameters**

None.

**Return Values**

**C++**

If the method was able to Snapshot a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

**COM**

If the method was able to Snapshot a call, it returns S_OK. Otherwise, it returns E_FAIL.
Examples

C++

CCall * pCall = NULL;
Arguments & rArguments;
Int nRet = 0;

if (pCall)
{
    // fill the argument with the consult call
    nRet = pCall -> Snapshot ( );
        // Check if the Snapshot call method failed
}

// Release the call after you are done with it
pCall->Release();

COM C++

HRESULT hr = S_OK;
ICall * pCall = NULL;

// First, get call object and make sure it is valid pointer
if ( pCall )
{
    hr = pCall-> Snapshot ( );

    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }

    // Release the call after you are done with it
    pCall->Release();
}
return hr;
Transfer

The Transfer method transfers a call to a third party. If the optional call reference is not provided, CTI OS will automatically locate the consultative call initiated on the current agent device and join it to the current call.

If the optional call reference is provided, the current call is joined with the referenced call. It does not matter either call is a consultative call, or which of the two calls is the consultative call. A consultative call can be dialed using MakeCall or MakeConsultCall.

To support Transfer/ Dial/Transfer, use the following sequence:

- Place the current call on Hold;
- Initiate a consultative call using MakeConsultCall( "transfer" ), or optionally MakeCall;
- Join the two calls with Transfer.

Syntax

C++

```cpp
int Transfer (CCall * pCall = NULL);
```

COM (standard COM API)

```c
HRESULT Transfer ( [in] VARIANT * pVariantArgs );
```

Input Parameters

**PCall**

- Pointer to a Call object, the default value is NULL

**pVariantArgs**

- Pointer to a variant that holds a pointer to a call object
Return Values

C++

If the method was able to Transfer a call, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAILOVER if the session is on FailOver mode.

COM

If the method was able to Transfer a call, it returns S_OK. Otherwise, it returns E_FAIL.

Examples

C++

CCall * pCall = NULL;
CCall * pAnotherCall = NULL;
Int nRet = 0;

if (pCall)
{
    // You need a second call to make the conference, it also could be null
    nRet = pCall->Transfer (pAnotherCall);
    // Check if the Transfer call method failed
}
// Release the call after you are done with it
pCall->Release();

COM C++

HRESULT hr =S_OK;
ICall * pCall = NULL;
VARIANT * vAnotherCall = NULL;
// First, get call object and make sure it is valid pointer
if ( pCall )
{
    // You need a second call to make the Transfer, it also could be null
    // and use variant to wrap it
hr = pCall->Transfer (vAnotherCall);
if(FAILED(hr))
{
    // You might want to log an error description HERE
}
// Release the call after you are done with it
pCall->Release();
return hr;

ICallEvents Interface

The Call object fires events on the ICallEvents interface. The following events are published to subscribers of the ICallEvents interface.

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, and the ConnectionDeviceIDType and ConnectionDeviceID uniquely identify the client’s local call connection, if any, or another valid call connection. 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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>NumCTIClients</td>
<td>The number of CTI Clients previously associated with this call. This value also indicates the number of CTI Client signatures and timestamps that are present in the floating part of the message.</td>
</tr>
<tr>
<td>Call and ECC Variables</td>
<td>Initial array of call and ecc variables.</td>
</tr>
<tr>
<td>CallType</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>ANI (optional)</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>UserToUserInfo (optional)</td>
<td>The ISDN user-to-user information element. unspecified, up to 131 bytes.</td>
</tr>
<tr>
<td>DNIS (optional)</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>DialedNumber (optional)</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits (optional)</td>
<td>The digits entered by the caller in response to IVR prompting.</td>
</tr>
<tr>
<td>RouterCallKeyDay (optional)</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 (optional)</td>
<td>The call key created by the ICM. The ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>CallWrapupData (optional)</td>
<td>Call-related wrapup data.</td>
</tr>
</tbody>
</table>
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. The event does not necessarily indicate that the subject call has been terminated.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call is located.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral. type: integer.</td>
</tr>
<tr>
<td>NumCTIClients</td>
<td>The number of CTI Clients associated with this call. This value also indicates the number of CTI Client signatures and timestamps that are present in the floating part of the message.</td>
</tr>
<tr>
<td>CallType</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>The previous device identifier of the call connection.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value previously assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>NewConnectionDeviceIDType</td>
<td>Indicates the type of the connection identifier supplied in the NewConnectionDeviceID field.</td>
</tr>
<tr>
<td>NewConnectionDeviceID</td>
<td>The new identifier of call connection.</td>
</tr>
<tr>
<td>NewConnectionCallID</td>
<td>The new Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>ANI (optional)</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>UserToUserInfo (optional)</td>
<td>The ISDN user-to-user information element.</td>
</tr>
<tr>
<td>DNIS (optional)</td>
<td>The DNIS provided with the call.</td>
</tr>
</tbody>
</table>
The OnCallDelivered event may be generated when the call arrives at the agent’s teleset. Note that when a call is alerting at a teleset, there is no formal connection between the call and the alerting device, so the ConnectionDeviceID of the calling connection is reported in this message.
## Field Object

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call is located.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>When LocalConnectionState is LCS_ALERTING, this field identifies the alerting teleset line, if known. Otherwise this field is set to 0xffff.</td>
</tr>
<tr>
<td>LineType</td>
<td>Indicates the type of the teleset line given in the LineHandle field, if any.</td>
</tr>
<tr>
<td>ServiceNumber</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>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>AlertingDeviceType</td>
<td>Indicates the type of the device identifier supplied in the AlertingDeviceID field.</td>
</tr>
<tr>
<td>CallingDeviceType</td>
<td>Indicates the type of the device identifier supplied in the CallingDeviceID field.</td>
</tr>
<tr>
<td>CalledDeviceType</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID field.</td>
</tr>
</tbody>
</table>
OnCallEstablished

The OnCallEstablished event may be generated when the call is answered at the agent’s teleset. This creates a new call connection and is reported as an OnCallEstablished. The ConnectionCallID identifies the call, and the ConnectionDeviceIDType and ConnectionDeviceID uniquely identify the new call connection that was created. When more than one call with the same ConnectionCallID value exists, the connection being created by this CALL_ESTABLISHED_EVENT shall apply to the call that does not yet have a destination connection established.
### ICallEvents Interface

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call is located.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>When LocalConnectionState is LCS_ALERTING, this field identifies the alerting teleset line, if known. Otherwise this field is set to 0xffff.</td>
</tr>
<tr>
<td>LineType</td>
<td>Indicates the type of the teleset line given in the LineHandle field, if any.</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>The service that the call is attributed to, as known to the peripheral.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>The ICM ServiceID of the service that the call is attributed to.</td>
</tr>
<tr>
<td>SkillGroupNumber</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>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SkillGroupID</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>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>AnsweringDeviceType</td>
<td>Indicates the type of the device identifier supplied in the AnsweringDeviceID field.</td>
</tr>
<tr>
<td>CallingDeviceType</td>
<td>Indicates the type of the device identifier supplied in the CallingDeviceID field.</td>
</tr>
<tr>
<td>CalledDeviceType</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID field.</td>
</tr>
<tr>
<td>LastRedirectDeviceType</td>
<td>Indicates the type of the device identifier supplied in the LastRedirectDeviceID field.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>AnsweringDeviceID (optional)</td>
<td>The device identifier of the device that is alerting.</td>
</tr>
<tr>
<td>CallingDeviceID (optional)</td>
<td>The device identifier of the calling device.</td>
</tr>
<tr>
<td>CalledDeviceID (optional)</td>
<td>The device identifier of the originally called device.</td>
</tr>
<tr>
<td>LastRedirectDeviceID (optional)</td>
<td>The device identifier of the previously alerted device.</td>
</tr>
</tbody>
</table>
OnCallHeld

Placing a call on hold at the agent’s teleset may generate an OnCallHeld event.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>HoldingDeviceType</td>
<td>Indicates the type of the device identifier supplied in the HoldingDeviceID field.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>HoldingDeviceID (optional)</td>
<td>The device identifier of the device that activated the hold.</td>
</tr>
</tbody>
</table>
OnCallRetrieved

Resuming a call previously placed on hold at the agent’s teleset may generate an OnCallRetrieved event.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>RetrievingDeviceType</td>
<td>Indicates the type of the device identifier supplied in the RetrievingDeviceID field.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>RetrievingDeviceID (optional)</td>
<td>The device identifier of the device that deactivated the hold.</td>
</tr>
</tbody>
</table>
OnCallCleared

An OnCallCleared event is generated when a call is terminated, normally when the last device disconnects from a call.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
</tbody>
</table>

OnCallConnectionCleared

An OnCallConnectionCleared event is generated when a party drops from a call.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
</tbody>
</table>
OnCallOriginated

The initiation of a call from the peripheral may generate an OnCallOriginated event.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral. type: integer.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>Identifies the teleset line being used.</td>
</tr>
<tr>
<td>LineType</td>
<td>Indicates the type of the teleset line.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNumber</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>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>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>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>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>CallingDeviceType</td>
<td>Indicates the type of the device identifier supplied in the CallingDeviceID field.</td>
</tr>
<tr>
<td>CalledDeviceType</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID field.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>CallingDeviceID (optional)</td>
<td>The device identifier of the calling device.</td>
</tr>
<tr>
<td>CalledDeviceID</td>
<td>The device identifier of the originally called device.</td>
</tr>
</tbody>
</table>
## OnCallFailed

The OnCallFailed event may be generated when a call cannot be completed.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>FailingDeviceType</td>
<td>Indicates the type of the device identifier supplied in the FailingDeviceID field.</td>
</tr>
<tr>
<td>CalledDeviceType</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID field.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>FailingDeviceID (optional)</td>
<td>The device identifier of the failing device.</td>
</tr>
<tr>
<td>CalledDeviceID</td>
<td>The device identifier of the originally called device.</td>
</tr>
</tbody>
</table>
OnCallTransferConferenceInitiated

The OnCallTransferConferenceInitiated event is generated when a consultative call is delivered. The event is delivered to both calls at the originating device.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
</tbody>
</table>

OnCallTransferred

The transfer of a call to another destination may generate an OnCallTransferred event.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>PrimaryDeviceIDType</td>
<td>Indicates the type of the connection identifier supplied in the PrimaryDeviceID fields.</td>
</tr>
<tr>
<td>PrimaryCallID</td>
<td>The Call ID value assigned to the primary call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>Identifies the teleset line being used.</td>
</tr>
<tr>
<td>LineType</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>SkillGroupNumber</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>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SkillGroupID</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>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>SecondaryDeviceIDType</td>
<td>Indicates the type of the connection identifier supplied in the SecondaryDeviceID field.</td>
</tr>
<tr>
<td>SecondaryCallID</td>
<td>The Call ID value assigned to the secondary call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>TransferringDeviceType</td>
<td>Indicates the type of the device identifier supplied in the TransferringDeviceID field.</td>
</tr>
<tr>
<td>TransferredDeviceType</td>
<td>Indicates the type of the device identifier supplied in the TransferredDeviceID field.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>PrimaryDeviceID</td>
<td>The connection identifier of the primary call connection.</td>
</tr>
<tr>
<td>SecondaryDeviceID</td>
<td>The connection identifier of the secondary call connection.</td>
</tr>
<tr>
<td>TransferringDeviceID (optional)</td>
<td>The device identifier of the device that transferred the call.</td>
</tr>
</tbody>
</table>
The joining of calls into a conference call may generate an OnCallConferenced event.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>PrimaryDeviceIDType</td>
<td>Indicates the type of the connection identifier supplied in the PrimaryDeviceID field.</td>
</tr>
<tr>
<td>PrimaryCallID</td>
<td>The Call ID value assigned to the primary call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>Identifies the teleset line being used.</td>
</tr>
<tr>
<td>LineType</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>SkillGroupNumber</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>
### Field | Description
--- | ---
SkillGroupID | The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority | The priority of the skill group, or 0 when skill group priority is not applicable or not available.
SecondaryDeviceIDType | Indicates the type of the connection identifier supplied in the SecondaryDeviceID field.
SecondaryCallID | The Call ID value assigned to the secondary call by the peripheral or the ICM.
ControllerDeviceType | Indicates the type of the device identifier supplied in the ControllerDeviceID field.
AddedPartyDeviceType | Indicates the type of the device identifier supplied in the AddedPartyDeviceID field.
LocalConnectionState | The state of the local end of the connection.
EventCause | Indicates a reason or explanation for the occurrence of the event.
PrimaryDeviceID | The connection identifier of the primary call connection.
SecondaryDeviceID | The connection identifier of the secondary call connection.
ControllerDeviceID (optional) | The device identifier of the conference controller device.
The removal of a call from a previous delivery target may generate an OnCallDiverted event.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallIdentifier</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralTypeID</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>ServiceNumber</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>
### OnTranslationRoute

This event is generated when a call is routed to a peripheral monitored by the PG via a translation route. The event parameters contains the call context data that will be assigned to the call after it arrives at the peripheral.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANI (optional)</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>UserToUserInfo (optional)</td>
<td>The ISDN user-to-user information element.</td>
</tr>
<tr>
<td>DNIS</td>
<td>The DNIS of the expected call.</td>
</tr>
<tr>
<td>DialedNumber (optional)</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits (optional)</td>
<td>The digits entered by the caller in response to IVR prompting.</td>
</tr>
</tbody>
</table>
Chapter 6  Call Object

ICallEvents Interface

The OnCallEnterpriseAgent event is generated when a call is routed to an Enterprise Agent. The event parameters contain the call context data that will be assigned to the call after it arrives at the agent’s desktop. Application developers should note that, while atypical, it is possible for the call to arrive at the agent and to receive an OnCallBegin event and other call events for the call before the OnCallEnterpriseAgent event is received.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RouterCallKeyDay</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call’s records in the ICM database.</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>The call key created by the ICM. The ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>Call and ECC Variables (optional)</td>
<td>Call-related variable data.</td>
</tr>
</tbody>
</table>

### OnCallEnterpriseAgent

The OnCallEnterpriseAgent event is generated when a call is routed to an Enterprise Agent. The event parameters contain the call context data that will be assigned to the call after it arrives at the agent’s desktop. Application developers should note that, while atypical, it is possible for the call to arrive at the agent and to receive an OnCallBegin event and other call events for the call before the OnCallEnterpriseAgent event is received.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNumber</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>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>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>
### SkillGroupID
The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.

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

### AgentInstrument
The agent instrument that the call will be routed to.

### RouterCallKeyDay
Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call’s records in the ICM database.

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

### ANI (optional)
The calling line ID of the caller.

### UserToUserInfo (optional)
The ISDN user-to-user information element.

### DialedNumber (optional)
The number dialed.

### CallerEnteredDigits (optional)
The digits entered by the caller in response to IVR prompting.

### Call and ECC Variables (optional)
Call-related variable data.
Chapter 6  Call Object

ICallEvents Interface

OnCallPreEventAbort

An OnCallPreEventAbort event is generated when a call that was previously announced via an OnCallPreEvent cannot be routed as intended due to a busy or other error condition detected during call routing.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentInstrument</td>
<td>The agent instrument that the call was to have been routed to.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call’s records in the ICM database</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>The call key created by the ICM. The ICM resets this counter at midnight.</td>
</tr>
</tbody>
</table>

OnCallRequestFailed

If an Answer, Release, Hold, Retrieve, Transfer, or Conference request fails, an OnCallRequestFailed event is generated.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>ID of request</td>
</tr>
</tbody>
</table>

OnAgentPrecallEvent

The OnAgentPrecallEvent event is roughly equivalent to a TranslationRouteEvent as far as CTI OS is concerned. A non-ACD agent (e.g., enterprise agent) will get this event when he has been selected to receive the call, but before it is actually delivered. As a result, we will identify the call by its RouterCallKeyDay and RouterCallKeyCallID. We can receive events for the call while it is in queue, and we can set the call data. However, since this event happens before the BEGIN_CALL_EVENT, we will not know the PeripheralID, and thus cannot create a peripheral-specific behavior implementation object. All we can do at this point is get and set data.
Field | Description
--- | ---
RouterCallKeyDay | 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 | The call key created by the ICM. The ICM resets this counter at midnight.
AgentInstrument | The agent instrument that the call will be routed to.
NumNamedVariables | Number of Named variables
NumNamedArrays | Number of Named Array
ServiceNumber | The service that the call is attributed to, as known to the peripheral.
ServiceID | The ICM ServiceID of the service that the call is attributed to.
SkillGroupNumber | The number of the agent SkillGroup the call is attributed to, as known to the peripheral.
SkillGroupID | The ICM SkillGroupID of the agent SkillGroup the call is attributed to.
SkillGroupPriority | The priority of the skill group, or 0 when skill group priority is not applicable or not available.
ANI | The calling line ID of the caller.
UserToUserInfo | The ISDN user-to-user information element.
DialedNumber | The number dialed.
CallerEnteredDigits | The digits entered by the caller in response to IVR prompting.
CallVariable 1 through 10 | Call-related variable data.
OnAgentPreCallAbortEvent

The OnAgentPreCallAbortEvent event is received when a previously identified AgentPreCallEvent is reversed. We will identify the call by its RouterCallKeyDay and RouterCallKeyCallID. We can receive events for the call while it is in queue, and we can set the call data. However, since this event happens before the BEGIN_CALL_EVENT, you do not know the PeripheralID, and thus cannot create a peripheral-specific behavior implementation object. All you can do at this point is get and set data.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RouterCallKeyDay</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>The call key created by the ICM. The ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>The agent instrument that the call will be routed to.</td>
</tr>
</tbody>
</table>

OnCallServiceInitiatedEvent

The initiation of telecommunications service (“dial tone”) at the agent’s teleset may generate an OnCallServiceInitiatedEvent to the CTI Client.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call or device to be monitored is located.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value of the call to be monitored. This field should be set to zero when creating a monitor for a device.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td></td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of the connection identifier</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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallingDeviceID</td>
<td>The device identifier of the calling device.</td>
</tr>
<tr>
<td>CallingDeviceIDType</td>
<td>Indicates the type of the device identifier supplied in the CallingDeviceID field.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>Identifies the teleset line being used.</td>
</tr>
<tr>
<td>LineType</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>The ICM ServiceID of the service that the call is attributed to.</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>The service that the call is attributed to, as known to the peripheral.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>The ICM SkillGroupID of the agent SkillGroup the call is attributed to.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>The number of the agent SkillGroup the call is attributed to, as known to the peripheral.</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionDeviceID</td>
<td>The identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of the connection identifier supplied in the ConnectionDeviceID.</td>
</tr>
<tr>
<td>QueuedDeviceID</td>
<td>The device identifier of the queuing device.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueuedDeviceIDType</td>
<td>Indicates the type of the device identifier supplied in the QueuedDeviceID.</td>
</tr>
<tr>
<td>CallingDeviceID</td>
<td>The device identifier of the calling device.</td>
</tr>
<tr>
<td>CallingDeviceIDType</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID.</td>
</tr>
<tr>
<td>CalledDeviceID</td>
<td>The device identifier of the called device.</td>
</tr>
<tr>
<td>CalledDeviceIDType</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID.</td>
</tr>
<tr>
<td>LastRedirectedDeviceID</td>
<td>The device identifier of the redirecting device.</td>
</tr>
<tr>
<td>LastRedirectedDeviceIDType</td>
<td>Indicates the type of the device identifier supplied in the LastRedirectDeviceID.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>Identifies the teleset line being used.</td>
</tr>
<tr>
<td>LineType</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>The ICM ServiceID of the service that the call is attributed to.</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>The service that the call is attributed to, as known to the peripheral.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>The ICM SkillGroupID of the agent SkillGroup the call is attributed to.</td>
</tr>
<tr>
<td>SkillGroupNumber</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>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
</tbody>
</table>
OnCallDequeuedEvent

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumQueued</td>
<td>The number of calls in the queue for this service.</td>
</tr>
<tr>
<td>NumSkillGroups</td>
<td>The number of Skill Group queues that the call has queued to, up to a maximum of 20. This value also indicates the number of SkillGroupNumber, SkillGroupID and SkillGroupPriority floating fields present in the floating part of the message.</td>
</tr>
</tbody>
</table>

Field | Description
--- | ---
ConnectionDeviceID | The identifier of the connection between the call and the device.
ConnectionDeviceIDType | Indicates the type of the connection identifier supplied in the ConnectionDeviceID
LocalConnectionState | The state of the local end of the connection
EventCause | Indicates a reason or explanation for the occurrence of the event
LineHandle | Identifies the teleset line being used.
LineType | Indicates the type of the teleset line.
ServiceID | The ICM ServiceID of the service that the call is attributed to.
ServiceNumber | The service that the call is attributed to, as known to the peripheral.
SkillGroupID | The ICM SkillGroupID of the agent SkillGroup the call is attributed to.
SkillGroupNumber | The number of an agent SkillGroup queue that the call has been added to, as known to the peripheral.
The connection of an outbound call to another network may generate an OnCallReachedNetworkEvent.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SkillGroupPriority</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>NumQueued</td>
<td>The number of calls in the queue for this service.</td>
</tr>
<tr>
<td>NumSkillGroups</td>
<td>The number of Skill Group queues that the call has queued to, up to a maximum of 20. This value also indicates the number of SkillGroupNumber, SkillGroupID and SkillGroupPriority floating fields present in the floating part of the message.</td>
</tr>
</tbody>
</table>

**OnCallReachedNetworkEvent**

The connection of an outbound call to another network may generate an OnCallReachedNetworkEvent.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionDeviceID</td>
<td>The identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of the connection identifier supplied in the ConnectionDeviceID</td>
</tr>
<tr>
<td>TrunkUsedDeviceID</td>
<td>The device identifier of the selected trunk.</td>
</tr>
<tr>
<td>TrunkUsedDeviceIDType</td>
<td>Indicates the type of the device identifier supplied in the TrunkUsedDeviceID</td>
</tr>
<tr>
<td>CalledDeviceID</td>
<td>The device identifier of the called device.</td>
</tr>
<tr>
<td>CalledDeviceIDType</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>Identifies the teleset line being used.</td>
</tr>
<tr>
<td>LineType</td>
<td>Indicates the type of the teleset line.</td>
</tr>
</tbody>
</table>
OnControlFailureConf

The OnControlFailureConf event is generated when an already established connection fails.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FailureCode</td>
<td>Code ID</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>Description of an error</td>
</tr>
<tr>
<td>AgentID</td>
<td>Agent ID that represents a specific client</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>Peripheral ID</td>
</tr>
</tbody>
</table>

OnSnapshotCallConf

The OnSnapshotCallConf event is generated when an already established connection fails.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallType</td>
<td>The general classification of the call type</td>
</tr>
<tr>
<td>NumCTIClients</td>
<td>The number of CTI Clients associated with this call. This value also indicates the number of CTI Client signatures and timestamps</td>
</tr>
<tr>
<td>NumCallDevices</td>
<td>Number of call devices</td>
</tr>
<tr>
<td>NumNamedVariables</td>
<td>Number of named variables</td>
</tr>
<tr>
<td>NumNamedArrays</td>
<td>Number of named arrays</td>
</tr>
<tr>
<td>ANI</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>UserToUserInfo</td>
<td>The ISDN user-to-user information element.</td>
</tr>
<tr>
<td>DNIS</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits</td>
<td>The digits entered by the caller in response to IVR prompting.</td>
</tr>
</tbody>
</table>
### ICallEvents Interface

#### OnServiceInitiated

The initiation of telecommunications service (“dial tone”) at the agent’s teleset may generate an OnCallServiceInitiated event.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>The device identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>The Call ID value assigned to this call by the peripheral or the ICM.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>Identifies the teleset line being used.</td>
</tr>
<tr>
<td>LineType</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>ServiceNumber</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>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ServiceID</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>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>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>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>CallingDeviceType</td>
<td>Indicates the type of the device identifier supplied in the CallingDeviceID field.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>CalledDeviceID (optional)</td>
<td>The device identifier of the device to which the call was diverted.</td>
</tr>
</tbody>
</table>
SkillGroup Object

The SkillGroup object is the abstraction for the data associated with a skill group. The SkillGroup is mainly a representation used for accessing statistics, and can be used in either the agent connection mode or in a supervisor application (monitor mode).

When an agent is logged in, the Session object receives an OnAgentStateEvent for each skill group the agent is logged into. Based on these events, the session create and update one SkillGroup object for each of the agent’s skill groups. The SkillGroups will be accessible directly from the Session, as well as through the Agent Object. The Agent Object maintains a list of the agent’s skills and provides a type of iterator to access skill group statistics for each of the agent’s skills.

The SkillGroup object maintains the agent’s SkillGroup state and any updated skill group statistics.

In a supervisor application (monitor mode), the application creates SkillGroup objects for each of the skills it wishes to monitor. The SkillGroup object containd all of the available statistics as properties. When any statistic changes, an OnSkillGroupStatistic event is received.
Table 7-1 lists the SkillGroup object methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableSkillGroupStatistics/DisableSkillGroupStatistics</td>
<td>Enable or disable skill group statistic messages.</td>
</tr>
<tr>
<td>GetPropertyAttribute</td>
<td>Obtains attribute information for a skill group property.</td>
</tr>
<tr>
<td>GetValue</td>
<td>Retrieves a skill group property based on the property’s name key.</td>
</tr>
<tr>
<td>OnEvent</td>
<td>Specifies an action to take on receipt of a skill group statistic update event.</td>
</tr>
</tbody>
</table>

### EnableSkillGroupStatistics/DisableSkillGroupStatistics

The EnableSkillGroupStatistics and DisableSkillGroupStatistics methods enable or disable skill group statistic messages. The user may specify the skill group numbers of those skill groups for which statistics are to be enabled or disabled.

If you do not specify any skill group numbers, the CTI OS server will decides which skill group statistics to enable or disable. If the agent is a supervisor, the CTI OS server will enable or disable statistics for all skill groups to which the supervisor's team members belong. If the agent is not a supervisor, the server will enable or disable statistics for all skill groups to which the agent belongs.

#### Syntax

**C++**

```cpp
int EnableSkillGroupStatistics(Arguments & rArguments);
int DisableSkillGroupStatistics (Arguments & rArguments);
```
Chapter 7  SkillGroup Object

Methods

**COM (standard COM API)**

```c
HRESULT EnableSkillGroupStatistics ( );
HRESULT DisableSkillGroupStatistics ( );
```

**Parameters**

None.

**Return Values**

**C++**

If the method succeeds, it returns CIL_OK. Otherwise, it returns error code E_CTIOS_IN_FAIOVER if the session is on FailOver mode.

**COM**

If the method succeeds, it returns S_OK. Otherwise, it returns E_FAIL.

**Examples**

**C++**

```c
CSkillGroup * pSkillGroup = NULL;
int nRet = 0;
Arguments & rArgs;
// Get a valid SkillGroup pointer
if (pSkillGroup)
{
    //
    nRet = pSkillGroup -> EnableSkillGroupStatistics (rArgs);
    // Check if the EnableSkillGroupStatistics method
    // failed
}
```
COM C++

HRESULT hr = S_OK;
ISkillGroupPtr  pSkillGroup = NULL;

// First, get SkillGroup object and make sure it is valid pointer
if (pSkillGroup)
{
    hr = pSkillGroup -> DisableSkillGroupStatistics ();
    if(FAILED(hr))
    {
        // You might want to log an error description HERE
    }
    pSkillGroup = NULL;
}
return hr;

VB

Dim skillNumbers As New ctiostools.Arguments
skillNumbers.AddItem( "Skill1", 7 )
skillNumbers.AddItem( "Skill2", 99 )
Dim enableArgs As New ctiostools.Arguments
enableArgs.AddItem( "SkillGroupNumbers", skillNumbers)
agent.EnableSkillGroupStatistics (enableArgs)

GetPropertyAttribute

The GetPropertyAttribute method gets, sets, or retrieves attribute information for any of the properties listed in Table 7-2. For additional information on GetPropertyAttribute, see Chapter 1, “Introduction.” Properties can be accessed using GetValue.
Statistics can be accessed by first using GetValue on the Skill Group object to obtain the “Statistics” arguments array and then using GetValue on the “Statistics” arguments array to obtain the desired value.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SkillGroupNumber</td>
<td>The number of the skill group from the Peripheral.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>The ICM SkillGroupID of the SkillGroup, if available.</td>
</tr>
<tr>
<td>SkillGroupName</td>
<td>The ICM SkillGroupName of the SkillGroup, if available.</td>
</tr>
<tr>
<td>SkillGroupState</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>Value represents skillgroup class</td>
</tr>
</tbody>
</table>

Not all the statistics values listed in Table 7-3 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.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentsLoggedOn</td>
<td>Number of agents that are currently logged on to the skill group</td>
</tr>
<tr>
<td>AgentsAvail</td>
<td>Number of agents in the skill group in Available state.</td>
</tr>
<tr>
<td>AgentsNotReady</td>
<td>Number of agents in the skill group in Not Ready state.</td>
</tr>
<tr>
<td>AgentsReady</td>
<td>Number of agents in the skill group in Ready state.</td>
</tr>
<tr>
<td>AgentsTalkingIn</td>
<td>Number of agents in the skill group currently talking on inbound calls.</td>
</tr>
</tbody>
</table>
### Table 7-3  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AgentsTalkingOut</strong></td>
<td>Number of agents in the skill group currently talking on outbound calls.</td>
</tr>
<tr>
<td><strong>AgentsTalkingOther</strong></td>
<td>Number of agents in the skill group currently talking on internal (not inbound or outbound) calls.</td>
</tr>
<tr>
<td><strong>AgentsWorkNotReady</strong></td>
<td>Number of agents in the skill group in Work Not Ready state.</td>
</tr>
<tr>
<td><strong>AgentsWorkReady</strong></td>
<td>Number of agents in the skill group in Work Ready state.</td>
</tr>
<tr>
<td><strong>AgentsBusyOther</strong></td>
<td>The number of agents in the skill group currently busy with calls assigned to other skill groups.</td>
</tr>
<tr>
<td><strong>AgentsReserved</strong></td>
<td>Number of agents in the skill group in Reserved state.</td>
</tr>
<tr>
<td><strong>AgentsHold</strong></td>
<td>Number of agents in the skill group with calls currently on hold.</td>
</tr>
<tr>
<td><strong>AgentsTalkingAutoOut</strong></td>
<td>Number of agents in the skill group currently talking on AutoOut (predictive) calls.</td>
</tr>
<tr>
<td><strong>AgentsTalkingPreview</strong></td>
<td>Number of agents in the skill group currently talking on outbound Preview calls.</td>
</tr>
<tr>
<td><strong>AgentsTalkingReservation</strong></td>
<td>Number of agents in the skill group currently talking on agent reservation calls.</td>
</tr>
<tr>
<td><strong>RouterCallsQNow</strong></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><strong>LongestRouterCallQNow</strong></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>
</tbody>
</table>
### Table 7-3 SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallsQNow</td>
<td>The number of calls currently queued to the skill group. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>CallsQTimeNow</td>
<td>The total queue time, in seconds, of calls currently queued to the skill group. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>LongestCallQNow</td>
<td>The queue time, in seconds, of the currently queued call that has been queued to the skill group the longest. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>AvailTimeTo5</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>Total time, in seconds, agents in the skill group were logged on during the last five minutes.</td>
</tr>
<tr>
<td>NotReadyTimeTo5</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>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>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 7-3  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentOutCallsTimeTo5</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>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>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>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>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>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>
<tr>
<td>HandledCallsTimeTo5</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>
</tbody>
</table>
### Table 7-3  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IncomingCallsHeldTo5</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>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>Number of internal calls received by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeTo5</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>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>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>AutoOutCallsTo5</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>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>Statistic</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AutoOutCallsTimeTo5</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>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>Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>PreviewCallsTo5</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>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>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>
</tbody>
</table>
### Table 7-3  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviewCallsHeldTo5</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>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>Total number of agent reservation calls completed by agents in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>ReservationCallsTalkTimeTo5</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>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>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>Total number of seconds agent reservation calls were placed on hold by agents in the skill group during the last five minutes.</td>
</tr>
</tbody>
</table>
### Table 7-3  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BargeInCallsTo5</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>Total number of supervisor call intercepts completed in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>MonitorCallsTo5</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>Total number of supervisor call whispers completed in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>EmergencyCallsTo5</td>
<td>Total number of emergency calls completed in the skill group during the last five minutes.</td>
</tr>
<tr>
<td>CallsQ5</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>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>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>Total seconds agents in the skill group were in the Available state during the last half-hour.</td>
</tr>
<tr>
<td>LoggedOnTimeToHalf</td>
<td>Total time, in seconds, agents in the skill group were logged on during the last half-hour.</td>
</tr>
<tr>
<td>NotReadyTimeToHalf</td>
<td>Total seconds agents in the skill group were in the Not Ready state during the last half-hour.</td>
</tr>
</tbody>
</table>
### Table 7-3  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentOutCallsToHalf</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>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>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>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>Total number of seconds outbound ACD calls were placed on hold by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>HandledCallsToHalf</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>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>
</tbody>
</table>
### Table 7-3  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandledCallsAfterCallTimeToHalf</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>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>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>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>Number of internal calls received by agents in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeToHalf</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>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>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>Total number of AutoOut (predictive) calls completed by agents in the skill group during the last half-hour.</td>
</tr>
</tbody>
</table>
**Table 7-3  SkillGroup Statistics (continued)**

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoOutCallsTalkTimeToHalf</td>
<td>Total talk 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 begins after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCallsTimeToHalf</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>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>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>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>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 7-3  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviewCallsTimeToHalf</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>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>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>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>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>
<tr>
<td>ReservationCallsTimeToHalf</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>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>
</tbody>
</table>
### Table 7-3  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationCallsHeldTimeToHalf</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>BargeInCallsToHalf</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>Total number of supervisor call intercepts completed in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>MonitorCallsToHalf</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>Total number of supervisor call whispers completed in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>EmergencyCallsToHalf</td>
<td>Total number of emergency calls completed in the skill group during the last half-hour.</td>
</tr>
<tr>
<td>CallsQHalf</td>
<td>The number of calls queued to the skill group during the current half-hour. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>CallsQTimeHalf</td>
<td>The total queue time, in seconds, of calls queued to the skill group during the current half-hour. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>LongestCallQHalf</td>
<td>The longest queue time, in seconds, of all calls queued to the skill group during the current half-hour. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>AvailTimeToday</td>
<td>Total seconds agents in the skill group were in the Available state.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LoggedOnTimeToday</td>
<td>Total time, in seconds, agents in the skill group were logged on.</td>
</tr>
<tr>
<td>NotReadyTimeToday</td>
<td>Total seconds agents in the skill group were in the Not Ready state.</td>
</tr>
<tr>
<td>AgentOutCallsToday</td>
<td>Total number of completed outbound ACD calls made by agents in the skill group.</td>
</tr>
<tr>
<td>AgentOutCallsTalkTimeToday</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCallsTimeToday</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCallsHeldToday</td>
<td>The total number of completed outbound ACD calls agents in the skill group have placed on hold at least once.</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeToday</td>
<td>Total number of seconds outbound ACD calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>HandledCallsToday</td>
<td>The number of inbound ACD calls handled by agents in the skill group.</td>
</tr>
<tr>
<td>HandledCallsTalkTimeToday</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by agents in the skill group. Includes hold time associated with the call.</td>
</tr>
</tbody>
</table>
### SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandledCallsAfter CallTimeToday</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>Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the skill group. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>IncomingCallsHeldToday</td>
<td>The total number of completed inbound ACD calls agents in the skill group placed on hold at least once.</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeToday</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsRcvdToday</td>
<td>Number of internal calls received by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeToday</td>
<td>Number of seconds spent on internal calls received by agents in the skill group.</td>
</tr>
<tr>
<td>InternalCallsHeldToday</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>Total number of seconds completed internal calls were placed on hold by agents in the skill group.</td>
</tr>
<tr>
<td>AutoOutCallsToday</td>
<td>Total number of AutoOut (predictive) calls completed by agents in the skill group during the current day.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AutoOutCallsTalkTimeToday</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>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>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>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>Total number of outbound Preview calls completed by agents in the skill group during the current day.</td>
</tr>
<tr>
<td>PreviewCallsTalkTimeToday</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>
</tbody>
</table>
### Table 7-3  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviewCallsTimeToday</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>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>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>Total number of agent reservation calls completed by agents in the skill group during the current day.</td>
</tr>
<tr>
<td>ReservationCallsTalkTimeToday</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>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>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>
</tbody>
</table>
Table 7-3  SkillGroup Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationCallsHeldTime Today</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>Total number of supervisor call barge-ins completed in the skill group during the current day.</td>
</tr>
<tr>
<td>InterceptCallsToday</td>
<td>Total number of supervisor call intercepts completed in the skill group during the current day.</td>
</tr>
<tr>
<td>MonitorCallsToday</td>
<td>Total number of supervisor call monitors completed in the skill group during the current day.</td>
</tr>
<tr>
<td>WhisperCallsToday</td>
<td>Total number of supervisor call whispers completed in the skill group during the current day.</td>
</tr>
<tr>
<td>EmergencyCallsToday</td>
<td>Total number of emergency calls completed in the skill group during the current day.</td>
</tr>
<tr>
<td>CallsQToday</td>
<td>The number of calls queued to the skill. This field is set to 0xFFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>CallsQTimeToday</td>
<td>The total queue time, in seconds, of calls queued to the skill group. This field is set to 0xFFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>LongestCallQToday</td>
<td>The longest queue time, in seconds, of all calls queued to the skill group. This field is set to 0xFFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
</tbody>
</table>

Syntax

GetPropertyAttribute(propertyName, attribute requested)
Example

' First get the statistics arguments
Dim args As ctios.Arguments
args = skillGroup.GetValue("Statistics")

' Then get the desired statistics
Dim agentsLoggedOn As Integer
Dim agentsAvail As Integer
agentsLoggedOn = args.GetValue("AgentsLoggedOn")
agentsAvail = args.GetValue("AgentsAvail")

GetValue

The GetValue method retrieves a property (or an array of properties) from the SkillGroup object based on the property’s name key. GetValue takes either a single key name or an array of key names as its required argument, and returns the value associated with that key.

Syntax

GetValue(key)

Example

Dim WithEvents sg as ctios. SkillGroupObject
Dim SkillGroupNumber As String
SkillGroupNumber = sg.GetValue(eSkillGroupNumber)
' Also valid
SkillGroupNumber = sg.GetValue("skillgroupnumber")
OnEvent

The object receives skill group statistic update events. The Agent object enables these when the client initiates a EnableSkillGroupStatistics or skill group statistics may have automatically been enabled on the server on behalf of all clients.

Syntax

OnEvent( event_id, arrayParameters )

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 OnSkillGroupStatisticsUpdated message may contain the following fields as well as the fields listed in Table 7-3.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>The ICM PeripheralID of the ACD on which the agent resides.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>The number of the agent skill group as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not available.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>The ICM SkillGroupID of the skill group. May contain the special value NULL_SKILL_GROUP when not available.</td>
</tr>
</tbody>
</table>
Helper Classes

The CTI OS CIL includes the following helper classes.

- The Arg class is the basic data type used in CIL for any parameter included in methods or events. Objects of this type allow CIL to be fully extensible and reusable.

- Arguments class is a subclass of Arg and its main responsibility is to maintain an array of objects of type Arg. Each object contained in this class is always associated with a key. Elements in an Arguments array can be dynamically added, removed, accessed using a key, or sequentially by a one based index value.

- CILRefArg class is a subclass of Arg and its main responsibility is to store a reference of a CCtiOsObject object. For instance, it can hold reference to a CAgent, CCall, CSkillGroup, CWaitObject or CCtiOsSession.

- The CCtiosException class is normally used within the Arguments class. It provides access to additional information when errors are generated, such as what parameter is in error, memory allocation failed among other.

- CCtiOsObject is the principal base class for the CTI OS Client Library. It serves as the root for all classes such as CAgent, CCall, CSkillGroup, CSession and CWaitObject.
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 8-1.

Table 8-1 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 array of Arg</td>
</tr>
</tbody>
</table>

Table 8-2 lists the available Arg class methods.
Chapter 8 Helper Classes

Arg Class

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 bool CreateInstance(Arg ** ppArg)

COM (standard COM API)

HRESULT CoCreateInstance(..., REFIID riid, ...)

Table 8-2 Arg Class Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateInstance</td>
<td>Creates an Arg object</td>
</tr>
<tr>
<td>SetValue</td>
<td>Sets the data in the Arg object</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>GetValueUInt</td>
<td></td>
</tr>
<tr>
<td>GetValueUShort</td>
<td></td>
</tr>
<tr>
<td>GetValueShort</td>
<td></td>
</tr>
<tr>
<td>GetValueBool</td>
<td></td>
</tr>
<tr>
<td>GetValueString</td>
<td></td>
</tr>
<tr>
<td>Clone</td>
<td>Creates an exact copy of the Arg object</td>
</tr>
<tr>
<td>GetType</td>
<td>Returns the type of the data stored in the argument (one of the values in Table 8-1).</td>
</tr>
<tr>
<td>GetClassID</td>
<td>Returns eArg</td>
</tr>
</tbody>
</table>
**Parameters**

`ppArg`

(output) Pointer to an Arg class object to receive the address of the new object instance.

`riid`

(input) A reference to the identifier of the interface IArg.

**Return Values**

**C++**

The first format, if successful, will return a reference to a new Arg object. Otherwise, it will raise a CCtiosException with iCode set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED.

The second format returns True if successful, False otherwise.

**COM**

Standard COM API error code

**Examples**

**C++**

```c++
try
{
    //First method to create objects
    Arg & arParam = Arg::CreateInstance();

    //Second method to create objects
    bool bAllocOk = false;
    Arg * pParam = NULL;
    bAllocOk = Arg::CreateInstance(&pParam);
    //Do not forget to release the two objects when done
    with them
}
catch (CCtiosException & e)
{
}
```
COM C++

IArg * pParam = NULL;

hr = CoCreateInstance(CLSID_Arg, NULL, CLSCTX_ALL, IID_IArg, (LPVOID *) & pParam);

VB

Dim arParam as Arg
Set arParam = new Arg

SetValue

The SetValue method sets the value in the object.

Syntax

C++

bool SetValue( int iValue );
bool SetValue( unsigned int uiValue );
bool SetValue( unsigned short usValue );
bool SetValue( short sValue );
bool SetValue( bool bValue );
bool SetValue( char * pcValue );
bool SetValue( string& strValue);
bool SetValue( const string& cstrValue);
bool SetValue( Arg & arValue);

COM

HRESULT SetValue([in] VARIANT * pVariant,
[out,retval] VARIANT_BOOL * bRetVal );
Input Parameters

Value
  Integer value to be set
uiValue
  Unsigned integer value to be set
usValue
  Unsigned short integer value to be set.
sValue
  Signed short integer value to be set.
bValue
  Boolean value to be set.
pcValue
  Points to the buffer containing the null terminated character string to be set.
strValue
  Reference to an STL string object containing the character string to be set.
cstrValue
  Constant reference to an STL string object containing the character string to be set.
arValue
  Reference to an Arg object to be set.
pVariant
  Points to a variant parameter that contains the data to be set on the IArg object. The following are the permitted types for this variant.
### Return Values

**C++**

If the method was able to set the value it returns true, otherwise it returns false.

**COM**

If the method was able to set the value it returns VARIANT_TRUE. Otherwise, it returns VARIANT_FALSE.

### Examples

**C++**

```cpp
bool bAllocOk = false;
Arg * pParam = NULL;

bAllocOk = Arg::CreateInstance(&pParam);

if (bAllocOk)
{
    pParam->SetValue(10);
    // Do useful work with the object
    pParam->Release();
}
```
### COM C++

```cpp
VARIANT vParam;
VARIANT_BOOL bRet = VARIANT_FALSE;

VariantInit(&vParam);

hr = CoCreateInstance(CLSID_Arg,
                       NULL,
                       CLSCTX_ALL,
                       IID_IArg, (LPVOID *) & pParam);

vParam.vt = VT_INT;
vParam.intVal = 10;

pParam->SetValue(&vParam, &bRet);
//Do useful work with the object
pParam->Release();
```

### VB

```vb
Dim arParam As Arg
Dim bRet As Boolean
Set arParam = new Arg

bRet = arParam.SetValue(10)

' Do useful work with the object
Set arParam = Nothing
```
GetValueType

The GetValueType method returns the value stored in the object. To extract a specific type of data you invoke the method designated for it.

Syntax

**C++**

```cpp
int GetValueInt();
unsigned int GetValueUInt();
unsigned short GetValueUShort();
short GetValueShort();
string& GetValueString();
bool GetValueBool();
```

```cpp
bool GetValueInt(int * piValue);
bool GetValueUInt(unsigned int * puiValue);
bool GetValueUShort(unsigned short * pusValue);
bool GetValueShort(short * psValue);
bool GetValueBool(bool * pbValue);
bool GetValueString(string* pstrValue);
```

**COM**

```cpp
HRESULT GetValue([out, retval] VARIANT * pVariantOut);
```

**Output Parameters**

- **piValue**
  Points to an integer variable that will receive the value.

- **puiValue**
  Points to an unsigned integer variable that will receive the value.

- **pusValue**
  Points to an unsigned short integer variable that will receive the value.

- **psValue**
  Points to a signed short integer variable that will receive the value.
Arg Class

pbValue

Points to a Boolean variable that will receive the value.

pstrValue

Points to a STL string object that will receive the value.

pVarianOut

Points to a variant variable that will receive the value. The following are the possible types to be returned.

<table>
<thead>
<tr>
<th>Variant Type</th>
<th>Standard C++ Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT_INT</td>
<td>Int</td>
</tr>
<tr>
<td>VT_UINT</td>
<td>Unsigned int</td>
</tr>
<tr>
<td>VT_I2</td>
<td>Short</td>
</tr>
<tr>
<td>VT_UI2</td>
<td>Unsigned short</td>
</tr>
<tr>
<td>VT_BOOL</td>
<td>Bool</td>
</tr>
<tr>
<td>VT_BSTR</td>
<td>string, const string and char *</td>
</tr>
</tbody>
</table>

Return Values

C++

First implementation, if successful, will return the value in the object; otherwise, it will raise a CCtiosException with iCode set to E_CTIOS_INVALID_ARGUMENT.

Second implementation. If the method was able to get the value it returns true. Otherwise, false.

COM

If the method was able to set the value pVarianOut->vt to any of the types listed in the table of pVarianOut, it returns the value in the object. Otherwise it returns VT_EMPTY.
Examples

**C++**

```cpp
try {
    // First method to create objects
    int iPeripheral;
    string strInstrument;

    // Getting peripheral using first implementation
    Arg & arPeripheral = m_Agent.GetValue(_T("PeripheralID"));
    iPeripheral = arPeripheral.GetValueInt();

    // Getting instrument using second implementation
    Arg & arInstrument = m_Agent.GetValue(_T("Instrument"));

    if (arInstrument.GetValueString(&strInstrument)) {
        // Do something interesting with peripheral and instrument
    }

    arPeripheral.Release();
    arInstrument.Release();
}
```

**COM C++**

```cpp
IArg * pInstrument = NULL;
VARIANT vPeripheralID;

// Get peripheral from COM Agent object and assign it to pInstrument

// Retrieving data from argument
VariantInit(&vPeripheralID);

pInstrument->GetValue(&vPeripheralID);
```
if(vPeripheral.vt != VT_EMPTY)
{
    m_iPeripheralID = vPeripheral.intVal;
}

pInstrument->Release()

VB

Dim arInstrument As Arg
Dim iPeripheral As Integer

' Initializes IArg
Set arInstrument = m_Agent.GetValue("Instrument")
iPeripheralID = arInstrument.GetValueInt
Set arInstrument = Nothing

Clone

The Clone method causes the object allocates a new object in memory and to copy its value and type to the new instance.

Syntax

C++

Arg & Clone()

COM

HRESULT Clone([out, retval] LPDISPATCH * pClonedArg);
Chapter 8  Helper Classes

Arg Class

Output Parameters

pClonedArg

Pointer to an IArg interface instance that receive the address of the new object instance.

Return Values

C++

If successful, will return a reference to a new Arg object; otherwise, it will raise a CCtiosException with iCode set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED.

COM

If successful, will return a pointer to an IArg interface; otherwise, it will raise a COM Exception with error code set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED.

Examples

C++

```cpp
try {
    //First method to create objects
    Arg & arDNCopied = arDiledNumber.Clone();
    cout << "Dialing " << arDNCopied.GetValueString();
    arDNCopied.Release();
    arDiledNumber.Release();
} catch (CCtiosException & e) {
    arDiledNumber.Release();
}
```
COM C++

IArg * pParam = NULL;
hr = arDiledNumber.Clone( (LPDISPATCH*) & pParam);
if(CIL_FAILED(hr))
    { MessageBox(NULL,"Failed cloning Argument ","App",MB_STOP);
    }
arDiledNumber->Release();

VB

On Error Goto HandleError
Dim arParam as Arg
Set arParam = arDiledNumber.Clone
    Set arParam = Nothing
    Set arDiledNumber = Nothing
Exit Sub
HandleError:
    MsgBox "Failed cloning Argument"
ArDiledNumber = Nothing

GetType

The GetType method returns the type of the data stored by the argument. See the table listed under the pVarianOut parameter for the GetValueType method for a list of possible types.

Syntax

C++

enumArgTypes GetType()

COM

HRESULT GetType([out,retval] int * nArgType );
Chapter 8  Helper Classes

Arg Class

Output Parameters

nArgType

Pointer to an integer that receive the enumerated constant that identifies data type stored in IArg.

Return Values

C++

Returns the enumerated value that identifies the data type stored in the Arg

COM

If the method invocation succeeded it returns S_OK. Otherwise, it returns CIL_FAILED.

Examples

C++

```
enumArgTypes enType;
for (int nI = 1; nI < arEventParam.NumElements(); nI++)
{
    Arg & arParam = arEventParam.GetelEment(nI);
    enType = arParam.GetType();
    if(enType == ARG_STRING)
    {
        //Doing some work
    }
    arParam.Release();
}
```

COM C++

```
HRESULT hr = S_OK;
int nType;
int nNumElem = 0;
```
arEventParam.NumElements(&nNumElem);
for (int nI = 1; nI < nNumElem; nI++)
{
    CComVariant vParam = arEventParam.GetElement(nI);
    IArg * pParam = (IArg*) vParam.pdispVal
    hr = pParam->GetType(&nType);
    if(nType == ARG_STRING)
    {
        //Doing some work
    }
    pParam->Release();
}

VB

Dim nType As Integer;
Dim nI As Integer
Dim arParam As Arg

For nI = 1 To nI < arEventParam.NumElements()
    Set arParam = arEventParam.GetElement(nI);
    nType = arParam.GetType;
    If nType = ARG_STRING Then
        //Doing some work
    End If
Set arParam = Nothing;
**GetClassID**

The GetClassID method returns the object’s class ID.

**Syntax**

**C++**

```cpp
int GetClassID()
```

**COM**

```cpp
HRESULT GetClassID ([out,retval] int * pClassID);
```

**Output Parameters**

pClassID

Pointer to an integer that receives the enumerated constant that identifies the object class type.

**Return Values**

**C++ and COM**

Retuns eArg if the pointer is referencing an Arg object.

**Examples**

**C++**

```cpp
int nClassID;
for (int nI = 1; nI < arEventParam.NumElements(); nI++)
{
    Arg & arParam = arEventParam.GetElement(nI);
    enType = arParam.GetClassID();
    switch(nClassID)
    {
```

Arg Class

```c
{
    case eArg:
        //Doing some work
        break;
    case eCILRefArg:
        break;
}
arParam.Release();
}
```

**COM C++**

```c
HRESULT hr = S_OK;
int nClassID;
int nNumElem = 0;
arEventParam.NumElements(&nNumElem);
for (int nI = 1; nI < nNumElem; nI++)
{
    CComVariant vParam = arEventParam.GetElement(nI);
    IArg * pParam = (IArg*) vParam.pdispVal
    hr = pParam->GetClassID(&nClassID);
    switch(nClassID)
    {
        case eArg:
            //Doing some work
            break;
        case eCILRefArg:
            break;
    }
    pParam->Release();
}
```

**VB**

```vb
Dim nClassID As Integer
Dim nI As Integer
Dim arParam As Arg

For nI = 1 To nI < arEventParam.NumElements()
    Set arParam = arEventParam.GetElement(nI);
    nClassID = arParam.GetClassID;
```
If nClassID = eArg Then
    //Doing some work
Else
    If nClassID = eCILRefArg Then
End If

Set arParam = Nothing;

Next nI

Arguments Class

The Arguments class provides key/value support. The Arguments class contains a list of values that can be associated with a key. To add an item, use the AddItem method and pass a key value, which must be a string or an integer, and a value, which can be anything. To retrieve the item, use GetValue with a key value. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Items stored in an Arguments array use binary trees and other techniques to provide fast access to any item. They can support very large array of values. Arguments also supports access by index value or by search key. The index value is useful for retrieving items sequentially and may not be as fast as retrieval by key.

The Arguments class methods includes all the Arg class methods plus the methods that are listed in Table 8-3.

Table 8-3 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>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>
</tbody>
</table>
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. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.
Chapter 8  Helper Classes

Arguments Class

Syntax

AddItem( key, value )

Parameters

key
  Key value for the item to be added.
value
  Value of the item to be added.

Returns

Returns True if the entry replaced an entry in Arguments with the same key value.

Clear

The Clear method deletes all the elements from Arguments object.

Syntax

void Clear( )

Parameters

None.

Clone

The Clone method creates a copy of the Arguments structure.

Syntax

Arg & Clone()
Parameters

None.

Returns

A reference to the Arguments structure copy.

CreateInstance

The CreateInstance method creates an object of type Arguments class. This object is released by calling the Release method.

Syntax

static Arguments & CreateInstance()
static bool CreateInstance(Arguments ** ppArguments)

DumpArgs

The DumpArgs method dumps all of the args to a string. It is used for debugging purposes.

Syntax

string DumpArgs()

Parameters

None.
### GetClassID

The GetClassID method returns a ClassID as integer.

**Syntax**

```c
int GetClassID()
```

**Parameters**

None.

**Returns**

Integer value for the ClassID.

### GetElement

The GetElement method is identical 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 zero (0) based. Index should never be less than zero or greater than NumElements minus 1.

**Syntax**

```c
GetElement( index )
```

### GetValue

The GetValue method returns the value stored under a key. This method will return a blank string if the key is invalid. The key can be tested using IsValid. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.
### Arguments Class

#### Syntax

```
GetValue( key )
```

#### Initialize

The `Initialize` method removes all items from an Arguments class. This method *does not* need to be called when an instance is created. It will automatically be in initialize mode.

**Syntax**

```
Initialize()
```

**Parameters**

None.

#### IsValid

The `IsValid` method returns True if key exists in the current Arguments array, otherwise it returns False.

**Syntax**

```
IsValid( key )
```

#### NumElements

The `NumElements` method returns number of elements stored in the current arguments array.

**Syntax**

```
NumElements()
```
Chapter 8    Helper Classes

Arguments Class

Parameters

None.

RemoveItem

The RemoveItem method expects a key. It locate and remove the item by key in Arguments. A True return value means the entry was located. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Syntax

RemoveItem( key )

SetElement

The SetElement method is identical to SetValue, 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 zero (0) based. Index should never be less than zero or greater than NumElements minus 1.

Syntax

SetElement( index, value )

SetValue

The SetValue method sets a value for a key. A True return value means the entry was located. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.
The CILRefArg class is a subclass of the Arg class. Its main responsibility is to store a reference of a CCtiOsObject object. This class allows object references to be included in argument arrays. The object types that can be used are any of the following: CAgent, CCall, CskillGroup, CWaitObject or CetiOsSession.

In addition to the methods inherited from the Arg class, the CILRefArg class contains the methods listed in Table 8-4.

**Table 8-4  CILRefArg Class Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetValue</td>
<td>Encapsulates the pointer to CTI OS object into the CILRefArg object.</td>
</tr>
<tr>
<td>GetValue</td>
<td>Returns the encapsulated pointer in the object.</td>
</tr>
<tr>
<td>GetClassID</td>
<td>Returns eCILRefArg</td>
</tr>
<tr>
<td>GetType</td>
<td>Returns ARG_NOTSET</td>
</tr>
</tbody>
</table>

**SetValue**

Sets the reference to the CTI OS Object in the CILRefArg

**Syntax**

```c++
bool SetValue(CTtiOsObject * 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.

Examples

CILRefArg * pCILRefArg = NULL;
bool bRet = CILRefArg::CreateInstance(&pCILRefArg);
if ((bRet) && (pCILRefArg))
{
    pCILRefArg->SetValue((CCtiOsObject*)(*itObject).second);
    rArrayRef.AddItem(*itObject).first, *pCILRefArg);
    pCILRefArg->Release();
}

GetValue

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

Syntax

CCtiOsObject * GetValue();

Parameters

None.
Return Values

If successful, it returns the value in the object. Otherwise, it will raise a CCTiosException with iCode set to E_CTIOS_INVALID_ARGUMENT.

Examples

```c
CILRefArg & rCILRefArg = (CILRefArg &) GetValue("CurrentAgent");
CAgent * pctiAgent = (CAgent *) rCILRefArg.GetValue();
    if ( pctiAgent )
    {
        pctiAgent->Logout();
    }
```

GetClassID

The GetClassID method returns the object’s class id.

Syntax

```c
int GetClassID()
```

Parameters

None.

Return Values

Returns eCILRefArg if the pointer is referencing a CILRefArg object.
Chapter 8      Helper Classes

Examples

```cpp
int nClassID;
for (int nI = 1; nI < arEventParam.NumElements(); nI++)
{
    Arg & arParam = arEventParam.GetElement(nI);
    enType = arParam.GetClassID();
    switch(nClassID)
    {
        case eArg:
            break;
        case eCILRefArg:
            //Doing some work
            break;
    }
    arParam.Release();
}
```

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.

CCtiOsException Class

**Note**

The CCtiOsException class is only available for C++.

The CCtiOsException class it is normally used within the Arguments class. It provides access to additional information when errors are generated, such as what parameter is in error, memory allocation failed, and so on. Table 8-5 lists the available CCtiOsException class methods.
CCtiosException Class

The CCtiosException constructor initializes an object of type CCtiosException class.

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.

Table 8-5  CCtiosException Class Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCtiosException</td>
<td>Class constructor.</td>
</tr>
<tr>
<td>getCode</td>
<td>Returns the error code that generated the exception.</td>
</tr>
<tr>
<td>getStatus</td>
<td>Returns the error status that generated the exception.</td>
</tr>
<tr>
<td>getString</td>
<td>Returns a text string containing the description of the exception.</td>
</tr>
<tr>
<td>what</td>
<td>Returns a text string containing the description of the exception, the code of an error and the status.</td>
</tr>
</tbody>
</table>
Chapter 8 Helper Classes

CCtiOsException Class

Return Values

None.

Example

String estr;

estr.Format(_T("Arguments::GetValue, Arg with key %d not found."), nKey);
throw CCtiOsException(estr, 0, 0);

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.

Example

Int nRetCode = 0;
String estr;
estr.Format(_T("Arguments::GetValue, Arg with key %d not found."), nKey);
CctiosException HighException(estr, 1, 0);

nRetCode = HighException.GetCode();
The GetStatus method returns the error status that generated the exception.

**Syntax**

```c
int GetStatus ();
```

**Parameters**

None.

**Return Values**

Returns an integer error status that generated the exception.

**Example**

```c
Int nRetStatus = 0;
String estr;
estr.Format(_T("Arguments::GetValue, Arg with key %d not found."),
nKey);
CctiosException HighException(estr, 1, 0);
nRetStatus = HighException. GetStatus ( );
```

**GetString**

The GetString method returns a text string containing the description of the exception.

**Syntax**

```c
const char* GetString();
```
Parameters

None.

Return Values

Returns a text string containing the description of the exception.

Example

```cpp
String estr;
estr.Format(_T("Arguments::GetValue, Arg with key %d not found."), nKey);
CctiosException HighException(estr, 1, 0);
cout << HighException.GetString() << endl;
```

What

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

Syntax

```cpp
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.
Example

String estr;
estr.Format(_T("Arguments::GetValue, Arg with key \%d not found."),
nKey);
CctiosException HighException(estr, 1, 0);

Cout << HighException. What() << endl;

CCtiOsObject Class

Note

The CCtiOsObject class is only available for C++.

CCtiOsObject is the principal base class for the CTI OS Client Library. It serves as the root for all classes such as CAgent, CCall, CSkillGroup, CSession and CWaitObject. CCtiOsObject provides basic services, including:

- Object lifetime control using a reference counting mechanism
- Run-time class information
- Dynamic management of properties

Table 8-6 lists the available CCtiOsObject class methods.
### CCtiOsObject Class

#### DumpProperties

The DumpProperties method dumps the properties and their values in a log file.

**Syntax**

```c
string DumpProperties();
```

**Parameters**

None.

**Return Values**

Returns a text string containing the properties and their values.

---

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DumpProperties</td>
<td>Dumps the properties and their values in a log file.</td>
</tr>
<tr>
<td>GetPropertyName</td>
<td>Returns a property name in a string format.</td>
</tr>
<tr>
<td>GetNumProperties</td>
<td>Returns the number of properties of an object.</td>
</tr>
<tr>
<td>GetValue</td>
<td>Returns the value stored in the argument.</td>
</tr>
<tr>
<td>GetValueInt</td>
<td></td>
</tr>
<tr>
<td>GetValueString</td>
<td></td>
</tr>
<tr>
<td>GetElement</td>
<td>Returns a value of an element.</td>
</tr>
<tr>
<td>GetPropertyAttribute</td>
<td>Returns information about a property.</td>
</tr>
<tr>
<td>SetValue</td>
<td>Sets the data of a specific property.</td>
</tr>
<tr>
<td>IsValid</td>
<td>Checks to see if the property of an object is valid.</td>
</tr>
</tbody>
</table>
Example

```cpp
// Get a valid CCTiOsObject pointer
cout << pCCTiOsObject->DumpProperties().c_str() << endl;
```

**GetPropertyName**

The GetPropertyName method returns a property name in a string format.

**C++**

```cpp
string GetPropertyName(int nProperty);
```

**Input Parameters**

- `nProperty`
  Property or index number

**Return Values**

Returns a property name in a string format.

**Example**

```cpp
string strPropertyName;
int Index = 1; // Get a property of index name 1
// Get a valid CCTiOsObject pointer
strPropertyName = pCCTiOsObject->GetPropertyName(Index);
```

**GetNumProperties**

The GetNumProperties method returns the number of properties of an object.

**Syntax**

```cpp
int GetNumProperties();
```
Parameters

None.

Return Values

Returns the number of properties of an object.

Example

```c
int nNumbProp = 0;
// Get a valid CCtiOsObject pointer
nNumbProp = pCCtiOsObject->GetNumProperties();
```

GetValueType

The GetValueType method returns data of a specific property.

Syntax

```c
Arg &GetValue(string& sKey);
Arg &GetValue(char * pKey);
int GetValueInt(string& sKey);
int GetValueInt(char * pKey);
string GetValueString(string& sKey);
string GetValueString(char * pKey);
```

Input Parameters

sKey

An STL character string containing the property name.

pKey

A pointer to a null terminated string containing the property name.
Return Values

Returns the number of properties of an object.

Example

```csharp
int iPeripheral
    string strInstrument;

    //Getting peripheral using first implementation
    Arg & arPeripheral = m_Agent.GetValue(_T("PeripheralID"));
    iPeripheral = arPeripheral.GetValueInt();

    //Getting instrument using second implementation
    strInstrument = m_Agent.GetValueString(_T("Instrument"));

    //Getting peripheral using third implementation
    iPeripheral = m_Agent.GetValueInt (_T("PeripheralID"));

    arPeripheral.Release();
```

GetElement

The GetElement method is intended to access information in a property that was defined as an array. It returns the information stored in position nElement.

Syntax

```csharp
Arg& GetElement( string& sKey, int nElement);
Arg& GetElement( int nKey, int nElement);
Arg& GetElement( char * pKey, int nElement);
```

Input Parameters

**sKey**

An STL string containing the property name

**pKey**

A pointer to a null terminated string containing the property name
nKey
An enumerated integer that identifies the property.

nElement
A one-based index that corresponds to the n-th element in the array.

**Return Values**

Returns the data of an element.

**GetPropertyAttribute**

The GetPropertyAttribute method returns information about a property. See Chapter 1, “Introduction” for more information.

**Syntax**

```
Arg& GetPropertyAttribute( string& strPropName, 
enumCTIOS_Attribute nAttribute);
Arg& GetPropertyAttribute( int nPropName, 
enumCTIOS_Attribute nAttribute);
Arg& GetPropertyAttribute( char * pPropName, 
enumCTIOS_Attribute nAttribute);
```

**Input Parameters**

- **strPropName**
  An STL string containing the property name.

- **pPropName**
  A pointer to a null terminated string containing the property name.

- **nAttribute**
  An enumerated integer that identifies the property.

**Return Values**

Returns information about a property of an object.
SetValue

The SetValue method sets the value of a property.

Syntax

```cpp
bool SetValue( string& sKey, string& sValue );
bool SetValue( string& sKeyValuePair );
bool SetValue( string& sKey, int nValue );
bool SetValue( const char * pKey, const char * pValue );
bool SetValue( const char * pKeyValuePair );
bool SetValue( const char * pKey, int nValue );
```

Input Parameters

- `sKey`: An STL string containing the property name
- `pKey`: A pointer to a null terminated string containing the property name
- `sKeyValuePair`: An STL string containing the property name and the value to assign. The format of this parameter is: “Key=Value”
- `pValue`: A pointer to a null terminated string containing the value to set in the property;
- `pKeyValuePair`: A pointer to a null terminated string containing the property name and the value to assign. The format of this parameter is: “Key=Value”.
- `nValue`: A numeric value to set in the property.

Return Values

Returns true if the method succeeds, false otherwise.
Example

//Setting peripheral
bool bRet = m_Agent. SetValue (_T("PeripheralID"), 5000);

//Setting instrument
bool bRet = m_Agent. SetValue (_T("Instrument"), _T("2229"));

//Setting Agent ID
bool bRet = m_Agent. SetValue (_T("AgentID=23840"));

IsValid

The IsValid method checks to see if the property of an object is valid.

Syntax

bool IsValid( char * pKey );
bool IsValid( string& sKey );

Input Parameters

sKey
  An STL string containing the property name
pKey
  A pointer to a null terminated string containing the property name

Return Values

Returns true if the method succeeds, false otherwise.

Example

//setting peripheral
if ( m_Agent. IsValid (_T("PeripheralID")))
{
  // Do something if it is valid
}
CTI OS CIL Messages

This appendix lists the messages in the CTI OS CIL message set. This appendix is divided into two sections.

- CIL message equivalents to CTI Server messages
- CIL CTI OS-specific messages

CIL CTI Server Message Equivalents

The following CTI OS message requests, confirmations, and events correspond to the CTI Server messages defined in the Cisco ICM Software CTI Server Message Reference Guide (Protocol Version 8).

Call Requests

- eSetCallDataRequest
- eReleaseCallRequest
- eAlternateCallRequest
- eAnswerCallRequest
- eClearCallRequest
- eClearConnectionRequest
- eConferenceCallRequest
- eConsultationCallRequest
eDeflectCallRequest
eHoldCallRequest
eReconnectCallRequest
eRetrieveCallRequest
eTransferCallRequest
eSnapshotCallRequest
eSendDTMFRequest

Agent Requests

eQueryAgentStateRequest
eSetAgentStateRequest
eMakeCallRequest
eMakePredictiveCallRequest
eSnapshotDeviceRequest
eUserMessageRequest
eQueryAgentStatisticsRequest

Skill Group Requests

eQuerySkillGroupStatisticsRequest

Supervisor Requests

eSessionMonitorStartRequest
eSessionMonitorStopRequest
eMonitorAgentTeamStartRequest
eMonitorAgentTeamStopRequest
eSupervisorAssistRequest
Appendix A  CTI OS CIL Messages

CIL CTI Server Message Equivalents

eEmergencyCallRequest
eSuperviseCallRequest
eAgentTeamConfigRequest
eSetAppDataRequest
eAgentDeskSettingsRequest
eListAgentTeamRequest
eBadCallRequest
eSetDeviceAttributesRequest
eStartRecordingRequest
eStopRecordingRequest

Generic CTI Server Confirmation Events
eControlFailureConf

Call Confirmation Events
eSetCallDataConf
eReleaseCallConf
eAlternateCallConf
eAnswerCallConf
eClearCallConf
eClearConnectionConf
eConferenceCallConf
eConsultationCallConf
eDeflectCallConf
eHoldCallConf
eReconnectCallConf
eRetrieveCallConf
Agent Confirmation Events

eQueryAgentStateConf
eSetAgentStateConf
eSnapshotDeviceConf
eUserMessageConf
eQueryAgentStatisticsConf
eMakeCallConf
eMakePredictiveCallConf

Supervisor Confirmation Events

eSessionMonitorStartConf
eSessionMonitorStopConf
eMonitorAgentTeamStartConf
eMonitorAgentTeamStopConf
eSupervisorAssistConf
eEmergencyCallConf
eSuperviseCallConf
eAgentTeamConfigConf
eSetAppDataConf
eAgentDeskSettingsConf
eListAgentTeamConf
eBadCallConf
eSetDeviceAttributesConf
Error and Failure Events

eFailureConf

eFailureEvent

Call Events

eCallDeliveredEvent
eCallEstablishedEvent
eCallHeldEvent
eCallRetrievedEvent
eCallClearedEvent
eCallConnectionClearedEvent
eCallOriginatedEvent
eCallFailedEvent
eCallConferencedEvent
eCallTransferredEvent
eCallDivertedEvent
eCallServiceInitiatedEvent
eCallQueuedEvent
eCallTranslationRouteEvent
eCallBeginEvent
eCallEndEvent
eCallDataUpdateEvent
eCallReachedNetworkEvent
eCallDequeuedEvent
eAgentPrecallEvent
eAgentPrecallAbortEvent

**Call Recording Events**

eRTPStartedEvent
eRTPStoppedEvent

**Agent Events**

eAgentStateEvent
eUserMessageEvent
eOnNewAgentStatisticsEvent

**Skill Group Events**

eOnNewSkillGroupStatisticsEvent

**Supervisor Events**

eEmergencyCallEvent
eAgentTeamConfigEvent
CIL CTI OS-specific messages

The following messages, requests, and events are specific to CTI OS. Chapters 4 through 7 include descriptions and field definitions for the most commonly occurring messages.

Session Requests

- eOpenConnection
- eCloseConnection
- eSetSessionModeRequest
- eGlobalSettingsDownloadRequest

Single Step Transfer/Conference Requests

- eSingleStepTransferRequest
- eSingleStepConferenceRequest

Supervisor Requests

- eGetAllSupervisedAgentsRequest
- eSetMonitoredAgentStateRequest
- eMonitorAllAgentTeamsRequest

Statistics Requests

- eEnableAgentStatisticsRequest
- eEnableSkillGroupStatisticsRequest
- eDisableAgentStatisticsRequest
- eDisableSkillGroupStatisticsRequest
CIL CTI OS-specific messages

eSubscribeForStatisticsRequest
eUnsubscribeForStatisticsRequest

Timer Service Requests

eStartTimerRequest
eStopTimerRequest

Team Maintenance Requests

eAssignToTeamRequest
eRemoveFromTeamRequest

CTI OS Specific Confirmation Events

eSingleStepTransferConf
eSingleStepConferenceConf

CTI OS Specific Events

eSendChatMessageReq
eSendChatMessageConf
eChatMessageEvent

Session Events

eOnConnection
eOnConnectionClosed
eOnConnectionRejected
eOnConnectionFailure
CIL CTI OS-specific messages

eOnCILError
eCTIOSFailureEvent
eOnHeartbeat
eOnMissingHeartbeat
eOnMonitorModeEstablished
eOnCurrentCallChanged
eSetAgentModeEvent
eOnCurrentAgentReset
eGlobalSettingsDownloadConf
ePreLogoutEvent
ePostLogoutEvent
eLogoutFailedEvent

**Button Enablement Events**

eButtonEnablementMaskChange

**Supervisor Events**

eSupervisorNotifyAgent
eMonitoredAgentStateChange
eMonitoredCallEvent
eNewAgentTeamMember
eSupervisorButtonChange
eStartMonitoringAgentEvent
eDropMonitoredAgentEvent
Filter Events

- eAddFilterEvent
- eRemoveFilterEvent

Name Lookup Service Requests

- eGetAgentNameRequest

Name Lookup Service Event

- eAgentInfoEvent
- eMonitoredAgentInfoEvent

Timeout Events

- eAgentPrecallEventTimeout

Supervised Calls

All the supervised call events are sent to an agent mode application only when the agent logged in the session is a supervisor. Each of the event has its corresponding CTI Server event sent to the agent being supervised.

- eMonitoredCallDeliveredEvent
- eMonitoredCallEstablishedEvent
- eMonitoredCallHeldEvent
- eMonitoredCallRetrievedEvent
- eMonitoredCallClearedEvent
- eMonitoredCallConnectionClearedEvent
- eMonitoredCallOriginatedEvent
- eMonitoredCallFailedEvent
Supervised Agent Events

All the supervised agent events are sent to an agent mode application only when
the agent logged in the session is a supervisor. Each of the event has its
 corresponding CTI Server event sent to the agent being supervised.

eMonitoredAgentStateEvent

Team Maintenance Events

Contains up-to-date information about a team composition.

eTeamSkillGroupDataUpdateEvent,
CTI OS Keywords

Table B-1 lists all the valid keywords that can be used in CTI OS for the following purposes:

- To refer to a property in a CTI OS Object
- To refer to a parameter on a message request
- To create expressions used in a filter

The table shows the three possible forms of a CTI OS keyword.

- The first column is the keyword in text string format. Any CIL implementation can use this form directly provided the keyword is enclosed in double quotes.
- The second column is a constant symbols that maps directly to a text string representation. This form is recommended over the string text representation because if changes to the keywords occur in future versions of CIL, you need only rebuild the executable.
- The third column is an enumerated integer value. This form eliminates the need for an application programmer to deal with character strings when referring to properties or parameters.

Note
Constant symbols and enumerated integers cannot be used to build expressions in a message filter.
## Table B-1 CTI OS Keywords

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvokeID</td>
<td>INVOKEID</td>
<td>ekwInvokeID</td>
</tr>
<tr>
<td>VersionNumber</td>
<td>VERSIONNUMBER</td>
<td>ekwVersionNumber</td>
</tr>
<tr>
<td>IdleTimeout</td>
<td>IDLETIMEOUT</td>
<td>ekwIdleTimeout</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>PERIPHERALID</td>
<td>ekwPeripheralID</td>
</tr>
<tr>
<td>ServiceMask</td>
<td>SERVICEMASK</td>
<td>ekwServiceMask</td>
</tr>
<tr>
<td>CallMsgMask</td>
<td>CALLMSGMASK</td>
<td>ekwCallMsgMask</td>
</tr>
<tr>
<td>AgentStateMask</td>
<td>AGENTSTATEMASK</td>
<td>ekwAgentStateMask</td>
</tr>
<tr>
<td>Reserved</td>
<td>RESERVED</td>
<td>ekwReserved</td>
</tr>
<tr>
<td>PStatus</td>
<td>PGSTATUS</td>
<td>ekwPStatus</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>PERIPHERALTYPE</td>
<td>ekwPeripheralTypeID</td>
</tr>
<tr>
<td>AgentState</td>
<td>AGENTSTATE</td>
<td>ekwAgentState</td>
</tr>
<tr>
<td>Status</td>
<td>STATUS</td>
<td>ekwStatus</td>
</tr>
<tr>
<td>NumCTIClients</td>
<td>NUMCTICLIENTS</td>
<td>ekwNumCTIClients</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>CONNECTIONCALLID</td>
<td>ekwConnectionCallID</td>
</tr>
<tr>
<td>CallType</td>
<td>CALLTYPE</td>
<td>ekwCallType</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>CONNECTIONDEVICEIDTYPE</td>
<td>ekwConnectionDeviceIDType</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>SERVICENUMBER</td>
<td>ekwServiceNumber</td>
</tr>
<tr>
<td>ServiceID</td>
<td>SERVICEID</td>
<td>ekwServiceID</td>
</tr>
<tr>
<td>AlertingDeviceType</td>
<td>ALERTINGDEVICEIDTYPE</td>
<td>ekwAlertingDeviceType</td>
</tr>
<tr>
<td>CallingDeviceType</td>
<td>CALLINGDEVICEIDTYPE</td>
<td>ekwCallingDeviceIDType</td>
</tr>
<tr>
<td>CalledDeviceType</td>
<td>CALLEDDEVICEIDTYPE</td>
<td>ekwCalledDeviceType</td>
</tr>
<tr>
<td>LastRedirectDeviceType</td>
<td>LASTREDIRECTDEVICEIDTYPE</td>
<td>ekwLastRedirectDeviceType</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>LOCALCONNECTIONSTATE</td>
<td>ekwLocalConnectionState</td>
</tr>
<tr>
<td>EventCause</td>
<td>EVENTCAUSE</td>
<td>ekwEventCause</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>SKILLGROUPNUMBER</td>
<td>ekwSkillGroupNumber</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>SKILLGROUPID</td>
<td>ekwSkillGroupID</td>
</tr>
<tr>
<td>AnsweringDeviceType</td>
<td>ANSWERINGDEVICEIDTYPE</td>
<td>ekwAnsweringDeviceType</td>
</tr>
<tr>
<td>HoldingDeviceType</td>
<td>HOLDINGDEVICEIDTYPE</td>
<td>ekwHoldingDeviceType</td>
</tr>
</tbody>
</table>
### Table B-1  CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RetrievingDeviceType</td>
<td>RETRIEVINGDEVICETYPE</td>
<td>ekwRetrievingDeviceType</td>
</tr>
<tr>
<td>ReleasingDeviceType</td>
<td>RELEASINGDEVICETYPE</td>
<td>ekwReleasingDeviceType</td>
</tr>
<tr>
<td>FailingDeviceType</td>
<td>FAILINGDEVICETYPE</td>
<td>ekwFailingDeviceType</td>
</tr>
<tr>
<td>PrimaryDeviceIDType</td>
<td>PRIMARYDEVICEIDTYPE</td>
<td>ekwPrimaryDeviceIDType</td>
</tr>
<tr>
<td>PrimaryCallID</td>
<td>PRIMARYCALLID</td>
<td>ekwPrimaryCallID</td>
</tr>
<tr>
<td>NumParties</td>
<td>NUMPARTIES</td>
<td>ekwNumParties</td>
</tr>
<tr>
<td>SecondaryDeviceIDType</td>
<td>SECONDARYDEVICEIDTYPE</td>
<td>ekwSecondaryDeviceIDType</td>
</tr>
<tr>
<td>SecondaryCallID</td>
<td>SECONDARYCALLID</td>
<td>ekwSecondaryCallID</td>
</tr>
<tr>
<td>ControllerDeviceType</td>
<td>CONTROLLERDEVICETYPE</td>
<td>ekwControllerDeviceType</td>
</tr>
<tr>
<td>AddedPartyDeviceType</td>
<td>ADDPARTYDEVICETYPE</td>
<td>ekwAddedPartyDeviceType</td>
</tr>
<tr>
<td>TransferringDeviceType</td>
<td>TRANSFERRINGDEVICETYPE</td>
<td>ekwTransferringDeviceType</td>
</tr>
<tr>
<td>TransferredDeviceType</td>
<td>TRANSFERREDDEVICETYPE</td>
<td>ekwTransferredDeviceType</td>
</tr>
<tr>
<td>DivertingDeviceType</td>
<td>DIVERTINGDEVICETYPE</td>
<td>ekwDivertingDeviceType</td>
</tr>
<tr>
<td>StateDuration</td>
<td>STATEDURATION</td>
<td>ekwStateDuration</td>
</tr>
<tr>
<td>QueueDeviceType</td>
<td>QUEUEDEVICETYPE</td>
<td>ekwQueueDeviceType</td>
</tr>
<tr>
<td>NumQueued</td>
<td>NUMQUEUED</td>
<td>ekwNumQueued</td>
</tr>
<tr>
<td>SystemEventID</td>
<td>SYSTEMEVENTID</td>
<td>ekwSystemEventID</td>
</tr>
<tr>
<td>SystemEventArg1</td>
<td>SYSTEMEVENTARG1</td>
<td>ekwSystemEventArg1</td>
</tr>
<tr>
<td>SystemEventArg2</td>
<td>SYSTEMEVENTARG2</td>
<td>ekwSystemEventArg2</td>
</tr>
<tr>
<td>SystemEventArg3</td>
<td>SYSTEMEVENTARG3</td>
<td>ekwSystemEventArg3</td>
</tr>
<tr>
<td>LineHandle</td>
<td>LINEHANDLE</td>
<td>ekwLineHandle</td>
</tr>
<tr>
<td>LineType</td>
<td>LINETYPE</td>
<td>ekwLineType</td>
</tr>
<tr>
<td>NewConnectionCallID</td>
<td>NEWCONNECTIONCALLID</td>
<td>ekwNewConnectionCallID</td>
</tr>
<tr>
<td>NewConnectionDeviceIDType</td>
<td>NEWCONNECTIONDEVICEIDTYPE</td>
<td>ekwNewConnectionDeviceIDType</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>SKILLGROUPPRIORITY</td>
<td>ekwSkillGroupPriority</td>
</tr>
<tr>
<td>ObjectState</td>
<td>OBJECTSTATE</td>
<td>ekwObjectState</td>
</tr>
<tr>
<td>TrunkUsedDeviceType</td>
<td>TRUNKUSEDDEVICETYPE</td>
<td>ekwTrunkUsedDeviceType</td>
</tr>
<tr>
<td>FailureCode</td>
<td>FAILURECODE</td>
<td>ekwFailureCode</td>
</tr>
<tr>
<td>NumSkillGroups</td>
<td>NUMSKILLGROUPS</td>
<td>ekwNumSkillGroups</td>
</tr>
<tr>
<td>Text String</td>
<td>Constant String</td>
<td>Enumerated Integer Value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>EventReasonCode</td>
<td>EVENTREASONCODE</td>
<td>ekwEventReasonCode</td>
</tr>
<tr>
<td>ActiveConnectionCallID</td>
<td>ACTIVECONNECTIONCALLID</td>
<td>ekwActiveConnectionCallID</td>
</tr>
<tr>
<td>ActiveConnectionDeviceIDType</td>
<td>ACTIVECONNECTIONDEVICEIDTYPE</td>
<td>ekwActiveConnectionDeviceIDType</td>
</tr>
<tr>
<td>ActiveConnectionDeviceID</td>
<td>ACTIVECONNECTIONDEVICEID</td>
<td>ekwActiveConnectionDeviceID</td>
</tr>
<tr>
<td>OtherConnectionCallID</td>
<td>OTHERCONNECTIONCALLID</td>
<td>ekwOtherConnectionCallID</td>
</tr>
<tr>
<td>OtherConnectionDeviceID</td>
<td>OTHERCONNECTIONDEVICEID</td>
<td>ekwOtherConnectionDeviceID</td>
</tr>
<tr>
<td>OtherConnectionDeviceIDType</td>
<td>OTHERCONNECTIONDEVICEIDTYPE</td>
<td>ekwOtherConnectionDeviceIDType</td>
</tr>
<tr>
<td>HeldConnectionCallID</td>
<td>HELDCONNECTIONCALLID</td>
<td>ekwHeldConnectionCallID</td>
</tr>
<tr>
<td>HeldConnectionDeviceID</td>
<td>HELDCONNECTIONDEVICEID</td>
<td>ekwHeldConnectionDeviceID</td>
</tr>
<tr>
<td>HeldConnectionDeviceIDType</td>
<td>HELDCONNECTIONDEVICEIDTYPE</td>
<td>ekwHeldConnectionDeviceIDType</td>
</tr>
<tr>
<td>CallPlacementType</td>
<td>CALLPLACEMENTTYPE</td>
<td>ekwCallPlacementType</td>
</tr>
<tr>
<td>CallMannerType</td>
<td>CALLMANNERTYPE</td>
<td>ekwCallMannerType</td>
</tr>
<tr>
<td>ConsultType</td>
<td>CONSULTTYPE</td>
<td>ekwConsultType</td>
</tr>
<tr>
<td>FacilityType</td>
<td>FACILITYTYPE</td>
<td>ekwFacilityType</td>
</tr>
<tr>
<td>Priority</td>
<td>PRIORITY</td>
<td>ekwPriority</td>
</tr>
<tr>
<td>PostRoute</td>
<td>POSTROUTE</td>
<td>ekwPostRoute</td>
</tr>
<tr>
<td>Reservation</td>
<td>RESERVATION</td>
<td>ekwReservation</td>
</tr>
<tr>
<td>AlertRings</td>
<td>ALERTRINGS</td>
<td>ekwAlertRings</td>
</tr>
<tr>
<td>AlertCallerFirst</td>
<td>ALERTCALLERFIRST</td>
<td>ekwAlertCallerFirst</td>
</tr>
<tr>
<td>AnsweringMachine</td>
<td>ANSWERINGMACHINE</td>
<td>ekwAnsweringMachine</td>
</tr>
<tr>
<td>AllocationState</td>
<td>ALLOCATIONSTATE</td>
<td>ekwAllocationState</td>
</tr>
<tr>
<td>TypeOfDevice</td>
<td>TYPEOFDEVICE</td>
<td>ekwTypeOfDevice</td>
</tr>
<tr>
<td>ClassOfDevice</td>
<td>CLASSOFDEVICE</td>
<td>ekwClassOfDevice</td>
</tr>
<tr>
<td>NumLines</td>
<td>NUMLINES</td>
<td>ekwNumLines</td>
</tr>
<tr>
<td>NumCallDevices</td>
<td>NUMCALLDEVICES</td>
<td>ekwNumCallDevices</td>
</tr>
<tr>
<td>NumCalls</td>
<td>NUMCALLS</td>
<td>ekwNumCalls</td>
</tr>
<tr>
<td>ToneDuration</td>
<td>TONEDURATION</td>
<td>ekwToneDuration</td>
</tr>
</tbody>
</table>
## Appendix B  CTI OS Keywords

**Table B-1  CTI OS Keywords (continued)**

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PauseDuration</td>
<td>PAUSEDURATION</td>
<td>ekwPauseDuration</td>
</tr>
<tr>
<td>SkillGroupState</td>
<td>SKILLGROUPSTATE</td>
<td>ekwSkillGroupState</td>
</tr>
<tr>
<td>QueryDeviceDataType</td>
<td>QUERYDEVICEDATATYPE</td>
<td>ekwQueryDeviceDataType</td>
</tr>
<tr>
<td>PeripheralErrorCode</td>
<td>PERIPHERALERRORCODE</td>
<td>ekwPeripheralErrorCode</td>
</tr>
<tr>
<td>AgentWorkMode</td>
<td>AGENTWORKMODE</td>
<td>ekwAgentWorkMode</td>
</tr>
<tr>
<td>MonitorID</td>
<td>MONITORID</td>
<td>ekwMonitorID</td>
</tr>
<tr>
<td>MonitoredDeviceType</td>
<td>MONITOREDDEVICETYPE</td>
<td>ekwMonitoredDeviceType</td>
</tr>
<tr>
<td>CallOption</td>
<td>CALLOPTION</td>
<td>ekwCallOption</td>
</tr>
<tr>
<td>DestinationCountry</td>
<td>DESTINATIONCOUNTRY</td>
<td>ekwDestinationCountry</td>
</tr>
<tr>
<td>AnswerDetectMode</td>
<td>ANSWERDTECTMODE</td>
<td>ekwAnswerDetectMode</td>
</tr>
<tr>
<td>AnswerDetectTime</td>
<td>ANSWERDTECTTIME</td>
<td>ekwAnswerDetectTime</td>
</tr>
<tr>
<td>AnswerDetectControl1</td>
<td>ANSWERDTECTCONTROL1</td>
<td>ekwAnswerDetectControl1</td>
</tr>
<tr>
<td>AnswerDetectControl2</td>
<td>ANSWERDTECTCONTROL2</td>
<td>ekwAnswerDetectControl2</td>
</tr>
<tr>
<td>ClientPort</td>
<td>CLIENTPORT</td>
<td>ekwClientPort</td>
</tr>
<tr>
<td>SessionID</td>
<td>SESSIONID</td>
<td>ekwSessionID</td>
</tr>
<tr>
<td>ServicesGranted</td>
<td>SERVICESGRANTED</td>
<td>ekwServicesGranted</td>
</tr>
<tr>
<td>PeripheralOnline</td>
<td>PERIPHERALONLINE</td>
<td>ekwPeripheralOnline</td>
</tr>
<tr>
<td>Distribution</td>
<td>DISTRIBUTION</td>
<td>ekwDistribution</td>
</tr>
<tr>
<td>CallsQNow</td>
<td>CALLSQNOW</td>
<td>ekwCallsQNow</td>
</tr>
<tr>
<td>CallsQTimeNow</td>
<td>CALLSQTIMENOW</td>
<td>ekwCallsQTimeNow</td>
</tr>
<tr>
<td>LongestCallQNow</td>
<td>LONGESTCALLQNOW</td>
<td>ekwLongestCallQNow</td>
</tr>
<tr>
<td>AvailTimeToHalf</td>
<td>AVAILTIMETOHALF</td>
<td>ekwAvailTimeToHalf</td>
</tr>
<tr>
<td>LoggedOnTimeToHalf</td>
<td>LOGGEDONTIMETOHALF</td>
<td>ekwLoggedOnTimeToHalf</td>
</tr>
<tr>
<td>NotReadyTimetoHalf</td>
<td>NOTREADYTIMETOHALF</td>
<td>ekwNotReadyTimetoHalf</td>
</tr>
<tr>
<td>AgentOutCallsToHalf</td>
<td>AGENTOUTCALLSTOHALF</td>
<td>ekwAgentOutCallsToHalf</td>
</tr>
<tr>
<td>AgentOutCallsTalkTimeToHalf</td>
<td>AGENTOUTCALLSTALKTIMETOHALF</td>
<td>ekwAgentOutCallsTalkTimeToHalf</td>
</tr>
<tr>
<td>AgentOutCallsTimeToHalf</td>
<td>AGENTOUTCALLSTIMETOHALF</td>
<td>ekwAgentOutCallsTimeToHalf</td>
</tr>
<tr>
<td>AgentOutCallsHeldToHalf</td>
<td>AGENTOUTCALLSHELDTOHALF</td>
<td>ekwAgentOutCallsHeldToHalf</td>
</tr>
</tbody>
</table>
### Table B-1 CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentOutCalls Held Time To Half</td>
<td>AGENTOUTCALLSHELDTIMETOHALF</td>
<td>ekwAgentOutCallsHeldTimeToHalf</td>
</tr>
<tr>
<td>Handled Calls To Half</td>
<td>HANDLEDCALLSTOHALF</td>
<td>ekwHandledCallsToHalf</td>
</tr>
<tr>
<td>Handled Calls Talk Time To Half</td>
<td>HANDLEDSCALLSTALKTIMETOHALF</td>
<td>ekwHandledCallsTalkTimeToHalf</td>
</tr>
<tr>
<td>Handled Calls Time To Half</td>
<td>HANDLEDSCALLSTIMETOHALF</td>
<td>ekwHandledCallsTimeToHalf</td>
</tr>
<tr>
<td>Incoming Calls Held To Half</td>
<td>INCOMINGCALLSHELDTOHALF</td>
<td>ekwIncomingCallsHeldToHalf</td>
</tr>
<tr>
<td>Incoming Calls Held Time To Half</td>
<td>INCOMINGCALLSHELDTIMETOHALF</td>
<td>ekwIncomingCallsHeldTimeToHalf</td>
</tr>
<tr>
<td>Internal Calls Rcvd To Half</td>
<td>INTERNALCALLSRCVDTOHALF</td>
<td>ekwInternalCallsRcvdToHalf</td>
</tr>
<tr>
<td>Internal Calls Rcvd Time To Half</td>
<td>INTERNALCALLSRCVDTIMETOHALF</td>
<td>ekwInternalCallsRcvdTimeToHalf</td>
</tr>
<tr>
<td>Internal Calls Held To Half</td>
<td>INTERNALCALLSHELDTOHALF</td>
<td>ekwInternalCallsHeldToHalf</td>
</tr>
<tr>
<td>Internal Calls Held Time To Half</td>
<td>INTERNALCALLSHELDTIMETOHALF</td>
<td>ekwInternalCallsHeldTimeToHalf</td>
</tr>
<tr>
<td>Calls Q Half</td>
<td>CALLSQHALF</td>
<td>ekwCallsQHalf</td>
</tr>
<tr>
<td>Calls Q Time Half</td>
<td>CALLSQTIMEHALF</td>
<td>ekwCallsQTimeHalf</td>
</tr>
<tr>
<td>Longest Call Q Half</td>
<td>LONGESTCALLQHALF</td>
<td>ekwLongestCallQHalf</td>
</tr>
<tr>
<td>Avail Time Today</td>
<td>AVAILTIMETODAY</td>
<td>ekwAvailTimeToday</td>
</tr>
<tr>
<td>Logged On Time Today</td>
<td>LOGGEDONTIMETODAY</td>
<td>ekwLoggedOnTimeToday</td>
</tr>
<tr>
<td>Not Ready Time Today</td>
<td>NOTREADYTIMETODAY</td>
<td>ekwNotReadyTimeToday</td>
</tr>
<tr>
<td>Agent Out Calls Today</td>
<td>AGENTOUTCALLSTODAY</td>
<td>ekwAgentOutCallsToday</td>
</tr>
<tr>
<td>Agent Out Calls Talk Time To Day</td>
<td>AGENTOUTCALLSTALKTIMETOSECOND</td>
<td>ekwAgentOutCallsTalkTimeToDay</td>
</tr>
<tr>
<td>Agent Out Calls Time Today</td>
<td>AGENTOUTCALLSTIMETOSECOND</td>
<td>ekwAgentOutCallsTimeToday</td>
</tr>
<tr>
<td>Agent Out Calls Held To Half</td>
<td>AGENTOUTCALLSHELDSTIMETOSECOND</td>
<td>ekwAgentOutCallsHeldToHalf</td>
</tr>
<tr>
<td>Agent Out Calls Held Time To Day</td>
<td>AGENTOUTCALLSHELDTIMETOSECOND</td>
<td>ekwAgentOutCallsHeldTimeToDay</td>
</tr>
<tr>
<td>Handled Calls Today</td>
<td>HANDLEDCALLSTODAY</td>
<td>ekwHandledCallsToday</td>
</tr>
<tr>
<td>Handled Calls Talk Time To Day</td>
<td>HANDLEDCALLSTALKTIMETOSECOND</td>
<td>ekwHandledCallsTalkTimeToDay</td>
</tr>
<tr>
<td>Handled Calls Time To Day</td>
<td>HANDLEDCALLSTIMETOSECOND</td>
<td>ekwHandledCallsTimeToDay</td>
</tr>
<tr>
<td>Incoming Calls Held To Day</td>
<td>INCOMINGCALLSHELDSTIMETOSECOND</td>
<td>ekwIncomingCallsHeldToDay</td>
</tr>
<tr>
<td>Incoming Calls Held Time To Day</td>
<td>INCOMINGCALLSHELDTIMETOSECOND</td>
<td>ekwIncomingCallsHeldTimeToDay</td>
</tr>
<tr>
<td>Internal Calls Rcvd Today</td>
<td>INTERNALCALLSRCVDTODAY</td>
<td>ekwInternalCallsRcvdToday</td>
</tr>
</tbody>
</table>
### Table B-1  CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>InternalCallsRecvdTimeToday</td>
<td>INTERNALCALLSRECVDTIMETODAY</td>
<td>ekwInternalCallsRecvdTimeToday</td>
</tr>
<tr>
<td>InternalCallsHeldToday</td>
<td>INTERNALCALLSHELDTODAY</td>
<td>ekwInternalCallsHeldToday</td>
</tr>
<tr>
<td>InternalCallsHeldTimeToday</td>
<td>INTERNALCALLSHELDTIMETODAY</td>
<td>ekwInternalCallsHeldTimeToday</td>
</tr>
<tr>
<td>CallsQToday</td>
<td>CALLSQTODAY</td>
<td>ekwCallsQToday</td>
</tr>
<tr>
<td>CallsQTimeToday</td>
<td>CALLSQTIMETODAY</td>
<td>ekwCallsQTimeToday</td>
</tr>
<tr>
<td>LongestCallQToday</td>
<td>LONGESTCALLQTODAY</td>
<td>ekwLongestCallQToday</td>
</tr>
<tr>
<td>AvailTimeSession</td>
<td>AVAILTIMESSESSION</td>
<td>ekwAvailTimeSession</td>
</tr>
<tr>
<td>LoggedOnTimeSession</td>
<td>LOGGEDONTIMESSESSION</td>
<td>ekwLoggedInTimeSession</td>
</tr>
<tr>
<td>NotReadyTimeSession</td>
<td>NOTREADYTIMESSESSION</td>
<td>ekwNotReadyTimeSession</td>
</tr>
<tr>
<td>AgentOutCallsSession</td>
<td>AGENTOUTCALLSSESSION</td>
<td>ekwAgentOutCallsSession</td>
</tr>
<tr>
<td>AgentOutCallsTalkTimeSession</td>
<td>AGENTOUTCALLSTALKTIMESSESSION</td>
<td>ekwAgentOutCallsTalkTimeSession</td>
</tr>
<tr>
<td>AgentOutCallsTimeSession</td>
<td>AGENTOUTCALLSTIMESSESSION</td>
<td>ekwAgentOutCallsTimeSession</td>
</tr>
<tr>
<td>AgentOutCallsHeldSession</td>
<td>AGENTOUTCALLSHELDSESSION</td>
<td>ekwAgentOutCallsHeldSession</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeSession</td>
<td>INCOMINGCALLSHELDSESSION</td>
<td>ekwIncomingCallsHeldSession</td>
</tr>
<tr>
<td>InternalCallsSession</td>
<td>INTERNALCALLSESSION</td>
<td>ekwInternalCallsSession</td>
</tr>
<tr>
<td>InternalCallsTimeSession</td>
<td>INTERNALCALLSTIMESSESSION</td>
<td>ekwInternalCallsTimeSession</td>
</tr>
<tr>
<td>InternalCallsRecvdSession</td>
<td>INTERNALCALLSRCVDSESSION</td>
<td>ekwInternalCallsRecvdSession</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeSession</td>
<td>INTERNALCALLSRCVDTIMESSESSION</td>
<td>ekwInternalCallsRcvdTimeSession</td>
</tr>
<tr>
<td>InternalCallsHeldSession</td>
<td>INTERNALCALLSHELDSESSION</td>
<td>ekwInternalCallsHeldSession</td>
</tr>
<tr>
<td>InternalCallsHeldTimeSession</td>
<td>INTERNALCALLSHELDTIMESSESSION</td>
<td>ekwInternalCallsHeldTimeSession</td>
</tr>
<tr>
<td>InternalCallsToday</td>
<td>INTERNALCALLSTODAY</td>
<td>ekwInternalCallsToday</td>
</tr>
<tr>
<td>Text String</td>
<td>Constant String</td>
<td>Enumerated Integer Value</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>InternalCallsTimeToday</td>
<td>INTERNALCALLSTIMETODAY</td>
<td>ekwInternalCallsTimeToday</td>
</tr>
<tr>
<td>NumNamedVariables</td>
<td>NUMNAMEDVARIABLES</td>
<td>ekwNumNamedVariables</td>
</tr>
<tr>
<td>NumNamedArrays</td>
<td>NUMNAMEDARRAYS</td>
<td>ekwNumNamedArrays</td>
</tr>
<tr>
<td>HandledCallsAfterCallTime Session</td>
<td>HANDLEDCALLSFORMANCESESSION</td>
<td>ekwHandledCallsAfterCallTimeSession</td>
</tr>
<tr>
<td>HandledCallsAfterCallTime ToHalf</td>
<td>HANDLEDCALLSFORMANCETIMETOHALF</td>
<td>ekwHandledCallsAfterCallTimeToHalf</td>
</tr>
<tr>
<td>HandledCallsAfterCallTime Today</td>
<td>HANDLEDCALLSFORMANCETIMETODAY</td>
<td>ekwHandledCallsAfterCallTimeToday</td>
</tr>
<tr>
<td>MaxActiveCalls</td>
<td>MAXACTIVECALLS</td>
<td>ekwMaxActiveCalls</td>
</tr>
<tr>
<td>MaxHeldCalls</td>
<td>MAXHELDCALLS</td>
<td>ekwMaxHeldCalls</td>
</tr>
<tr>
<td>MaxDevicesInConference</td>
<td>MAXDEVICESINCONFERENCE</td>
<td>ekwMaxDevicesInConference</td>
</tr>
<tr>
<td>TransferConferenceSetup</td>
<td>TRANSFERCONFERENCESETUP</td>
<td>ekwTransferConferenceSetup</td>
</tr>
<tr>
<td>CallEventsSupported</td>
<td>CALLEVENTSSUPPORTED</td>
<td>ekwCallEventsSupported</td>
</tr>
<tr>
<td>CallControlSupported</td>
<td>CALLCONTROLSUPPORTED</td>
<td>ekwCallControlSupported</td>
</tr>
<tr>
<td>OtherFeaturesSupported</td>
<td>OTHERFEATURESUPPORTED</td>
<td>ekwOtherFeaturesSupported</td>
</tr>
<tr>
<td>CCTimestamp</td>
<td>CCTIMESTAMP</td>
<td>ekwCCTimestamp</td>
</tr>
<tr>
<td>CallVariableMask</td>
<td>CALLVARIABLEMASK</td>
<td>ekwCallVariableMask</td>
</tr>
<tr>
<td>CalledPartyDisposition</td>
<td>CALLEDPARTYDISPOSITION</td>
<td>ekwCalledPartyDisposition</td>
</tr>
<tr>
<td>AgentsLoggedOn</td>
<td>AGENTSLOGGEDON</td>
<td>ekwAgentsLoggedOn</td>
</tr>
<tr>
<td>AgentsAvail</td>
<td>AGENTSAVAIL</td>
<td>ekwAgentsAvail</td>
</tr>
<tr>
<td>AgentsNotReady</td>
<td>AGENTSNOTREADY</td>
<td>ekwAgentsNotReady</td>
</tr>
<tr>
<td>AgentsReady</td>
<td>AGENTSREADY</td>
<td>ekwAgentsReady</td>
</tr>
<tr>
<td>AgentsTalkingIn</td>
<td>AGENTSTALKINGIN</td>
<td>ekwAgentsTalkingIn</td>
</tr>
<tr>
<td>AgentsTalkingOut</td>
<td>AGENTSTALKINGOUT</td>
<td>ekwAgentsTalkingOut</td>
</tr>
<tr>
<td>AgentsTalkingOther</td>
<td>AGENTSTALKINGOTHER</td>
<td>ekwAgentsTalkingOther</td>
</tr>
<tr>
<td>AgentsWorkNotReady</td>
<td>AGENTSWORKNOTREADY</td>
<td>ekwAgentsWorkNotReady</td>
</tr>
<tr>
<td>AgentsWorkReady</td>
<td>AGENTSWORKREADY</td>
<td>ekwAgentsWorkReady</td>
</tr>
<tr>
<td>AgentsBusyOther</td>
<td>AGENTSBUSYOTHER</td>
<td>ekwAgentsBusyOther</td>
</tr>
</tbody>
</table>
### Table B-1  CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentsReserved</td>
<td>AGENTSRESERVED</td>
<td>ekwAgentsReserved</td>
</tr>
<tr>
<td>AgentsHold</td>
<td>AGENTS HOLD</td>
<td>ekwAgentsHold</td>
</tr>
<tr>
<td>AvailTimeTo5</td>
<td>AVAILTIMETO5</td>
<td>ekwAvailTimeTo5</td>
</tr>
<tr>
<td>LoggedOnTimeTo5</td>
<td>LOGGEDONTIMETO5</td>
<td>ekwLoggedOnTimeTo5</td>
</tr>
<tr>
<td>NotReadyTimeTo5</td>
<td>NOTREADYTIMETO5</td>
<td>ekwNotReadyTimeTo5</td>
</tr>
<tr>
<td>AgentOutCallsTo5</td>
<td>AGENTOUTCALLSTO5</td>
<td>ekwAgentOutCallsTo5</td>
</tr>
<tr>
<td>AgentOutCallsTalkTimeTo5</td>
<td>AGENTOUTCALLSTALKTIMETO5</td>
<td>ekwAgentOutCallsTalkTimeTo5</td>
</tr>
<tr>
<td>AgentOutCallsTimeTo5</td>
<td>AGENTOUTCALLSTIMETO5</td>
<td>ekwAgentOutCallsTimeTo5</td>
</tr>
<tr>
<td>AgentOutCallsHeldTo5</td>
<td>AGENTOUTCALLSHELDTO5</td>
<td>ekwAgentOutCallsHeldTo5</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeTo5</td>
<td>AGENTOUTCALLSHELDTIMETO5</td>
<td>ekwAgentOutCallsHeldTimeTo5</td>
</tr>
<tr>
<td>HandledCallsTo5</td>
<td>HANDLEDCALLSTO5</td>
<td>ekwHandledCallsTo5</td>
</tr>
<tr>
<td>HandledCallsTalkTimeTo5</td>
<td>HANDLEDCALLSTALKTIMETO5</td>
<td>ekwHandledCallsTalkTimeTo5</td>
</tr>
<tr>
<td>HandledCallsAfterCallTimeTo5</td>
<td>HANDLEDCALLSAFTERCALLTIMETO5</td>
<td>ekwHandledCallsAfterCallTimeTo5</td>
</tr>
<tr>
<td>HandledCallsTimeTo5</td>
<td>HANDLEDCALLSTIMETO5</td>
<td>ekwHandledCallsTimeTo5</td>
</tr>
<tr>
<td>IncomingCallsHeldTo5</td>
<td>INCOMINGCALLSHELDTO5</td>
<td>ekwIncomingCallsHeldTo5</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeTo5</td>
<td>INCOMINGCALLSHELDTIMETO5</td>
<td>ekwIncomingCallsHeldTimeTo5</td>
</tr>
<tr>
<td>InternalCallsRcvdTo5</td>
<td>INTERNALCALLSRCVDTO5</td>
<td>ekwInternalCallsRcvdTo5</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeTo5</td>
<td>INTERNALCALLSRCVDTIMETO5</td>
<td>ekwInternalCallsRcvdTimeTo5</td>
</tr>
<tr>
<td>InternalCallsHeldTo5</td>
<td>INTERNALCALLSHELDTO5</td>
<td>ekwInternalCallsHeldTo5</td>
</tr>
<tr>
<td>InternalCallsHeldTimeTo5</td>
<td>INTERNALCALLSHELDTIMETO5</td>
<td>ekwInternalCallsHeldTimeTo5</td>
</tr>
<tr>
<td>CallsQ5</td>
<td>CALLSQ5</td>
<td>ekwCallsQ5</td>
</tr>
<tr>
<td>CallsQTime5</td>
<td>CALLSQTIME5</td>
<td>ekwCallsQTime5</td>
</tr>
<tr>
<td>LongestCallQ5</td>
<td>LONGESTCALLQ5</td>
<td>ekwLongestCallQ5</td>
</tr>
<tr>
<td>SupervisoryAction</td>
<td>SUPERVISORY ACTION</td>
<td>ekwSupervisory Action</td>
</tr>
<tr>
<td>TeamID</td>
<td>TEAMID</td>
<td>ekwTeamID</td>
</tr>
<tr>
<td>NumberOfAgents</td>
<td>NUMBEROFAGENTS</td>
<td>ekwNumberOfAgents</td>
</tr>
<tr>
<td>ConfigOperation</td>
<td>CONFIGOPERATION</td>
<td>ekwConfigOperation</td>
</tr>
<tr>
<td>Direction</td>
<td>DIRECTION</td>
<td>ekwDirection</td>
</tr>
<tr>
<td>RTPType</td>
<td>RTPTYPE</td>
<td>ekwRTPType</td>
</tr>
</tbody>
</table>
### Table B-1  CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitRate</td>
<td>BITRATE</td>
<td>ekwBitRate</td>
</tr>
<tr>
<td>EchoCancellation</td>
<td>ECHOCANCELLATION</td>
<td>ekwEchoCancellation</td>
</tr>
<tr>
<td>PacketSize</td>
<td>PACKETSIZE</td>
<td>ekwPacketSize</td>
</tr>
<tr>
<td>PayloadType</td>
<td>PAYLOADTYPE</td>
<td>ekwPayloadType</td>
</tr>
<tr>
<td>AgentConnectionCallID</td>
<td>AGENTCONNECTIONCALLID</td>
<td>ekwAgentConnectionCallID</td>
</tr>
<tr>
<td>AgentConnectionDeviceID Type</td>
<td>AGENTCONNECTIONDEVICEID TYPE</td>
<td>ekwAgentConnectionDeviceID Type</td>
</tr>
<tr>
<td>SupervisorConnectionCallID</td>
<td>SUPERVISORCONNECTIONCALLID</td>
<td>ekwSupervisorConnectionCallID</td>
</tr>
<tr>
<td>SupervisorConnectionDeviceID Type</td>
<td>SUPERVISORCONNECTIONDEVICEID TYPE</td>
<td>ekwSupervisorConnectionDeviceID Type</td>
</tr>
<tr>
<td>DeskSettingsMask</td>
<td>DESKSETTINGSMASK</td>
<td>ekwDeskSettingsMask</td>
</tr>
<tr>
<td>WrapupDataIncomingMode</td>
<td>WRAPUPDATAINCOMINGMODE</td>
<td>ekwWrapupDataIncomingMode</td>
</tr>
<tr>
<td>WrapupDataOutgoingMode</td>
<td>WRAPUPDATAOUTGOINGMODE</td>
<td>ekwWrapupDataOutgoingMode</td>
</tr>
<tr>
<td>LogoutNonActivityTime</td>
<td>LOGOUTNONACTIVITYTIME</td>
<td>ekwLogoutNonActivityTime</td>
</tr>
<tr>
<td>QualityRecordingRate</td>
<td>QUALITYRECORDINGRATE</td>
<td>ekwQualityRecordingRate</td>
</tr>
<tr>
<td>RingNoAnswerTime</td>
<td>RINGNOANSWERTIME</td>
<td>ekwRingNoAnswerTime</td>
</tr>
<tr>
<td>WorkModeTimer</td>
<td>WORKMODETIMER</td>
<td>ekwWorkModeTimer</td>
</tr>
<tr>
<td>RingNoAnswerDN</td>
<td>RINGNOANSWERDN</td>
<td>ekwRingNoAnswerDN</td>
</tr>
<tr>
<td>NumberOfAgentTeams</td>
<td>NUMBEROFAGENTTEAMS</td>
<td>ekwNumberOfAgentTeams</td>
</tr>
<tr>
<td>More</td>
<td>MORE</td>
<td>ekwMore</td>
</tr>
<tr>
<td>LatSupervisorID</td>
<td>LATSUPERVISORID</td>
<td>ekwLatSupervisorID</td>
</tr>
<tr>
<td>SilentMonitorWarningMessage</td>
<td>SILENTMONITORWARNINGMESSAGE</td>
<td>ekwSilentMonitorWarningMessage</td>
</tr>
<tr>
<td>SilentMonitorAudibleIndication</td>
<td>SILENTMONITORAUDIBLEINDICATION</td>
<td>ekwSilentMonitorAudibleIndication</td>
</tr>
<tr>
<td>SupervisorAssistCallMethod</td>
<td>SUPERVISORASSISTCALLMETHOD</td>
<td>ekwSupervisorAssistCallMethod</td>
</tr>
<tr>
<td>EmergencyCallMethod</td>
<td>EMERGENCYCALLMETHOD</td>
<td>ekwEmergencyCallMethod</td>
</tr>
<tr>
<td>AutoRecordOnEmergency</td>
<td>AUTORECORDONEMERGENCY</td>
<td>ekwAutoRecordOnEmergency</td>
</tr>
<tr>
<td>RecordingMode</td>
<td>RECORDINGMODE</td>
<td>ekwRecordingMode</td>
</tr>
<tr>
<td>SegmentNumber</td>
<td>SEGMENTNUMBER</td>
<td>ekwSegmentNumber</td>
</tr>
</tbody>
</table>
### Table B-1  CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoOutCallsSession</td>
<td>AUTOOUTCALLSSESSION</td>
<td>ekwAutoOutCallsSession</td>
</tr>
<tr>
<td>AutoOutCallsTalkTimeSession</td>
<td>AUTOOUTCALLSTALKTIMESSESSION</td>
<td>ekwAutoOutCallsTalkTimeSession</td>
</tr>
<tr>
<td>AutoOutCallsTimeSession</td>
<td>AUTOOUTCALLSTIMESESSION</td>
<td>ekwAutoOutCallsTimeSession</td>
</tr>
<tr>
<td>AutoOutCallsHeldSession</td>
<td>AUTOOUTCALLSHELDSESSION</td>
<td>ekwAutoOutCallsHeldSession</td>
</tr>
<tr>
<td>AutoOutCallsHeldTimeSession</td>
<td>AUTOOUTCALLSHELDTIMESESSION</td>
<td>ekwAutoOutCallsHeldTimeSession</td>
</tr>
<tr>
<td>AutoOutCallsToday</td>
<td>AUTOOUTCALLSTODAY</td>
<td>ekwAutoOutCallsToday</td>
</tr>
<tr>
<td>AutoOutCallsTalkTimeToday</td>
<td>AUTOOUTCALLSTALKTIMETODAY</td>
<td>ekwAutoOutCallsTalkTimeToday</td>
</tr>
<tr>
<td>AutoOutCallsTimeToday</td>
<td>AUTOOUTCALLSTIMETODAY</td>
<td>ekwAutoOutCallsTimeToday</td>
</tr>
<tr>
<td>AutoOutCallsHeldToday</td>
<td>AUTOOUTCALLSHELDTODAY</td>
<td>ekwAutoOutCallsHeldToday</td>
</tr>
<tr>
<td>AutoOutCallsHeldTimeToday</td>
<td>AUTOOUTCALLSHELDTIMETODAY</td>
<td>ekwAutoOutCallsHeldTimeToday</td>
</tr>
<tr>
<td>AutoOutCallsToHalf</td>
<td>AUTOOUTCALLSTOHALF</td>
<td>ekwAutoOutCallsToHalf</td>
</tr>
<tr>
<td>AutoOutCallsTalkTimeToHalf</td>
<td>AUTOOUTCALLSTALKTIMETOHALF</td>
<td>ekwAutoOutCallsTalkTimeToHalf</td>
</tr>
<tr>
<td>AutoOutCallsTimeToHalf</td>
<td>AUTOOUTCALLSTIMETOHALF</td>
<td>ekwAutoOutCallsTimeToHalf</td>
</tr>
<tr>
<td>AutoOutCallsHeldToHalf</td>
<td>AUTOOUTCALLSHELDTOHALF</td>
<td>ekwAutoOutCallsHeldToHalf</td>
</tr>
<tr>
<td>AutoOutCallsHeldTimeToHalf</td>
<td>AUTOOUTCALLSHELDTIMETOHALF</td>
<td>ekwAutoOutCallsHeldTimeToHalf</td>
</tr>
<tr>
<td>AutoOutCallsTo5</td>
<td>AUTOOUTCALLSTO5</td>
<td>ekwAutoOutCallsTo5</td>
</tr>
<tr>
<td>AutoOutCallsTalkTimeTo5</td>
<td>AUTOOUTCALLSTALKTIMETO5</td>
<td>ekwAutoOutCallsTalkTimeTo5</td>
</tr>
<tr>
<td>AutoOutCallsTimeTo5</td>
<td>AUTOOUTCALLSTIMETO5</td>
<td>ekwAutoOutCallsTimeTo5</td>
</tr>
<tr>
<td>AutoOutCallsHeldTo5</td>
<td>AUTOOUTCALLSHELDTO5</td>
<td>ekwAutoOutCallsHeldTo5</td>
</tr>
<tr>
<td>AutoOutCallsHeldTimeTo5</td>
<td>AUTOOUTCALLSHELDTIMETO5</td>
<td>ekwAutoOutCallsHeldTimeTo5</td>
</tr>
<tr>
<td>PreviewCallsSession</td>
<td>PREVIEWCALLSSESSION</td>
<td>ekwPreviewCallsSession</td>
</tr>
<tr>
<td>PreviewCallsTalkTimeSession</td>
<td>PREVIEWCALLSTALKTIMESSESSION</td>
<td>ekwPreviewCallsTalkTimeSession</td>
</tr>
<tr>
<td>PreviewCallsTimeSession</td>
<td>PREVIEWCALLSTIMESESSION</td>
<td>ekwPreviewCallsTimeSession</td>
</tr>
<tr>
<td>PreviewCallsHeldSession</td>
<td>PREVIEWCALLSHELDSESSION</td>
<td>ekwPreviewCallsHeldSession</td>
</tr>
<tr>
<td>PreviewCallsHeldTimeSession</td>
<td>PREVIEWCALLSHELDTIMESSESSION</td>
<td>ekwPreviewCallsHeldTimeSession</td>
</tr>
<tr>
<td>PreviewCallsToday</td>
<td>PREVIEWCALLSTODAY</td>
<td>ekwPreviewCallsToday</td>
</tr>
<tr>
<td>PreviewCallsTalkTimeToday</td>
<td>PREVIEWCALLSTALKTIMETODAY</td>
<td>ekwPreviewCallsTalkTimeToday</td>
</tr>
<tr>
<td>PreviewCallsTimeToday</td>
<td>PREVIEWCALLSTIMETODAY</td>
<td>ekwPreviewCallsTimeToday</td>
</tr>
<tr>
<td>PreviewCallsHeldToday</td>
<td>PREVIEWCALLSHELDTODAY</td>
<td>ekwPreviewCallsHeldToday</td>
</tr>
<tr>
<td>Text String</td>
<td>Constant String</td>
<td>Enumerated Integer Value</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>PreviewCallsHeldTimeToday</td>
<td>PREVIEWCALLSHELDTIMETODAY</td>
<td>ekwPreviewCallsHeldTimeToday</td>
</tr>
<tr>
<td>PreviewCallsToHalf</td>
<td>PREVIEWCALLSTOHALF</td>
<td>ekwPreviewCallsToHalf</td>
</tr>
<tr>
<td>PreviewCallsTalkTimeToHalf</td>
<td>PREVIEW CALLSTALKTIMETOHALF</td>
<td>ekwPreviewCallsTalkTimeToHalf</td>
</tr>
<tr>
<td>PreviewCallsTimeToHalf</td>
<td>PREVIEWCALLSTIMETOHALF</td>
<td>ekwPreviewCallsTimeToHalf</td>
</tr>
<tr>
<td>PreviewCallsHeldToHalf</td>
<td>PREVIEWCALLSHELDTOHALF</td>
<td>ekwPreviewCallsHeldToHalf</td>
</tr>
<tr>
<td>PreviewCallsHeldTimeToHalf</td>
<td>PREVIEWCALLSHELDTIMETOHALF</td>
<td>ekwPreviewCallsHeldTimeToHalf</td>
</tr>
<tr>
<td>PreviewCallsTo5</td>
<td>PREVIEWCALLSTO5</td>
<td>ekwPreviewCallsTo</td>
</tr>
<tr>
<td>PreviewCallsTalkTimeTo5</td>
<td>PREVIEWCALLSTALKTIMETO5</td>
<td>ekwPreviewCallsTalkTimeTo5</td>
</tr>
<tr>
<td>PreviewCallsTimeTo5</td>
<td>PREVIEWCALLSTIMETO5</td>
<td>ekwPreviewCallsTimeTo5</td>
</tr>
<tr>
<td>PreviewCallsHeldTo5</td>
<td>PREVIEWCALLSHELDTO5</td>
<td>ekwPreviewCallsHeldTo</td>
</tr>
<tr>
<td>PreviewCallsHeldTimeTo5</td>
<td>PREVIEWCALLSHELDTIMETO5</td>
<td>ekwPreviewCallsHeldTimeTo5</td>
</tr>
<tr>
<td>ReserveCallsSession</td>
<td>RESERVECALLSSESSION</td>
<td>ekwReserveCallsSession</td>
</tr>
<tr>
<td>ReserveCallsTalkTimeSession</td>
<td>RESERVECALLSTALKTIMESSESSION</td>
<td>ekwReserveCallsTalkTimeSession</td>
</tr>
<tr>
<td>ReserveCallsTimeSession</td>
<td>RESERVECALLSTIMESSESSION</td>
<td>ekwReserveCallsTimeSession</td>
</tr>
<tr>
<td>ReserveCallsHeldSession</td>
<td>RESERVECALLSHELDSESSION</td>
<td>ekwReserveCallsHeldSession</td>
</tr>
<tr>
<td>ReserveCallsHeldTimeSession</td>
<td>RESERVECALLSHELDTIMESSESSION</td>
<td>ekwReserveCallsHeldTimeSession</td>
</tr>
<tr>
<td>ReserveCallsToday</td>
<td>RESERVECALLSTODAY</td>
<td>ekwReserveCallsToday</td>
</tr>
<tr>
<td>ReserveCallsTalkTimeToday</td>
<td>RESERVECALLSTALKTIMETODAY</td>
<td>ekwReserveCallsTalkTimeToday</td>
</tr>
<tr>
<td>ReserveCallsTimeToday</td>
<td>RESERVECALLSTIMETODAY</td>
<td>ekwReserveCallsTimeToday</td>
</tr>
<tr>
<td>ReserveCallsHeldToday</td>
<td>RESERVECALLSHELDTODAY</td>
<td>ekwReserveCallsHeldToday</td>
</tr>
<tr>
<td>ReserveCallsHeldTimeToday</td>
<td>RESERVECALLSHELDTIMETODAY</td>
<td>ekwReserveCallsHeldTimeToday</td>
</tr>
<tr>
<td>ReserveCallsToHalf</td>
<td>RESERVECALLSTOHALF</td>
<td>ekwReserveCallsToHalf</td>
</tr>
<tr>
<td>ReserveCallsTalkTimeToHalf</td>
<td>RESERVECALLSTALKTIMETOHALF</td>
<td>ekwReserveCallsTalkTimeToHalf</td>
</tr>
<tr>
<td>ReserveCallsTimeToHalf</td>
<td>RESERVECALLSTIMETOHALF</td>
<td>ekwReserveCallsTimeToHalf</td>
</tr>
<tr>
<td>ReserveCallsHeldToHalf</td>
<td>RESERVECALLSHELDTOHALF</td>
<td>ekwReserveCallsHeldToHalf</td>
</tr>
<tr>
<td>ReserveCallsHeldTimeToHalf</td>
<td>RESERVECALLSHELDTIMETOHALF</td>
<td>ekwReserveCallsHeldTimeToHalf</td>
</tr>
<tr>
<td>ReserveCallsTo5</td>
<td>RESERVECALLSTO5</td>
<td>ekwReserveCallsTo</td>
</tr>
<tr>
<td>ReserveCallsTalkTimeTo5</td>
<td>RESERVECALLSTALKTIMETO5</td>
<td>ekwReserve CallsTalkTimeTo5</td>
</tr>
<tr>
<td>ReserveCallsTimeTo5</td>
<td>RESERVECALLSTIMETO5</td>
<td>ekwReserveCallsTimeTo5</td>
</tr>
</tbody>
</table>
### Table B-1 CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReserveCallsHeldTo5</td>
<td>RESERVECALLSHELDTO5</td>
<td>ekwReserveCallsHeldTo5</td>
</tr>
<tr>
<td>ReserveCallsHeldTimeTo5</td>
<td>RESERVECALLSHELDTIMETO5</td>
<td>ekwReserveCallsHeldTimeTo5</td>
</tr>
<tr>
<td>BargeInCallsSession</td>
<td>BARGEINCALLSSESSION</td>
<td>ekwBargeInCallsSession</td>
</tr>
<tr>
<td>BargeInCallsToday</td>
<td>BARGEINCALLSTODAY</td>
<td>ekwBargeInCallsToday</td>
</tr>
<tr>
<td>BargeInCallsToHalf</td>
<td>BARGEINCALLSTOHALF</td>
<td>ekwBargeInCallsToHalf</td>
</tr>
<tr>
<td>BargeInCallsTo5</td>
<td>BARGEINCALLSTO5</td>
<td>ekwBargeInCallsTo5</td>
</tr>
<tr>
<td>InterceptCallsSession</td>
<td>INTERCEPTCALLSSESSION</td>
<td>ekwInterceptCallsSession</td>
</tr>
<tr>
<td>InterceptCallsToday</td>
<td>INTERCEPTCALLSTODAY</td>
<td>ekwInterceptCallsToday</td>
</tr>
<tr>
<td>InterceptCallsToHalf</td>
<td>INTERCEPTCALLSTOHALF</td>
<td>ekwInterceptCallsToHalf</td>
</tr>
<tr>
<td>InterceptCallsTo5</td>
<td>INTERCEPTCALLSTO5</td>
<td>ekwInterceptCallsTo5</td>
</tr>
<tr>
<td>MonitorCallsSession</td>
<td>MONITORCALLSSESSION</td>
<td>ekwMonitorCallsSession</td>
</tr>
<tr>
<td>MonitorCallsToday</td>
<td>MONITORCALLSTODAY</td>
<td>ekwMonitorCallsToday</td>
</tr>
<tr>
<td>MonitorCallsToHalf</td>
<td>MONITORCALLSTOHALF</td>
<td>ekwMonitorCallsToHalf</td>
</tr>
<tr>
<td>MonitorCallsTo5</td>
<td>MONITORCALLSTO5</td>
<td>ekwMonitorCallsTo5</td>
</tr>
<tr>
<td>WhisperCallsSession</td>
<td>WHISPERCALLSSESSION</td>
<td>ekwWhisperCallsSession</td>
</tr>
<tr>
<td>WhisperCallsToday</td>
<td>WHISPERCALLSTODAY</td>
<td>ekwWhisperCallsToday</td>
</tr>
<tr>
<td>WhisperCallsToHalf</td>
<td>WHISPERCALLSTOHALF</td>
<td>ekwWhisperCallsToHalf</td>
</tr>
<tr>
<td>WhisperCallsTo5</td>
<td>WHISPERCALLSTO5</td>
<td>ekwWhisperCallsTo5</td>
</tr>
<tr>
<td>EmergencyCallsSession</td>
<td>EMERGENCYCALLSSESSION</td>
<td>ekwEmergencyCallsSession</td>
</tr>
<tr>
<td>EmergencyCallsToday</td>
<td>EMERGENCYCALLSTODAY</td>
<td>ekwEmergencyCallsToday</td>
</tr>
<tr>
<td>EmergencyCallsToHalf</td>
<td>EMERGENCYCALLSTOHALF</td>
<td>ekwEmergencyCallsToHalf</td>
</tr>
<tr>
<td>EmergencyCallsTo5</td>
<td>EMERGENCYCALLSTO5</td>
<td>ekwEmergencyCallsTo5</td>
</tr>
<tr>
<td>AgentsTalkingAutoOut</td>
<td>AGENTSTALKINGAUTOOUT</td>
<td>ekwAgentsTalkingAutoOut</td>
</tr>
<tr>
<td>AgentsTalkingPreview</td>
<td>AGENTSTALKINGPREVIEW</td>
<td>ekwAgentsTalkingPreview</td>
</tr>
<tr>
<td>AgentsTalkingReservation</td>
<td>AGENTSTALKINGRESERVATION</td>
<td>ekwAgentsTalkingReservation</td>
</tr>
<tr>
<td>RouterCallsQNow</td>
<td>ROUTERCALLSQNOW</td>
<td>ekwRouterCallsQNow</td>
</tr>
<tr>
<td>ServerMode</td>
<td>SERVERMODE</td>
<td>ekwServerMode</td>
</tr>
<tr>
<td>RegisteredServiceID</td>
<td>REGISTEREDSERVICEID</td>
<td>ekwRegisteredServiceID</td>
</tr>
<tr>
<td>ServerData</td>
<td>SERVERDATA</td>
<td>ekwServerData</td>
</tr>
</tbody>
</table>
### Table B-1  CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LongestRouterCallQNow</td>
<td>LONGESTROUTERCALLQNOW</td>
<td>ekwLongestRouterCallQNow</td>
</tr>
<tr>
<td>CampaignID</td>
<td>CAMPAIGNID</td>
<td>ekwCampaignID</td>
</tr>
<tr>
<td>QueryRuleID</td>
<td>QUERYRULEID</td>
<td>ekwQueryRuleID</td>
</tr>
<tr>
<td>TaskID</td>
<td>TASKID</td>
<td>ekwTaskID</td>
</tr>
<tr>
<td>MRDID</td>
<td>MRDID</td>
<td>ekwMRDID</td>
</tr>
<tr>
<td>ICMAgentID</td>
<td>ICMAGENTID</td>
<td>ekwICMAgentID</td>
</tr>
<tr>
<td>AgentMode</td>
<td>AGENTMODE</td>
<td>ekwAgentMode</td>
</tr>
<tr>
<td>IsAvailable</td>
<td>ISAVAILABLE</td>
<td>ekwIsAvailable</td>
</tr>
<tr>
<td>MaxTaskLimit</td>
<td>MAXTASKLIMIT</td>
<td>ekwMaxTaskLimit</td>
</tr>
<tr>
<td>MaxTasks</td>
<td>MAXTASKS</td>
<td>ekwMaxTasks</td>
</tr>
<tr>
<td>MakeRoutable</td>
<td>MAKEROOUTABLE</td>
<td>ekwMakeRoutable</td>
</tr>
<tr>
<td>Reason</td>
<td>REASON</td>
<td>ekwReason</td>
</tr>
<tr>
<td>PreviousICMTaskID</td>
<td>PREVIOUSICMTASKID</td>
<td>ekwPreviousICMTaskID</td>
</tr>
<tr>
<td>ICMDisposition</td>
<td>ICMDISPOSITION</td>
<td>ekwICMDisposition</td>
</tr>
<tr>
<td>ApplicationDisposition</td>
<td>APPLICATIONDISPOSITION</td>
<td>ekwApplicationDisposition</td>
</tr>
<tr>
<td>DoThisWithTaskOnTheWay</td>
<td>DOTHISWITHTASKONTHEWAY</td>
<td>ekwDoThisWithTaskOnTheWay</td>
</tr>
<tr>
<td>IsInterrupted</td>
<td>ISINTERRUPTED</td>
<td>ekwIsInterrupted</td>
</tr>
<tr>
<td>InterruptingMRDID</td>
<td>INTERRUPTINGMRDID</td>
<td>ekwInterruptingMRDID</td>
</tr>
<tr>
<td>InterruptingInvokeID</td>
<td>INTERRUPTINGINVOKEID</td>
<td>ekwInterruptingInvokeID</td>
</tr>
<tr>
<td>IsNotReady</td>
<td>ISNOTREADY</td>
<td>ekwIsNotReady</td>
</tr>
<tr>
<td>AvailableDuration</td>
<td>AVAILABLEDURATION</td>
<td>ekwAvailableDuration</td>
</tr>
<tr>
<td>NumTasks</td>
<td>NUMTASKS</td>
<td>ekwNumTasks</td>
</tr>
<tr>
<td>Paused</td>
<td>PAUSED</td>
<td>ekwPaused</td>
</tr>
<tr>
<td>TaskState</td>
<td>TASKSTATE</td>
<td>ekwTaskState</td>
</tr>
<tr>
<td>OfferDuration</td>
<td>OFFERDURATION</td>
<td>ekwOfferDuration</td>
</tr>
<tr>
<td>BeginDuration</td>
<td>BEGINDURATION</td>
<td>ekwBeginDuration</td>
</tr>
<tr>
<td>InterruptDuration</td>
<td>INTERRUPTDURATION</td>
<td>ekwInterruptDuration</td>
</tr>
<tr>
<td>WrapupDuration</td>
<td>WRAPUPDURATION</td>
<td>ekwWrapupDuration</td>
</tr>
<tr>
<td>ReadyCount</td>
<td>READYCOUNT</td>
<td>ekwReadyCount</td>
</tr>
</tbody>
</table>
### CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>TYPE</td>
<td>ekwType</td>
</tr>
<tr>
<td>ClientID</td>
<td>CLIENTID</td>
<td>ekwClientID</td>
</tr>
<tr>
<td>ClientPassword</td>
<td>CLIENTPASSWORD</td>
<td>ekwClientPassword</td>
</tr>
<tr>
<td>ClientSignature</td>
<td>CLIENTSIGNATURE</td>
<td>ekwClientSignature</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>AGENTEXTENSION</td>
<td>ekwAgentExtension</td>
</tr>
<tr>
<td>AgentID</td>
<td>AGENTID</td>
<td>ekwAgentID</td>
</tr>
<tr>
<td>AgentLastName</td>
<td>AGENTLASTNAME</td>
<td>ekwAgentLastName</td>
</tr>
<tr>
<td>AgentFirstName</td>
<td>AGENTFIRSTNAME</td>
<td>ekwAgentFirstName</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>AGENTINSTRUMENT</td>
<td>ekwAgentInstrument</td>
</tr>
<tr>
<td>ANI</td>
<td>ANI</td>
<td>ekwANI</td>
</tr>
<tr>
<td>UserToUserInfo</td>
<td>USERTOUSERINFO</td>
<td>ekwUserToUserInfo</td>
</tr>
<tr>
<td>DNIS</td>
<td>DNIS</td>
<td>ekwDNIS</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>DIALEDNUMBER</td>
<td>ekwDialedNumber</td>
</tr>
<tr>
<td>CallerEnteredDigits</td>
<td>CALLERENTEREDDIGITS</td>
<td>ekwCallerEnteredDigits</td>
</tr>
<tr>
<td>CallVariable1</td>
<td>CALLVARIABLE1</td>
<td>ekwCallVariable1</td>
</tr>
<tr>
<td>CallVariable2</td>
<td>CALLVARIABLE2</td>
<td>ekwCallVariable2</td>
</tr>
<tr>
<td>CallVariable3</td>
<td>CALLVARIABLE3</td>
<td>ekwCallVariable3</td>
</tr>
<tr>
<td>CallVariable4</td>
<td>CALLVARIABLE4</td>
<td>ekwCallVariable4</td>
</tr>
<tr>
<td>CallVariable5</td>
<td>CALLVARIABLE5</td>
<td>ekwCallVariable5</td>
</tr>
<tr>
<td>CallVariable6</td>
<td>CALLVARIABLE6</td>
<td>ekwCallVariable6</td>
</tr>
<tr>
<td>CallVariable7</td>
<td>CALLVARIABLE7</td>
<td>ekwCallVariable7</td>
</tr>
<tr>
<td>CallVariable8</td>
<td>CALLVARIABLE8</td>
<td>ekwCallVariable8</td>
</tr>
<tr>
<td>CallVariable9</td>
<td>CALLVARIABLE9</td>
<td>ekwCallVariable9</td>
</tr>
<tr>
<td>CallVariable10</td>
<td>CALLVARIABLE10</td>
<td>ekwCallVariable10</td>
</tr>
<tr>
<td>CTIClientSignature</td>
<td>CTICLIENTSIGNATURE</td>
<td>ekwCTIClientSignature</td>
</tr>
<tr>
<td>CTIClientTimestamp</td>
<td>CTICLIENTTIMESTAMP</td>
<td>ekwCTIClientTimestamp</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>CONNECTIONDEVICEID</td>
<td>ekwConnectionDeviceID</td>
</tr>
<tr>
<td>AlertingDeviceID</td>
<td>ALERTINGDEVICEID</td>
<td>ekwAlertingDeviceID</td>
</tr>
<tr>
<td>CallingDeviceID</td>
<td>CALLINGDEVICEID</td>
<td>ekwCallingDeviceID</td>
</tr>
<tr>
<td>Text String</td>
<td>Constant String</td>
<td>Enumerated Integer Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>CalledDeviceID</td>
<td>CALLEDDEVICEID</td>
<td>ekwCalledDeviceID</td>
</tr>
<tr>
<td>LastRedirectDeviceID</td>
<td>LASTREDIRECTDEVICEID</td>
<td>ekwLastRedirectDeviceID</td>
</tr>
<tr>
<td>AnsweringDeviceID</td>
<td>ANSWERINGDEVICEID</td>
<td>ekwAnsweringDeviceID</td>
</tr>
<tr>
<td>HoldingDeviceID</td>
<td>HOLDINGDEVICEID</td>
<td>ekwHoldingDeviceID</td>
</tr>
<tr>
<td>RetrievingDeviceID</td>
<td>RETRIEVINGDEVICEID</td>
<td>ekwRetrievingDeviceID</td>
</tr>
<tr>
<td>ReleasingDeviceID</td>
<td>RELEASINGDEVICEID</td>
<td>ekwReleasingDeviceID</td>
</tr>
<tr>
<td>FailingDeviceID</td>
<td>FAILINGDEVICEID</td>
<td>ekwFailingDeviceID</td>
</tr>
<tr>
<td>PrimaryDeviceID</td>
<td>PRIMARYDEVICEID</td>
<td>ekwPrimaryDeviceID</td>
</tr>
<tr>
<td>SecondaryDeviceID</td>
<td>SECONDARYDEVICEID</td>
<td>ekwSecondaryDeviceID</td>
</tr>
<tr>
<td>ControllerDeviceID</td>
<td>CONTROLLERDEVICEID</td>
<td>ekwControllerDeviceID</td>
</tr>
<tr>
<td>AddedPartyDeviceID</td>
<td>ADDEDPARTYDEVICEID</td>
<td>ekwAddedPartyDeviceID</td>
</tr>
<tr>
<td>ConnectedPartyCallID</td>
<td>CONNECTEDPARTYCALLID</td>
<td>ekwConnectedPartyCallID</td>
</tr>
<tr>
<td>ConnectedPartyDeviceIDType</td>
<td>CONNECTEDPARTYDEVICEIDTYPE</td>
<td>ekwConnectedPartyDeviceIDType</td>
</tr>
<tr>
<td>TransferringDeviceID</td>
<td>TRANSFERRINGDEVICEID</td>
<td>ekwTransferringDeviceID</td>
</tr>
<tr>
<td>TransferredDeviceID</td>
<td>TRANSFERREDDEVICEID</td>
<td>ekwTransferredDeviceID</td>
</tr>
<tr>
<td>DivertingDeviceID</td>
<td>DIVERTINGDEVICEID</td>
<td>ekwDivertingDeviceID</td>
</tr>
<tr>
<td>QueueDeviceID</td>
<td>QUEUEDEVICEID</td>
<td>ekwQueueDeviceID</td>
</tr>
<tr>
<td>CallWrapupData</td>
<td>CALLWRAPUPDATA</td>
<td>ekwCallWrapupData</td>
</tr>
<tr>
<td>UserText</td>
<td>USERTEXT</td>
<td>ekwUserText</td>
</tr>
<tr>
<td>NewConnectionDeviceID</td>
<td>NEWCONNECTIONDEVICEID</td>
<td>ekwNewConnectionDeviceID</td>
</tr>
<tr>
<td>ObjectName</td>
<td>OBJECTNAME</td>
<td>ekwObjectName</td>
</tr>
<tr>
<td>TrunkUsedDeviceID</td>
<td>TRUNKUSEDDEVICEID</td>
<td>ekwTrunkUsedDeviceID</td>
</tr>
<tr>
<td>FltSkillGroupNumber</td>
<td>FLTSKILLGROUPNUMBER</td>
<td>ekwFltSkillGroupNumber</td>
</tr>
<tr>
<td>FltSkillGroupId</td>
<td>FLTSKILLGROUPID</td>
<td>ekwFltSkillGroupId</td>
</tr>
<tr>
<td>FltSkillGroupPriority</td>
<td>FLTSKILLGROUPPRIORITY</td>
<td>ekwFltSkillGroupPriority</td>
</tr>
<tr>
<td>FltSkillGroupState</td>
<td>FLTSKILLGROUPSTATE</td>
<td>ekwFltSkillGroupState</td>
</tr>
<tr>
<td>AgentPassword</td>
<td>AGENTPASSWORD</td>
<td>ekwAgentPassword</td>
</tr>
<tr>
<td>PositionID</td>
<td>POSITIONID</td>
<td>ekwPositionID</td>
</tr>
</tbody>
</table>
Table B-1  CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupervisorID</td>
<td>SUPERVISORID</td>
<td>ekwSupervisorID</td>
</tr>
<tr>
<td>ActiveConnDeviceID</td>
<td>ACTIVECONNDEVICEID</td>
<td>ekwActiveConnDeviceID</td>
</tr>
<tr>
<td>OtherConnDeviceID</td>
<td>OTHERCONNDEVICEID</td>
<td>ekwOtherConnDeviceID</td>
</tr>
<tr>
<td>HeldConnDeviceID</td>
<td>HELDCONNDEVICEID</td>
<td>ekwHeldConnDeviceID</td>
</tr>
<tr>
<td>FacilityCode</td>
<td>FACILITYCODE</td>
<td>ekwFacilityCode</td>
</tr>
<tr>
<td>FltLineHandle</td>
<td>FLTLINEHANDLE</td>
<td>ekwFltLineHandle</td>
</tr>
<tr>
<td>FltLineType</td>
<td>FLTLINE&gt;Type</td>
<td>ekwFltLineType</td>
</tr>
<tr>
<td>CallConnectionCallID</td>
<td>CALLCONNECTIONCALLID</td>
<td>ekwCallConnectionCallID</td>
</tr>
<tr>
<td>CallConnectionDeviceIDType</td>
<td>CALLCONNECTIONDEVICEIDTYPE</td>
<td>ekwCallConnectionDeviceIDType</td>
</tr>
<tr>
<td>CallConnectionDeviceID</td>
<td>CALLCONNECTIONDEVICEID</td>
<td>ekwCallConnectionDeviceID</td>
</tr>
<tr>
<td>CallDeviceType</td>
<td>CALLDEVICETYPE</td>
<td>ekwCallDeviceType</td>
</tr>
<tr>
<td>CallDeviceID</td>
<td>CALLDEVICEID</td>
<td>ekwCallDeviceID</td>
</tr>
<tr>
<td>CallDeviceConnState</td>
<td>CALLDEVICECONNSTATE</td>
<td>ekwCallDeviceConnState</td>
</tr>
<tr>
<td>DTMFString</td>
<td>DTMFSTRING</td>
<td>ekwDTMFString</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>ROUTERCALLKEYDAY</td>
<td>ekwRouterCallKeyDay</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>ROUTERCALLKEYCALLID</td>
<td>ekwRouterCallKeyCallID</td>
</tr>
<tr>
<td>DataStructure</td>
<td>DATASTRUCTURE</td>
<td>ekwDataStructure</td>
</tr>
<tr>
<td>CallState</td>
<td>CALLSTATE</td>
<td>ekwCallState</td>
</tr>
<tr>
<td>MonitoredDeviceID</td>
<td>MONITOREDDEVICEID</td>
<td>ekwMonitoredDeviceID</td>
</tr>
<tr>
<td>AccountCode</td>
<td>ACCOUNTCODE</td>
<td>ekwAccountCode</td>
</tr>
<tr>
<td>OriginatingDeviceID</td>
<td>ORIGINATINGDEVICEID</td>
<td>ekwOriginatingDeviceID</td>
</tr>
<tr>
<td>OriginatingLineID</td>
<td>ORIGINATINGLINEID</td>
<td>ekwOriginatingLineID</td>
</tr>
<tr>
<td>ClientAddress</td>
<td>CLIENTADDRESS</td>
<td>ekwClientAddress</td>
</tr>
<tr>
<td>ExpandedCallContxt</td>
<td>EXPANDEDCALLCONTXT</td>
<td>ekwExpandedCallContxt</td>
</tr>
<tr>
<td>CallControlTable</td>
<td>CALLCONTROLTABLE</td>
<td>ekwCallControlTable</td>
</tr>
<tr>
<td>SupervisorInstrument</td>
<td>SUPERVISORINSTRUMENT</td>
<td>ekwSupervisorInstrument</td>
</tr>
<tr>
<td>AtcAgentID</td>
<td>ATCAGENTID</td>
<td>ekwAtcAgentID</td>
</tr>
<tr>
<td>AgentFlags</td>
<td>AGENTFLAGS</td>
<td>ekwAgentFlags</td>
</tr>
<tr>
<td>Text String</td>
<td>Constant String</td>
<td>Enumerated Integer Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>AtcAgentState</td>
<td>ATCAGENTSTATE</td>
<td>ekwAtcAgentState</td>
</tr>
<tr>
<td>AtcStateDuration</td>
<td>ATCSTATEDURATION</td>
<td>ekwAtcStateDuration</td>
</tr>
<tr>
<td>AgentConnectionDeviceID</td>
<td>AGENTCONNECTIONDEVICEID</td>
<td>ekwAgentConnectionDeviceID</td>
</tr>
<tr>
<td>SupervisorConnectionDeviceID</td>
<td>SUPERVISORCONNECTIONDEVICEID</td>
<td>ekwSupervisorConnectionDeviceID</td>
</tr>
<tr>
<td>ListTeamID</td>
<td>LISTTEAMID</td>
<td>ekwListTeamID</td>
</tr>
<tr>
<td>DefaultDevicePortAddress</td>
<td>DEFAULTDEVICEPORTADDRESS</td>
<td>ekwDefaultDevicePortAddress</td>
</tr>
<tr>
<td>ServiceName</td>
<td>SERVICENAME</td>
<td>ekwServiceName</td>
</tr>
<tr>
<td>CustomerPhoneNumber</td>
<td>CUSTOMERPHONENUMBER</td>
<td>ekwCustomerPhoneNumber</td>
</tr>
<tr>
<td>CustomerAccountNumber</td>
<td>CUSTOMERACCOUNTNUMBER</td>
<td>ekwCustomerAccountNumber</td>
</tr>
<tr>
<td>ApplicationPathID</td>
<td>APPLICATIONPATHID</td>
<td>ekwApplicationPathID</td>
</tr>
<tr>
<td>ApplicationData</td>
<td>APPLICATIONDATA</td>
<td>ekwApplicationData</td>
</tr>
<tr>
<td>ScriptSelector</td>
<td>SCRIPTSELECTOR</td>
<td>ekwScriptSelector</td>
</tr>
<tr>
<td>ApplicationString1</td>
<td>APPLICATIONSTRING1</td>
<td>ekwApplicationString1</td>
</tr>
<tr>
<td>ApplicationString2</td>
<td>APPLICATIONSTRING2</td>
<td>ekwApplicationString2</td>
</tr>
<tr>
<td>AgentInfo</td>
<td>AGENTINFO</td>
<td>ekwAgentInfo</td>
</tr>
<tr>
<td>WrapupData</td>
<td>WRAPUPDATA</td>
<td>ekwWrapupData</td>
</tr>
<tr>
<td>RouteSelected</td>
<td>ROUTESELECTED</td>
<td>ekwRouteSelected</td>
</tr>
<tr>
<td>SkillGroups</td>
<td>SKILLGROUPS</td>
<td>ekwSkillGroups</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>ENABLEMENTMASK</td>
<td>ekwEnablementMask</td>
</tr>
<tr>
<td>DeviceID</td>
<td>DEVICEID</td>
<td>ekwDeviceID</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>UNIQUEOBJECTID</td>
<td>ekwUniqueObjectID</td>
</tr>
<tr>
<td>ServerObjectID</td>
<td>SERVEROBJECTID</td>
<td>ekwServerObjectID</td>
</tr>
<tr>
<td>NewUniqueObjectID</td>
<td>NEWUNIQUEOBJECTID</td>
<td>ekwNewUniqueObjectID</td>
</tr>
<tr>
<td>Monitored</td>
<td>MONITORED</td>
<td>ekwMonitored</td>
</tr>
<tr>
<td>CallStatus</td>
<td>CALLSTATUS</td>
<td>ekwCallStatus</td>
</tr>
<tr>
<td>EventID</td>
<td>EVENTID</td>
<td>ekwEventID</td>
</tr>
<tr>
<td>MessageID</td>
<td>MESSAGEID</td>
<td>ekwMessageID</td>
</tr>
<tr>
<td>MessageType</td>
<td>MESSAGETYPE</td>
<td>ekwMessageType</td>
</tr>
</tbody>
</table>
### Table B-1  CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DriverKey</td>
<td>DRIVERKEY</td>
<td>ekwDriverKey</td>
</tr>
<tr>
<td>ecc</td>
<td>ECC</td>
<td>ekwecc</td>
</tr>
<tr>
<td>Indicator</td>
<td>INDICATOR</td>
<td>ekwIndicator</td>
</tr>
<tr>
<td>ForwardType</td>
<td>FORWARDTYPE</td>
<td>ekwForwardType</td>
</tr>
<tr>
<td>ForwardDeviceType</td>
<td>FORWARDDEVICETYPE</td>
<td>ekwForwardDeviceType</td>
</tr>
<tr>
<td>Forwarding</td>
<td>FORWARDING</td>
<td>ekwForwarding</td>
</tr>
<tr>
<td>LastNumberOfType</td>
<td>LASTNUMBERTYPE</td>
<td>ekwLastNumberOfType</td>
</tr>
<tr>
<td>ICMCentralControllerTime</td>
<td>ICMCENTRALCONTROLLERTIME</td>
<td>ekwICMCentralControllerTime</td>
</tr>
<tr>
<td>Statistics</td>
<td>STATISTICS</td>
<td>ekwStatistics</td>
</tr>
<tr>
<td>IncomingOrOutgoing</td>
<td>INCOMINGOROUTGOING</td>
<td>ekwIncomingOrOutgoing</td>
</tr>
<tr>
<td>WrapupOKEnabled</td>
<td>WRAPUPOKENABLED</td>
<td>ekwWrapupOKEnabled</td>
</tr>
<tr>
<td>Instrument</td>
<td>INSTRUMENT</td>
<td>ekwInstrument</td>
</tr>
<tr>
<td>Password</td>
<td>PASSWORD</td>
<td>ekwPassword</td>
</tr>
<tr>
<td>State</td>
<td>STATE</td>
<td>ekwState</td>
</tr>
<tr>
<td>Text</td>
<td>CLIENT_MESSAGE_TEXT</td>
<td>ekwClientMessageText</td>
</tr>
<tr>
<td>BMUDataField</td>
<td>BMUDATAFIELD</td>
<td>ekwBMUDataField</td>
</tr>
<tr>
<td>MakeCallSetup</td>
<td>MAKECALLSETUP</td>
<td>ekwMakeCallSetup</td>
</tr>
<tr>
<td>CallDeviceConnectionState</td>
<td>CALLDEVICECONNECTIONSTATE</td>
<td>ekwCallDeviceConnectionState</td>
</tr>
<tr>
<td>InternalCallsRcvdTod</td>
<td>INTERNALCALLSRCVDTODAY</td>
<td>ekwInternalCallsRcvdTodToday</td>
</tr>
<tr>
<td>InternalCallsRcvdTimeToday</td>
<td>INTERNALCALLSRCVDTimetODAY</td>
<td>ekwInternalCallsRcvdTimeToday</td>
</tr>
<tr>
<td>InternalCallsRcvdSession</td>
<td>INTERNALCALLSRCVDSESSION</td>
<td>ekwInternalCallsRcvdSession</td>
</tr>
<tr>
<td>ReservationCallsSession</td>
<td>RESERVATIONCALLSSESSION</td>
<td>ekwReservationCallsSession</td>
</tr>
<tr>
<td>ReservationCallsTalkTimeSession</td>
<td>RESERVATIONCALLSTALKTIMESESSION</td>
<td>ekwReservationCallsTalkTimeSession</td>
</tr>
<tr>
<td>ReservationCallsTimeSession</td>
<td>RESERVATIONCALLSTIMESESSION</td>
<td>ekwReservationCallsTimeSession</td>
</tr>
<tr>
<td>ReservationCallsHeldSession</td>
<td>RESERVATIONCALLSHELDSESSION</td>
<td>ekwReservationCallsHeldSession</td>
</tr>
<tr>
<td>ReservationCallsHeldTimeSession</td>
<td>RESERVATIONCALLSHELDTIMESESSION</td>
<td>ekwReservationCallsHeldTimeSession</td>
</tr>
</tbody>
</table>
Table B-1  CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationCallsToday</td>
<td>RESERVATIONCALLSTODAY</td>
<td>ekwReservationCallsToday</td>
</tr>
<tr>
<td>ReservationCallsTalkTime</td>
<td>RESERVATIONCALLSTALKTIME</td>
<td>ekwReservationCallsTalkTimeToday</td>
</tr>
<tr>
<td>ReservationCallsHeldTime</td>
<td>RESERVATIONCALLSHELDTIME</td>
<td>ekwReservationCallsHeldTimeToday</td>
</tr>
<tr>
<td>ReservationCallsTimeToday</td>
<td>RESERVATIONCALLSTIMETODAY</td>
<td>ekwReservationCallsTimeToday</td>
</tr>
<tr>
<td>ReservationCallsHeldToday</td>
<td>RESERVATIONCALLSHELDTODAY</td>
<td>ekwReservationCallsHeldToday</td>
</tr>
<tr>
<td>ReservationCallsTo5</td>
<td>RESERVATIONCALLSTO5</td>
<td>ekwReservationCallsTo5</td>
</tr>
<tr>
<td>ReservationCallsTalkTimeTo5</td>
<td>RESERVATIONCALLSTALKTIMETO5</td>
<td>ekwReservationCallsTalkTimeTo5</td>
</tr>
<tr>
<td>ReservationCallsHeldTo5</td>
<td>RESERVATIONCALLSHELDTO5</td>
<td>ekwReservationCallsHeldTo5</td>
</tr>
<tr>
<td>ReservationCallsToHalf</td>
<td>RESERVATIONCALLSTOHALF</td>
<td>ekwReservationCallsToHalf</td>
</tr>
<tr>
<td>ReservationCallsTalkTimeToHalf</td>
<td>RESERVATIONCALLSTALKTIME</td>
<td>ekwReservationCallsTalkTimeToHalf</td>
</tr>
<tr>
<td>ReservationCallsHeldToHalf</td>
<td>RESERVATIONCALLSHELDTOHALF</td>
<td>ekwReservationCallsHeldToHalf</td>
</tr>
<tr>
<td>EventTime</td>
<td>EVENTTIME</td>
<td>ekwEventTime</td>
</tr>
<tr>
<td>CurrentServer</td>
<td>CURRENTSERVER</td>
<td>ekwCurrentServer</td>
</tr>
<tr>
<td>CurrentPort</td>
<td>CURRENTPORT</td>
<td>ekwCurrentPort</td>
</tr>
<tr>
<td>ReasonCode</td>
<td>REASONCODE</td>
<td>ekwReasonCode</td>
</tr>
<tr>
<td>FailedServer</td>
<td>FAILEDSERVER</td>
<td>ekwFailedServer</td>
</tr>
<tr>
<td>HeartBeatInterval</td>
<td>HEARTBEATINTERVAL</td>
<td>ekwHeartBeatInterval</td>
</tr>
<tr>
<td>MissedHeartBeats</td>
<td>MISSEDHEARTBEATS</td>
<td>ekwMissedHeartbeats</td>
</tr>
<tr>
<td>ClientAgentTemporaryID</td>
<td>CLIENTAGENTTEMPORARYID</td>
<td>ekwClientAgentTemporaryID</td>
</tr>
<tr>
<td>ClassIdentifier</td>
<td>CLASSIDENTIFIER</td>
<td>ekwClassIdentifier</td>
</tr>
<tr>
<td>IsSupervisor</td>
<td>ISSUPERVISOR</td>
<td>ekwIsSupervisor</td>
</tr>
<tr>
<td>Extension</td>
<td>EXTENSION</td>
<td>ekwExtension</td>
</tr>
</tbody>
</table>
### Table B-1  CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LastError</td>
<td>LASTERROR</td>
<td>ekwLastError</td>
</tr>
<tr>
<td>CallReference</td>
<td>CALLREFERENCE</td>
<td>ekwCallReference</td>
</tr>
<tr>
<td>AgentReference</td>
<td>AGENTREFERENCE</td>
<td>ekwAgentReference</td>
</tr>
<tr>
<td>AgentCallReference</td>
<td>AGENTCALLREFERENCE</td>
<td>ekwAgentCallReference</td>
</tr>
<tr>
<td>MonitorFlag</td>
<td>MONITORFLAG</td>
<td>ekwMonitorFlag</td>
</tr>
<tr>
<td>CtiOsA</td>
<td>CTIOSA</td>
<td>ekwCtiOsA</td>
</tr>
<tr>
<td>CtiOsB</td>
<td>CTIOSB</td>
<td>ekwCtiOsB</td>
</tr>
<tr>
<td>PortA</td>
<td>PORTA</td>
<td>ekwPortA</td>
</tr>
<tr>
<td>PortB</td>
<td>PORTB</td>
<td>ekwPortB</td>
</tr>
<tr>
<td>Heartbeat</td>
<td>HEARTBEAT</td>
<td>ekwHeartbeat</td>
</tr>
<tr>
<td>MaxHeartbeats</td>
<td>MAXHEARTBEATS</td>
<td>ekwMaxHeartbeats</td>
</tr>
<tr>
<td>CurrentAgent</td>
<td>CURRENTAGENT</td>
<td>ekwCurrentAgent</td>
</tr>
<tr>
<td>ActiveCall</td>
<td>ACTIVECALL</td>
<td>ekwActiveCall</td>
</tr>
<tr>
<td>CurrentFilter</td>
<td>CURRENTFILTER</td>
<td>ekwCurrentFilter</td>
</tr>
<tr>
<td>Agents</td>
<td>AGENTS</td>
<td>ekwAgents</td>
</tr>
<tr>
<td>Calls</td>
<td>CALLS</td>
<td>ekwCalls</td>
</tr>
<tr>
<td>WaitObjects</td>
<td>WAITOBJECTS</td>
<td>ekwWaitObjects</td>
</tr>
<tr>
<td>ObjectReferences</td>
<td>OBJECTREFERENCES</td>
<td>ekwObjectReferences</td>
</tr>
<tr>
<td>DesktopSettings</td>
<td>DESKTOPSETTINGS</td>
<td>ekwDesktopSettings</td>
</tr>
<tr>
<td>CallVariables</td>
<td>CALLVARIABLES</td>
<td>ekwCallVariables</td>
</tr>
<tr>
<td>CallReferenceObjectID</td>
<td>CALLREFERENCEOBJECTID</td>
<td>ekwCallReferenceObjectID</td>
</tr>
<tr>
<td>RecordEnable</td>
<td>RECORDENABLE</td>
<td>ekwRecordEnable</td>
</tr>
<tr>
<td>TimeOfDay</td>
<td>TIMEOFDAY</td>
<td>ekwTimeOfDay</td>
</tr>
<tr>
<td>ConnectionMode</td>
<td>CONNECTIONMODE</td>
<td>ekwConnectionMode</td>
</tr>
<tr>
<td>TryingServer</td>
<td>TRYINGSERVER</td>
<td>ekwTryingServer</td>
</tr>
<tr>
<td>TryingPort</td>
<td>TRYINGPORT</td>
<td>ekwTryingPort</td>
</tr>
<tr>
<td>TryingSince</td>
<td>TRYINGSINCE</td>
<td>ekwTryingSince</td>
</tr>
<tr>
<td>MessageFilter</td>
<td>MESSAGEFILTER</td>
<td>ekwMessageFilter</td>
</tr>
<tr>
<td>DesktopType</td>
<td>DESKTOPTYPE</td>
<td>ekwDesktopType</td>
</tr>
<tr>
<td>Text String</td>
<td>Constant String</td>
<td>Enumerated Integer Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>SupervisorBtnEnablementMask</td>
<td>SUPERVISORBTNENABLEMENT MASK</td>
<td>ekwSupervisorBtnEnablementMask</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>ERRORMESSAGE</td>
<td>ekwErrorMessage</td>
</tr>
<tr>
<td>Mode</td>
<td>MODE</td>
<td>ekwMode</td>
</tr>
<tr>
<td>Call</td>
<td>CALL</td>
<td>ekwCall</td>
</tr>
<tr>
<td>SkillGroupName</td>
<td>SKILLGROUPNAME</td>
<td>ekwSkillGroupName</td>
</tr>
<tr>
<td>OldUniqueObjectID</td>
<td>OLDUNIQUEOBJECTID</td>
<td>ekwOldUniqueObjectID</td>
</tr>
<tr>
<td>EmbeddedArgs</td>
<td>EMBEDDEDARGS</td>
<td>ekwEmbeddedArgs</td>
</tr>
<tr>
<td>EventMask</td>
<td>EVENTMASK</td>
<td>ekwEventMask</td>
</tr>
<tr>
<td>ID</td>
<td>ID</td>
<td>ekwID</td>
</tr>
<tr>
<td>Filter</td>
<td>FILTER</td>
<td>ekwFilter</td>
</tr>
<tr>
<td>CILConnectionID</td>
<td>CILCONNECTIONID</td>
<td>ekwCILConnectionID</td>
</tr>
<tr>
<td>IncomingWrapupStrings</td>
<td>INCOMINGWRAPUPSTRINGS</td>
<td>ekwIncomingWrapupStrings</td>
</tr>
<tr>
<td>OutgoingWrapupStrings</td>
<td>OUTGOINGWRAPUPSTRINGS</td>
<td>ekwOutgoingWrapupStrings</td>
</tr>
<tr>
<td>NotReadyReasonCodes</td>
<td>NOTREADYREASONCODES</td>
<td>ekwNotReadyReasonCodes</td>
</tr>
<tr>
<td>LogoutReasonCodes</td>
<td>LOGOUTREASONCODES</td>
<td>ekwLogoutReasonCodes</td>
</tr>
<tr>
<td>DriverID</td>
<td>DRIVERID</td>
<td>ekwDriverID</td>
</tr>
<tr>
<td>SkillGroupNumbers</td>
<td>SKILLGROUPNUMBERS</td>
<td>ekwSkillGroupNumbers</td>
</tr>
<tr>
<td>Target</td>
<td>TARGET</td>
<td>ekwTarget</td>
</tr>
<tr>
<td>Source</td>
<td>SOURCE</td>
<td>ekwSource</td>
</tr>
<tr>
<td>SupervisorKey</td>
<td>SUPERVISORKEY</td>
<td>ekwSupervisorKey</td>
</tr>
<tr>
<td>BargedInCallID</td>
<td>BARGEDINCALLID</td>
<td>ekwBargedInCallID</td>
</tr>
<tr>
<td>TeamUniqueID</td>
<td>TEAMUNIQUEID</td>
<td>ekwTeamUniqueID</td>
</tr>
<tr>
<td>ValidCalls</td>
<td>VALIDCALLS</td>
<td>ekwValidCalls</td>
</tr>
<tr>
<td>StatusBarMessage</td>
<td>STATUSBARMESSAGE</td>
<td>ekwStatusBarMessage</td>
</tr>
<tr>
<td>FailoverRequired</td>
<td>FAILOVERREQUIRED</td>
<td>ekwFailoverRequired</td>
</tr>
<tr>
<td>CTIOSSystemEventID</td>
<td>CTIOSSYSTEMEVENTID</td>
<td>ekwCTIOSSystemEventID</td>
</tr>
<tr>
<td>AutoLogin</td>
<td>AUTOLOGIN</td>
<td>ekwAutoLogin</td>
</tr>
<tr>
<td>SavedAgentState</td>
<td>SAVEDAGENTSTATE</td>
<td>ekwSavedAgentState</td>
</tr>
</tbody>
</table>
### Table B-1  CTI OS Keywords (continued)

<table>
<thead>
<tr>
<th>Text String</th>
<th>Constant String</th>
<th>Enumerated Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WaitingforRecovery</td>
<td>WAITINGFORRECOVERY</td>
<td>ekwWaitingForRecovery</td>
</tr>
<tr>
<td>SavedLoginInfo</td>
<td>SAVEDLOGININFO</td>
<td>ekwSavedLoginInfo</td>
</tr>
<tr>
<td>EventType</td>
<td>EVENTTYPE</td>
<td>ekwEventType</td>
</tr>
<tr>
<td>RequestID</td>
<td>REQUESTID</td>
<td>ekwRequestID</td>
</tr>
<tr>
<td>NonClientRequest</td>
<td>NONCLIENTREQUEST</td>
<td>ekwNonClientRequest</td>
</tr>
<tr>
<td>TimerInterval</td>
<td>TIMERINTERVAL</td>
<td>ekwTimerInterval</td>
</tr>
<tr>
<td>TimerCount</td>
<td>TIMERCOUNT</td>
<td>ekwTimerCount</td>
</tr>
<tr>
<td>CTIClients</td>
<td>CTICLIENTS</td>
<td>ekwCTIClients</td>
</tr>
</tbody>
</table>
INDEX

A

accessing call variables 1-12
accessing ECC variables 1-12
accessing elements 1-9
accessing properties 1-7
AddItem
  Arguments class method 8-20
agent object methods 5-1
  AgentTeamList 5-3
  BadCallLine 5-5
  DisableAgentStatistics 5-8
  Emergency 5-6
  Enable Agent Statistics 5-8
  GetAgentState 5-10
  GetElement 5-12
  GetMonitoredAgent 5-12
  GetMonitoredCall 5-14
  GetPropertyAttribute 5-16
  GetSkillGroups 5-21
  GetValue 5-21
  Login 5-22
  Logout 5-24
  MakeCall 5-26
  MonitorAgentTeamAll 5-30, 5-31
OnEvent 5-33
  SendChatMessage 5-34
  SendUserMessage 5-34
  SetAgentState 5-36
  SetMonitoredAgent 5-38
  SetMonitoredCall 5-40
  SetValue 5-41
  SuperviseCall 5-42
  Supervisor Assist 5-44
agent object methods 5-1
  AgentTeamList agent object method 5-3
Alternate call object method 6-3
Answer call object method 6-5
Arg class 8-2
Arg class methods
  Clone 8-12
  CreateInstance 8-3
  GetClassID 8-17
  GetType 8-14
  GetValue 8-17
  SetClassID 8-14
  SetPropertyAttribute 5-16
arguments class methods
  AddItem 8-20
  Clear 8-21
  Clone 8-21
  DumpArgs 8-22
Index

GetClassID 8-23
GetElement 8-23
GetValue 8-23
Initialize 8-24
IsValid 8-24
NumElements 8-24
RemoveItem 8-25
SetElement 8-25
SetValue 8-25

OnEvent 6-22
Reconnect 6-23
Record 6-25
Retrieve 6-27
SendDTMFSignal 6-29
SetCallData 6-31
SingleStepConference 6-34
SingleStepTransfer 6-36
Snapshot 6-38
Transfer 6-40
call variables
accessing 1-12
CCtiosException class 8-29
CILRefArg class 8-26
CILRefArg class methods
GetClassID 8-28
GetType 8-29
GetValue 8-27
SetValue 8-26
Clear
Arguments class method 8-21
Clear call object method 6-6
ClearConnection call object method 6-8
Clone
Arg class method 8-12
Arguments class method 8-21
COM 3-6
COM event handling 3-6
Conference call object method 6-10

B
BadCallLine agent object method 5-5

call variables
accessing 1-12

C
C++ event handling 3-2
call object methods 6-2
Alternate 6-3
Answer 6-5
Clear 6-6
ClearConnection 6-8
Conference 6-10
Deflect 6-12
GetElement 6-14
GetPropertyAttribute 6-14
GetValue 6-16
Hold 6-17
MakeConsultCall 6-19
<table>
<thead>
<tr>
<th>Connect session object method</th>
<th>4-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateInstance</td>
<td></td>
</tr>
<tr>
<td>Arg class method</td>
<td>8-3</td>
</tr>
<tr>
<td>Arguments class method</td>
<td>8-22</td>
</tr>
<tr>
<td>CTI OS</td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td>2-1</td>
</tr>
<tr>
<td>CTI OS Client Interface Library (CIL)</td>
<td>2-1</td>
</tr>
<tr>
<td>CTI OS Keywords</td>
<td>B-2</td>
</tr>
<tr>
<td>CTI OS messages</td>
<td>A-1</td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>Deflect call object method</td>
<td>6-12</td>
</tr>
<tr>
<td>DisableAgentStatistics agent object method</td>
<td>5-8</td>
</tr>
<tr>
<td>DisableSkillGroupStatistics SkillGroup object method</td>
<td>7-2</td>
</tr>
<tr>
<td>Disconnect session object method</td>
<td>4-8</td>
</tr>
<tr>
<td>DumpArgs</td>
<td></td>
</tr>
<tr>
<td>Arguments class method</td>
<td>8-22</td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>ECC variables</td>
<td>1-12</td>
</tr>
<tr>
<td>Emergency agent object method</td>
<td>5-6</td>
</tr>
<tr>
<td>Enable Agent Statistics agent object method</td>
<td>5-8</td>
</tr>
<tr>
<td>EnableSkillGroupStatistics SkillGroup object method</td>
<td>7-2</td>
</tr>
<tr>
<td>enumerated constants</td>
<td>1-9</td>
</tr>
<tr>
<td>error handling</td>
<td>1-13</td>
</tr>
<tr>
<td>event handling</td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td>3-2</td>
</tr>
<tr>
<td>COM</td>
<td>3-6</td>
</tr>
<tr>
<td>G</td>
<td></td>
</tr>
<tr>
<td>GetAgentState agent object method</td>
<td>5-10</td>
</tr>
<tr>
<td>GetClassID</td>
<td></td>
</tr>
<tr>
<td>Arg class method</td>
<td>8-17</td>
</tr>
<tr>
<td>Arguments class method</td>
<td>8-23</td>
</tr>
<tr>
<td>CILRefArg class method</td>
<td>8-28</td>
</tr>
<tr>
<td>GetElement agent object method</td>
<td>5-12</td>
</tr>
<tr>
<td>GetElement Arguments class method</td>
<td>8-23</td>
</tr>
<tr>
<td>GetElement call object method</td>
<td>6-14</td>
</tr>
<tr>
<td>GetMonitoredAgent agent object method</td>
<td>5-12</td>
</tr>
<tr>
<td>GetMonitoredCall agent object method</td>
<td>5-14</td>
</tr>
<tr>
<td>GetPropertyAttribute agent object method</td>
<td>5-16</td>
</tr>
<tr>
<td>GetPropertyAttribute call object method</td>
<td>6-14</td>
</tr>
<tr>
<td>GetPropertyAttribute session object method</td>
<td>4-9</td>
</tr>
<tr>
<td>GetPropertyAttribute SkillGroup object method</td>
<td>7-4</td>
</tr>
<tr>
<td>GetSkillGroups agent object method</td>
<td>5-21</td>
</tr>
<tr>
<td>GetType</td>
<td></td>
</tr>
<tr>
<td>Arg class method</td>
<td>8-14</td>
</tr>
<tr>
<td>CILRefArg class method</td>
<td>8-29</td>
</tr>
<tr>
<td>GetValue</td>
<td></td>
</tr>
<tr>
<td>Arguments class method</td>
<td>8-23</td>
</tr>
<tr>
<td>CILRefArg class method</td>
<td>8-27</td>
</tr>
</tbody>
</table>
GetValue agent object method 5-21
GetValue call object method 6-16
GetValue session object method 4-12
GetValue SkillGroup object method 7-23
GetValue Type
Arg class method 8-9

H
Hold call object method 6-17

I
IAgentEvents interface 5-45
ICallEvents interface 6-42
In 3-2
Initialize
Arguments class method 8-24
isAgentMode session object method 4-12
ISessionEvents interface 4-27
ISkillGroupEvents interface 7-24
isSupervisorMode session object method 4-14
IsValid
Arguments class method 8-24

L
Login agent object method 5-22
Logout agent object method 5-24

M
MakeCall agent object method 5-26
MakeConsultCall call object method 6-19
MakeRequest session object method 4-15
MonitorAgentTeamAll agent object method 5-30, 5-31
Monitor mode 1-2

N
NumElements
Arguments class method 8-24

O
Object Interface Framework 2-1
OnAgentPrecallAbortEvent event 6-67
OnAgentPrecallEvent event 6-65
OnAgentStateChange event 5-46
OnAgentStatistics event 5-47
OnCallBegin event 6-42
OnCallCleared event 6-53
OnCallConferenced event 6-59
OnCallConnectionCleared event 6-53
OnCallDataUpdate event 6-45
OnCallDelivered event 6-46
OnCallDequeuedEvent event 6-70
OnCallDiverted event 6-61
OnCallEnd event 6-44
OnCallEnterpriseAgent event  6-63
OnCallEstablished event  6-48
OnCallFailed event  6-56
OnCallHeld event  6-51
OnCallOriginated event  6-54
OnCallPreEventAbort event  6-65
OnCallQueuedEvent event  6-68
OnCallReachedNetworkEvent event  6-71
OnCallRequestFailed event  6-65
OnCallRetrieved event  6-52
OnCallServiceInitiated event  6-73
OnCallServiceInitiatedEvent event  6-67
OnCallTransferConferenceInitiated event  6-57
OnCallTransferred event  6-57
OnConnectionClosed message  4-29
OnConnection event  4-27
OnConnectionFailure event  4-28
OnConnectionRejected message  4-30
OnControlFailureConf event  6-72
OnCurrentAgentReset message  4-30
OnCurrentCallChanged message  4-30
OnEvent agent object method  5-33
OnEvent call object method  6-22
OnEvent session object method  4-18
OnEvent SkillGroup object method  7-24
OnHeartbeat event  4-28
OnMissingHeartbeat event  4-28
OnMonitorModeEstablished event  4-29
OnSetAgentModeEvent message  4-30
OnSkillGroupStatisticsUpdated event  7-24
OnSnapshotCallConf event  6-72
OnTranslationRoute event  6-62
OnUserMessage event  5-47

P
property attributes  1-9
property names  1-9

R
Reconnect call object method  6-23
Record call object method  6-25
RemoveItem
  Arguments class method  8-25
RequestDesktopSettings session object method  4-18
Retrieve call object method  6-27

S
SendChatMessage agent object method  5-34
SendDTMFSignal call object method  6-29
SendUserMessage agent object method  5-34
session object methods  4-4
  Connect  4-5
  Disconnect  4-8
 GetPropertyAttribute  4-9
Index

GetValue  4-12
isAgentMode  4-12
isSupervisorMode  4-14
MakeRequest  4-15
OnEvent  4-18
RequestDesktopSettings  4-18
SetAgent  4-19
SetCurrentCall  4-21
SetMessageFilter  4-23
SetAgent session object method  4-19
SetAgentState agent object method  5-36
SetCallData call object method  6-31
SetCurrentCall session object method  4-21
SetElement
  Arguments class method  8-25
SetMessageFilter session object method  4-23
SetMonitoredAgent agent object method  5-38
SetMonitoredCall agent object method  5-40
SetValue
  Arg class method  8-5
  Arguments class method  8-25
SetValue agent object method  5-41
SingleStepConference call object method  6-34
SingleStepTransfer call object method  6-36
SkillGroup object methods  7-2
  DisableSkillGroupStatistics  7-2
  EnableSkillGroupStatistics  7-2
GetPropertyAttribute  7-4
GetValue  7-23
OnEvent  7-24
Snapshot call object method  6-38
SuperviseCall agent object method  5-42
Supervisor Assist agent object method  5-44
Transfer call object method  6-40