System Requirements for Building Custom Applications 33
Environment Set Up for .NET 34
  Microsoft Visual Studio 34
    Add CTI OS Toolkit Components to Add Reference Dialog Box 34
    Add Cisco CTI OS ActiveX Controls to Toolbox 35
Integration Between Your Application and CTI OS via CIL 35
  Integration Planning and Design 36
  Language and Interface 36
CTI Application Testing 37
  Test Plan Development 37
  Test Environment 38
Developer Sample Applications 38
CTI OS ActiveX Controls 39
  Build Simple Softphone with ActiveX Controls 39
  Hook for Screenpops 41
    CTI OS SessionResolver 42
    VB .NET Code Sample to Retrieve Common Session 42
COM CIL. in Visual Studio 43
  COM CIL. 43
  Add COM Support to Your Application 44
    Important Note About COM Method Syntax 44
Use CIL Dynamic Link Libraries 44
Create COM Object at Run Time 45
COM Events in C++ 45
  Additional Information 46
C++ CIL and Static Libraries 46
  Header Files and Libraries 46
  Configure Project Settings for Compiling and Linking 47
Event Subscription in C++ 51
Removal of STLPort Requirement 52
  Additional Information 52
Java CIL Libraries 52
  Additional Information 52
.NET CIL Libraries 53
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Information</td>
<td>53</td>
</tr>
<tr>
<td>CTI OS Server Connection</td>
<td>53</td>
</tr>
<tr>
<td>Connect to CTI OS Server</td>
<td>54</td>
</tr>
<tr>
<td>Session Object Lifetime (C++ Only)</td>
<td>54</td>
</tr>
<tr>
<td>Set Event Listener and Subscribe to Events</td>
<td>54</td>
</tr>
<tr>
<td>Set Connection Parameters for Session</td>
<td>55</td>
</tr>
<tr>
<td>Connect Session to CTI OS Server</td>
<td>55</td>
</tr>
<tr>
<td>Connection Failures</td>
<td>55</td>
</tr>
<tr>
<td>Connection Failure Events</td>
<td>56</td>
</tr>
<tr>
<td>Connection Attempt Error Codes in Java and .NET CIL</td>
<td>56</td>
</tr>
<tr>
<td>Configure Agent to Automatically Log In After Failover</td>
<td>57</td>
</tr>
<tr>
<td>Stop Failover Procedure</td>
<td>57</td>
</tr>
<tr>
<td>Connection Mode</td>
<td>57</td>
</tr>
<tr>
<td>Set Connection Mode in OnConnection() Event Handler</td>
<td>57</td>
</tr>
<tr>
<td>Agent Mode</td>
<td>57</td>
</tr>
<tr>
<td>Select Agent Mode</td>
<td>58</td>
</tr>
<tr>
<td>Monitor Mode</td>
<td>58</td>
</tr>
<tr>
<td>Monitor Mode Filters</td>
<td>59</td>
</tr>
<tr>
<td>Select Monitor Mode</td>
<td>61</td>
</tr>
<tr>
<td>Deal with Failover in Monitor Mode</td>
<td>61</td>
</tr>
<tr>
<td>Settings Download</td>
<td>62</td>
</tr>
<tr>
<td>Disconnect from CTI OS Server Before Shutdown</td>
<td>63</td>
</tr>
<tr>
<td>Agent Login and Logout</td>
<td>64</td>
</tr>
<tr>
<td>Log In an Agent</td>
<td>64</td>
</tr>
<tr>
<td>Duplicate Login Attempts</td>
<td>65</td>
</tr>
<tr>
<td>Overview of Duplicate Login Attempts</td>
<td>65</td>
</tr>
<tr>
<td>Create Values in CTI OS Server Registry to Control Duplicate Sign In Attempts</td>
<td>66</td>
</tr>
<tr>
<td>Agent Login with Incorrect Credentials</td>
<td>67</td>
</tr>
<tr>
<td>Get Registry Configuration Values to Desktop Application</td>
<td>67</td>
</tr>
<tr>
<td>Detect Duplicate Login Attempt in Desktop Application</td>
<td>68</td>
</tr>
<tr>
<td>Handle Duplicate Login Attempts in Desktop Application</td>
<td>69</td>
</tr>
<tr>
<td>Log Out an Agent</td>
<td>69</td>
</tr>
<tr>
<td>Typical Logout Procedure</td>
<td>70</td>
</tr>
<tr>
<td>Calls</td>
<td>72</td>
</tr>
</tbody>
</table>
Skill Group Statistics Sent to Desktop Application 86
Skill Group Statistics Computed by Sample CTI OS Desktop 86
Silent Monitoring 87
  CTI OS Based Silent Monitoring 88
  Create a Silent Monitor Object 88
  Session Mode 88
  Silent Monitor Session 89
  Silent Monitor Manager Shutdown 91
  CTI OS Silent Monitor Management in Monitor Mode 91
Unified CM-Based Silent Monitoring in Your Application 92
  CCM-Based Silent Monitor Overview 92
  CTI OS Monitor Mode Applications 92
  CCM-Based Silent Monitor Request 92
  Current Agent Being Silently Monitored 94
  CCM-Based Silent Monitor Request End 94
  Determine if CCM-Based Silent Monitoring Is Enabled 95
Agent Greeting 96
Deployment of Custom CTI OS Applications 96
  Application Deployment Using ActiveX Controls 96
  Application Deployment Using COM (but Not ActiveX Controls) 101
  Application Deployment Using C++ CIL 101
  Application Deployment Using .NET CIL 101
  Custom Application and CTI OS Security 102
Supervisor Applications 102
  General Flow 102
  Monitored and Unmonitored Events 103
  Supervisor Application Flow to Request and Monitor Team 103
    OnNewAgentTeamMember Events 104
    OnNewAgentTeamMember Events and Supervisors 105
    OnMonitoredAgentStateChange Events 105
    OnMonitoredAgentInfo Event 106
    Time in State 106
    OnSkillInfo Event 106
    Agent Team Information Displayed in Grid Format 106
Supervisor Application Flow to Monitor an Agent 108
  OnSupervisorButtonChange 111
  Monitored Call Events 111
  Supervisor Application Makes Agent Ready or Logs Agent Out 111
Supervisor Application Flow to Monitor a Call 112
  MonitoredCallEvents 112
  Barging into Calls 113
  Intercepting Calls 114
  Monitored Call Data 114
Sample Code in CTI OS Toolkit 115
  .NET Samples 115
    CTI Toolkit Combo Desktop.NET 115
    CtiOs Data Grid.NET 116
    All Agents Sample.NET 117
    All Calls Sample.NET 117
  Java CIL Samples 117
  Win32 Samples 117

CHAPTER 5
CTI OS ActiveX Controls 119
  CTI OS ActiveX Controls 119
  Property Pages 121
  Button Controls and Grid Controls 121
    Button Controls 126
    Grid Controls 126
    Supervisor Status Bar 127
CTI OS ActiveX Control Descriptions 127
  AgentGreetingCtl 127
    RecordGreetingCtl 127
  AgentStateCtl 127
    Related Methods 129
    Related Events 130
  AgentSelectCtl 131
    Methods 132
  AgentStatisticsCtl 138
Methods 138
AlternateCtl 142
AnswerCtl 143
BadLineCtl 143
CallAppearanceCtl 143
  Related Methods 144
  Related Events 144
  Methods 144
ChatCtl 148
  Methods 149
ConferenceCtl 151
EmergencyAssistCtl 153
HoldCtl 153
MakeCallCtl 154
ReconnectCtl 155
SkillgroupStatisticsCtl 155
  Methods 155
StatusBarCtl 159
SupervisorOnlyCtl 160
RecordCtl 161
TransferCtl 161
The Silent Monitor StandAlone ActiveX Control 163
  Connect 163
  Disconnect 164
  StartMonitoring 164
  StopMonitoring 164
  SilentMonitor Com Object Events 164
Deployment 165
  Sample Usage in Visual Basic 6.0 165

Chapter 6  Event Interfaces and Events 167
  Event Interfaces and Events 167
  Event Publication Model 168
  ISessionEvents Interface 168
OnConnection 169
OnConnectionClosed 169
OnConnectionFailure 169
OnConnectionRejected 170
OnCTIOSFailure 170
OnCurrentAgentReset 172
OnCurrentCallChanged 172
OnFailure Event 173
OnGlobalSettingsDownloadConf 173
OnHeartbeat 182
OnMissingHeartbeat 183
OnMonitorModeEstablished 183
OnSnapshotDeviceConf 184
OnSnapshotSkillGroupList 185
OnTranslationRoute 185
ICallEvents Interface 187
OnAgentPrecallEvent 187
OnAgentPrecallAbortEvent 189
OnAlternateCallConf 190
OnAnswerCallConf 191
OnCallBegin 192
OnCallCleared 194
OnCallConnectionCleared 195
OnCallConferenced 196
OnCallDataUpdate 199
OnCallDelivered 201
OnCallDequeuedEvent 203
OnCallDiverted 204
OnCallEnd 205
OnCallEstablished 206
OnCallFailed 208
OnCallHeld 209
OnCallOriginated 209
OnCallQueuedEvent 211
OnCallReachedNetworkEvent 212
OnCallRetrieved 213
OnCallServiceInitiatedEvent 214
OnCallStartRecordingConf 215
OnCallStopRecordingConf 216
OnCallTransferred 217
OnClearCallConf 219
OnClearConnectionConf 220
OnConferenceCallConf 220
OnConsultationCallConf 221
OnControlFailureConf 221
OnHoldCallConf 222
OnMakePredictiveCallConf 223
OnReconnectCallConf 223
OnReleaseCallConf 223
OnRetrieveCallConf 224
OnSendDTMFCConf 224
OnSetCallDataConf 225
OnSnapshotCallConf 225
OnTransferCallConf 228
IAgentEvents Interface 228
OnAgentDeskSettingsConf 229
OnAgentGreetingControlConf 231
OnAgentInfoEvent 232
OnAgentStateChange 232
OnAgentStatistics 235
OnChatMessage 236
OnControlFailureConf 236
OnEmergencyCall 242
OnLogoutFailed 243
OnMakeCallConf 243
OnNewAgentTeamMember 244
OnPostLogout 246
OnPreLogout 248
OnQueryAgentStateConf  250
OnSetAgentModeEvent  253
OnSetAgentStateConf  254
OnStartMonitoringAgent  255
OnStopMonitoringAgent  255
OnUserMessageConf  256
ISkillGroupEvents Interface  256
OnSkillGroupStatisticsUpdated  257
OnSkillInfoEvent  257
IButtonEnablementEvents  258
OnButtonEnablementChange  258
OnSupervisorButtonChange  261
IMonitoredAgentEvents Interface  262
IMonitoredCallEvents Interface  262
ISilentMonitorEvents  263
OnCallRTPStarted  264
OnCallRTPStopped  265
OnStartSilentMonitorConf  267
OnSilentMonitorStartedEvent  268
  For CTI OS Based Silent Monitor  268
  For CCM-Based Silent Monitor  269
OnSilentMonitorStartRequestedEvent  270
OnSilentMonitorSessionDisconnected  272
OnSilentMonitorStopRequestedEvent  273
  For CTI OS Based Silent Monitor  273
  For CCM-Based Silent Monitor  274
OnSilentMonitorStatusReportEvent  275
OnStopSilentMonitorConf  277
OnRTPStreamTimedoutEvent  279
IGenericEvents Interface  280
  OnEvent  280
Java Adapter Classes  280
  IAllInOne  280
  IAgentEvents  280
CtiOs Object 291

CtiOs Object 291

Methods 291

DumpProperties 292

GetAllProperties 293

GetElement 293

GetLastError (Java and .NET Only) 294

GetNumProperties 295

GetPropertyType 296

GetPropertyName 296

GetValue 298

GetValueArray 299

GetValueBoolObj (Java and .NET Only) 299

GetValueInt 300

GetValueIntObj (Java Only) 301

GetValueShortObj (Java Only) 301

GetValueString 301

GetValueUIntObj (Java Only) 303

GetValueUShortObj (Java Only) 303

IsValid 303

ReportError (Java and .NET only) 304

SetValue (Java and .NET) 305

SetValue (C++ COM and VB) 305

Session Object 307

Session Object 307
Session Object Properties 308

Methods 310

AddEventListener (Java and .NET Only) 312
AddListener Methods (C++ Only) 312
Connect 313
CreateSilentMonitorManager 315
CreateWaitObject (C++ Java and .NET) 316
DestroySilentMonitorManager 317
DestroyWaitObject (C++ Java and .NET) 318
DisableSkillGroupStatistics (C++ Java and .NET) 318
Disconnect 319
DumpProperties 319
EnableSkillGroupStatistics (C++ Java and .NET) 320
GetAllAgents 320
GetAllCalls 323
GetAllProperties 325
GetAllSkillGroups 325
GetCurrentAgent 326
GetCurrentCall 326
GetCurrentSilentMonitor 327
GetElement 328
GetNumProperties 328
GetObjectFromObjectId 328
GetPropertyId 329
GetPropertyType 329
GetSystemStatus (Java .NET and C++ Only) 329
GetValue Methods 329
IsAgent 329
IsCCMSilentMonitor 330
IsSupervisor 331
IsValid 331
RemoveEventListener (Java and .NET) 331
RemoveListener Methods (C++ Only) 332
RequestDesktopSettings 332
GetValue Methods 362
IsAgent 362
IsSupervisor 363
Login 363
Logout 366
MakeCall 367
MakeEmergencyCall 372
QueryAgentState 373
ReportBadCallLine 374
RequestAgentTeamList 375
RequestSupervisorAssist 376
SendChatMessage 377
SetAgentGreetingAction 378
SetAgentState 379
StartMonitoringAgent 380
StartMonitoringAgentTeam 381
StartMonitoringAllAgentTeams 382
StartMonitoringCall 383
StopMonitoringAgent 384
StopMonitoringAgentTeam 385
StopMonitoringAllAgentTeams 385
SuperviseCall 386

CHAPTER 10 389

Call Object 389
Call Object 389
Current Call 389
ECC Variables 390
Passing Call Variables 390
ECC Variable Value Retrieval 390
ECC Values 391
Properties 393
Methods 395
Argument Parameters 396
Alternate 397
Arg Class  442
  AddRef  442
  Clone  443
  CreateInstance  444
  DumpArg  444
  GetArgType (.NET Only)  445
  GetType  446
  GetValue Methods  447
  Release  448
  SetValue  449
Arguments Class  450
  Usage Notes  451
  AddItem (C++ COM VB Only)  452
  AddRef (C++ and COM Only)  453
  Clear  454
  Clone  454
  CreateInstance (C++ and COM Only)  455
  DumpArgs  456
  GetElement Methods  456
  GetValue Methods  458
  IsValid  460
  NumElements  461
  Release (C++ and COM Only)  462
  RemoveItem  462
  SetElement (C++ COM and VB Only)  463
  SetValue  464
CILRefArg Class (C++ Java and .NET Only)  465
  GetType  466
  GetUniqueObjectID (Java and .NET Only)  467
  GetValue  467
  SetValue  467
CCtiOsException Class (C++ Java and .NET Only)  468
  CCtiOsException Constructor  468
  GetCode  469
GetStatus 469
GetString 469
What 470
CWaitObject Class 470
Methods 470
CreateWaitObject 471
DestroyWaitObject 471
DumpEventMask 471
GetMask 472
GetTriggerEvent 472
InMask 473
SetMask 473
WaitOnMultipleEvents 473
Logger Class (.NET and Java Only) 474
Methods 474
Logger() Constructor 475
GetTraceMask 475
SetTraceMask 476
AddLogListener 476
RemoveLogListener 476
Trace 477
LogWrapper Class (.NET and Java Only) 477
Methods 477
LogWrapper() Constructor 478
LogWrapper(string filename) Constructor 479
Return Values 479
LogWrapper(string int int int) Constructor 479
Dispose (.NET Only) 480
GetMaxDaysBeforeExpire (.NET Only) 480
SetMaxNumberFiles 481
GetMaxNumberFiles (.NET Only) 481
SetMaxDaysBeforeExpire 482
ProcessConfigFile 482
CHAPTER 13  SilentMonitorManager Object  483

SilentMonitorManager Object  483

Properties  484

Methods  485

Argument Parameter Rules  486

AcceptSilentMonitoring  486

GetIPPhoneInfo  488

GetSessionInfo  489

GetSMSessionList  490

IsMonitoredTarget  491

SetIPPhoneInfo  491

StartSilentMonitorRequest  493

StartSMMonitoredMode  494

StartSMMonitoringMode  496

StopSilentMonitorMode  497

StopSilentMonitorRequest  497

CHAPTER 14  CTI OS Keywords and Enumerated Types  499

Keywords  499

Java CIL Keywords  500

.NET CIL Keywords  500

Enumerated Types  500

Java Interfaces  500

APPENDIX A  CTI OS Logging  503

CTI OS Client Logs (COM and C++)  503

Install Tracing Mechanism (COM and C++)  503

Set Trace Levels (COM and C++)  504

Trace Configuration (COM and C++)  504

Java CIL Logging Utilities  505

ILogEvents  505

LogEvent  505

Logger  506
LogEventsAdapter  506
Logging and tracing (Java)  507
Logging and tracing (.NET)  508
  Default Logging Mechanism  508
    Log Trace Events with LogWrapper Class  508
Custom Logging Mechanism  510
  Log Trace Events with Logger Class  510
Trace Configuration (Java and .NET)  511
Preface

- Change History, on page xxiii
- About This Guide, on page xxiii
- Audience, on page xxiv
- Related Documents, on page xxiv
- Communications, Services, and Additional Information, on page xxiv
- Field Alerts and Field Notices, on page xxiv
- Documentation Feedback, on page xxv
- Conventions, on page xxv

Change History

This table lists changes made to this guide. Most recent changes appear at the top.

<table>
<thead>
<tr>
<th>Change</th>
<th>See</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Release of document for Release 12.0</td>
<td></td>
<td>January 2019</td>
</tr>
<tr>
<td>Added a table that includes the software versions for building custom applications.</td>
<td>System Requirements for Building Custom Applications</td>
<td></td>
</tr>
<tr>
<td>Updated the Visual Studio Version to 2015</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Updated the .NET Framework version to 4.7.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Updated the Java version to 1.8 update 161</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

About This Guide

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

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

Related Documents

<table>
<thead>
<tr>
<th>Subject</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related documentation includes the documentation sets for Cisco CTI Object Server (CTI OS), Cisco Unified Contact Center Management Portal, Cisco Unified Customer Voice Portal (Unified CVP), Cisco Unified IP IVR, and Cisco Unified Intelligence Center</td>
<td>To see all related documentation sets, go to <a href="https://www.cisco.com/cisco/web/psa/default.html?mode=prod">https://www.cisco.com/cisco/web/psa/default.html?mode=prod</a>. Select <strong>Products &gt; Customer Collaboration &gt; Contact Center.</strong></td>
</tr>
</tbody>
</table>

Communications, Services, and Additional Information

- To receive timely, relevant information from Cisco, sign up at [Cisco Profile Manager](https://www.cisco.com/cisco/profile-manager).
- To get the business impact you’re looking for with the technologies that matter, visit [Cisco Services](https://www.cisco.com/cisco/services).
- To submit a service request, visit [Cisco Support](https://www.cisco.com/cisco/support).
- To discover and browse secure, validated enterprise-class apps, products, solutions and services, visit [Cisco Marketplace](https://www.cisco.com/cisco/marketplace).
- To obtain general networking, training, and certification titles, visit [Cisco Press](https://www.cisco.com/cisco/press).
- To find warranty information for a specific product or product family, access [Cisco Warranty Finder](https://www.cisco.com/cisco/warrantyfinder).

Cisco Bug Search Tool

Cisco Bug Search Tool (BST) is a web-based tool that acts as a gateway to the Cisco bug tracking system that maintains a comprehensive list of defects and vulnerabilities in Cisco products and software. BST provides you with detailed defect information about your products and software.

Field Alerts and Field Notices

Cisco can modify its products or determine key processes to be important. These changes are announced through use of the Cisco Field Alerts and Cisco Field Notices. You can register to receive Field Alerts and Field Notices through the Product Alert Tool on Cisco.com. This tool enables you to create a profile to receive announcements by selecting all products of interest.
Sign in [www.cisco.com](http://www.cisco.com) and then access the tool at [https://www.cisco.com/cisco/support/notifications.html](https://www.cisco.com/cisco/support/notifications.html).

**Documentation Feedback**

To provide comments about this document, send an email message to the following address: contactcenterproducts_docfeedback@cisco.com.

We appreciate your comments.

**Conventions**

This document uses the following conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface font</strong></td>
<td>Boldface font is used to indicate commands, such as user entries, keys, buttons, and folder and submenu names. For example:</td>
</tr>
<tr>
<td></td>
<td>• Choose <strong>Edit &gt; Find</strong>.</td>
</tr>
<tr>
<td></td>
<td>• Click <strong>Finish</strong>.</td>
</tr>
<tr>
<td><strong>italic font</strong></td>
<td>Italic font is used to indicate the following:</td>
</tr>
<tr>
<td></td>
<td>• To introduce a new term. Example: A <em>skill group</em> is a collection of agents who share similar skills.</td>
</tr>
<tr>
<td></td>
<td>• A syntax value that the user must replace. Example: IF <em>(condition, true-value, false-value)</em></td>
</tr>
<tr>
<td></td>
<td>• A book title. Example: See the <em>Cisco Unified Contact Center Enterprise Installation and Upgrade Guide</em>.</td>
</tr>
<tr>
<td><strong>window font</strong></td>
<td>Window font, such as Courier, is used for the following:</td>
</tr>
<tr>
<td></td>
<td>• Text as it appears in code or that the window displays. Example:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;html&gt;&lt;title&gt;Cisco Systems, Inc. &lt;/title&gt;&lt;/html&gt;</code></td>
</tr>
<tr>
<td><strong>&lt; &gt;</strong></td>
<td>Angle brackets are used to indicate the following:</td>
</tr>
<tr>
<td></td>
<td>• For arguments where the context does not allow italic, such as ASCII output.</td>
</tr>
<tr>
<td></td>
<td>• A character string that the user enters but that does not appear on the window such as a password.</td>
</tr>
</tbody>
</table>
Introduction

• Introduction to CTI, on page 1
• CTI-Enabled Applications, on page 1
• Events and Requests Within CTI Environment, on page 2
• Overview of CTI OS, on page 4

Introduction to CTI

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

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

Integrating communications media with the customer service platform helps agents serve customers improved and faster in the following two ways:

• It enables the agent to leverage the information and events provided by the media to direct workflow.
• It increases the depth and breadth of customer information presented to the agent when the customer’s contact arrives at the workstation.

CTI-Enabled Applications

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

Screen Pop

The most common CTI application is a screen pop. In a screen pop, the customer service platform is provided with customer information at the arrival of a phone call and begins processing the customer's transaction at the same time as the communication begins between the customer and the agent. This transfer of customer information is called the call context information: a rich set of customer-specific data that travels with the call throughout the enterprise.
Forexample, a phone call can trigger a screen pop application for a cellular telephone company. It uses the customer ANI (automated number identification, or calling line ID) to do a database look up to retrieve the customer's account information and displays this customer record for the agent. By the time the agent can say “Thank you for calling ABC Telephony Company,” the account record is on the screen and the agent is ready to service the customer's request.

**Agent State Control**

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

For example, a sales application can automatically send an agent to a wrap-up or after-call work state when the customer contact terminates. The agent can then enter wrap up data about that transaction or customer inquiry and (subject to a timer) change the state automatically back to available when the wrap up work is complete.

**Third-Party Call Control**

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

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

**Events and Requests Within CTI Environment**

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

**Asynchronous Events**

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

The following figure depicts the stages of a typical inbound telephone call and its associated events.

*Figure 1: Typical Inbound Call Events Flow*

The following events are generated, based on the state of the call:
OnCallBegin: Indicates that the call has entered the setup phase.
OnCallDelivered: Generated when the call starts ringing.
OnCallEstablished: Generated when the call is answered.
OnCallCleared: Generated when the voice connection is terminated (e.g. call hung up).
OnCallEnd: Generated when the logical call appearance (including call data) is complete.

In addition to the events and states shown in the figure above, the following are typical call events that CTI applications use:

OnCallHeld: Generated when the call transitions from the active to held state.
OnCallRetrieved: Generated when the call is removed from hold.
OnCallTransferred: Indicates that the call has been transferred to another party.
OnCallConferenced: Indicates that a new party has been added to the call.

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

**Request-Response Paradigm**

In addition to responding to asynchronous events, a CTI enabled application can make programmatic requests for services via the CTI interface. Specifically, the CTI application uses the request-response mechanism to perform agent state and third-party call control, and to set call context data.

The typical request-response flow for CTI uses the model shown in the following figure:

*Figure 2: Sample Request-Response Message Flow*

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

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

Figure 3: CTI OS Three-Tiered Architecture Topology

The CTI OS application architecture employs three tiers:

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

Advantages of CTI OS as a CTI Development Interface

CTI OS brings several major advances to developing custom CTI integration solutions. The CIL provides an object-oriented and event driven application programming interface (API), while the CTI OS server does all the heavy-lifting of the CTI integration: updating call context information, determining which buttons to enable on softphones, providing easy access to supervisor features, and automatically recovering from failover scenarios.

- **Rapid integration.** Developing CTI applications with CTI OS is significantly easier and faster than any previously available Cisco CTI integration platform. The same object oriented interface is used across programming languages, enabling rapid integrations in .NET, and C++, Visual Basic, or any Microsoft COM compliant container environment. Developers can use CTI OS to create a screen pop application in as little as five minutes. The only custom-development effort required is within the homegrown application to which CTI is being added.

- **Complex solutions made simple.** CTI OS enables complex server-to-server integrations and multiple agent monitoring-type applications. The CIL provides a single object-oriented interface that you can use
in two modes: Agent Mode and Monitor Mode. See CTI OS Client Interface Library Architecture, on page 7 for an explanation of these two modes.

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

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

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

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

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

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

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

**Illustrative Code Fragments**

Throughout this manual, illustrative code fragments are provided both to clarify the usage and as examples. These fragments are written in several languages, including .NET (VB .NET). Note that the VB code fragments are written using the VB 6 syntax.
CHAPTER 2

CTI OS Client Interface Library Architecture

- Object Server Architecture, on page 7
- Client Interface Library Architecture, on page 7
- CIL Object Model Object Interfaces, on page 9
- Where to Go from Here, on page 17

Object Server Architecture

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

All objects are identified by a UniqueObjectID. The UniqueObjectID is the key that maps a server-side object and the client-side proxy (or proxies). Requests made on a client-side object are sent to the CTI OS Server, and the corresponding server-side object services the request (see the following figure).

Figure 4: CTI OS Object Server and Client Object Sharing

Client Interface Library Architecture

The Client Interface Library has a three-tiered architecture (see figure below), which implements the functionality provided to developers. The CIL architecture comprises the Connection Layer, the Service Layer and Object Interface Layer. The CIL architecture also includes the custom application, which the customer develops to use the Client interface Library services.
Connection Layer

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

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

The C++ CIL connection objects offered a parameter for setting QoS markings (DSCP packet markings). This mechanism does not work when the desktop is deployed on Windows 7. If you require QoS markings on these platforms, manage QoS across the enterprise with a Group Policy. Group Policies are administered using Active Directory, but that information is beyond the scope of this document.

For more information about C++ CIL connection objects, see ISessionEvents Interface, on page 168.

For additional information about QoS and DSCP, see the Solution Design Guide for Cisco Unified Contact Center Enterprise.

For additional information about CTI OS QoS support, see the CTI OS System Manager Guide for Cisco Unified ICM.

Service Layer

The Service Layer sits between the Connection Layer and the Object Interface Layer. Its main purpose is to translate the low-level network packets the Connection Layer sends and receives and the high-level command and event messages the Object Interface Layer uses. The Service Layer implements a generic message serialization protocol which translates key-value pairs into a byte stream for network transmission and deserializes the messages back to key-value pairs on the receiving side. This generic serialization mechanism ensures forward-compatibility, because future enhancements to the message set do not require any changes at the Connection or Service Layers.

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

**Object Interface Layer**

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

**Custom Application**

The custom application is the business application that is developed to integrate with the CTI OS Client Interface Library. The custom application uses the CIL in the two following ways:

- The CIL provides the object-based interface for interacting with CTI OS, to send requests for agent and call control.
- The CIL provides an events subscription service, which the custom application takes advantage of to receive events from CTI OS.

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

**CIL Object Model Object Interfaces**

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

*Figure 6: CIL Object Model Object Interfaces*

![CIL Object Model Object Interfaces](image)

**Session Object**

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

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

Session Modes

You can set a Session object to work in one of two modes: Agent Mode or Monitor Mode. The Session object maintains the state of the Session mode, and recovers the session mode during failover. The client application must set the session mode after it connects to the CTI OS Server; the Session mode remains active until the connection to the CTI OS Server is closed.

Agent Mode

A client connects to CTI OS Server in Agent Mode when it wants to receive events for a specific agent or supervisor. After you set the Agent Mode, the CIL receives the events for the specified agent, as well as all call events for that agent's calls. If you also configure the agent as a Supervisor in Unified ICM, the CIL receives events for all agents in the Supervisor's team.

Monitor Mode

A client connects to the CTI OS Server in Monitor Mode when it wants to receive a programmer-specified set of events, such as all agent state events. For more information about setting up a Monitor Mode connection, see Select Monitor Mode, on page 61.

For the complete interface specification of the Session object, see Session Object, on page 307

Agent Object

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

For the complete interface specification of the Agent object, see Agent Object, on page 343

Call Object

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

For the complete interface specification of the Call object, see Call Object, on page 389

SkillGroup Object

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

For the complete interface specification of the SkillGroup object, see SkillGroup Object, on page 423
Object Creation

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

Objects are created either by the programmer or by the Session object as required to support the event flow received from the CTI OS Server. In Agent Mode, the programmer creates a single Agent object with which to log in; in Monitor Mode, Agent objects are created as required by the event flow. Call and SkillGroup objects are always created by the Session object.

An Agent, Call, or SkillGroup object is created (by the Session) when the Session receives an event for an object (identified by its UniqueObjectID) that is not yet present at the CIL. This ensures that the CIL always has the appropriate collection of proxy objects, one for each object on the CTI OS Server that it is using. When a new object is created, it is added to the Session object's collection, and is accessible from the Session via the GetValue mechanism. See Session Object, on page 307

Reference Counting

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

An application or object that holds a reference to a CIL object must use the AddRef method to add to its reference count. When the reference is no longer required, the application or object holding that reference must use the Release() method to decrement the reference count. Reference counting is discussed further in CtiOs Object, on page 291.

Note

Reference counting must be done explicitly in C++ applications (COM or non-COM). Visual Basic, Java, and the .NET framework perform automatic reference counting.

Call Object Lifetime

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

The Call object remains valid at the CIL until the receipt of the OnCallEnd event. When the OnCallEnd event is received, the Session object publishes the event to any subscribers to the event interfaces. Applications and objects must release any remaining references to the Call object within their event handler for OnCallEnd to delete the Call object. When the Call object's OnEvent method returns after handling OnCallEnd, the Session checks the reference count for zero; if any references remain, the Call object is removed from the Call object collection but is not deleted until the last reference to it is released.

Agent Object Lifetime

In Agent Mode, the client programmer must create an Agent object, which causes its reference count to be incremented to one, and must pass it to the Session in the SetAgent method.
In C++, you must create the object on the heap memory store so that it can exist beyond the scope of the method creating it. For clients using other CILs, this is handled automatically.

The Session holds a reference to the Agent object as long as it is in use, but the client programmer must release the last reference to the object to prevent a memory leak.

In Monitor Mode, objects are created at the CIL when the CIL receives an event for that agent for the first time (for example, in an OnAgentStateChange event). When the Session receives an event for an unrecognized Agent, that new Agent is added to the Session's collection of agents.

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

### SkillGroup Object Lifetime

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

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

### Methods That Call AddRef()

The following tables detail the various methods that call AddRef(). To prevent memory leaks, C++ and COM application developers that call these methods in their applications must be aware of the impact of these methods on the reference count and must appropriately release the reference when no longer using the object:

#### Table 1: SessionLib (C++)

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Method Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAgent</td>
<td>GetSkillGroups(), GetMonitoredCall()</td>
<td>The client application must call Release() on the returned object when the object is no longer needed.</td>
</tr>
<tr>
<td>CILRefArg</td>
<td>CreateInstance(), GetValue()</td>
<td>The client application must call Release() on the returned object when the object is no longer needed.</td>
</tr>
<tr>
<td>CILRefArg</td>
<td>SetValue(), operator=</td>
<td>These methods increment the reference count on the passed in object. When the CILRefArg is deleted the reference count of the enclosed object is decremented.</td>
</tr>
<tr>
<td>Object Name</td>
<td>Method Name</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CCtiOsSession</td>
<td>SetCurrentCall()</td>
<td>This method increments the reference count on the passed in object. The previous &quot;current&quot; call's reference count is decremented. If an end call event is received for the current call, its reference count is decremented one extra time.</td>
</tr>
<tr>
<td>CCtiOsSession</td>
<td>DestroyWaitObject()</td>
<td>This method call decrements the reference count on the passed in object.</td>
</tr>
<tr>
<td>CCtiOsSession</td>
<td>CreateWaitObject()</td>
<td>The client application must call DestroyWaitObject() on the returned object when the object is no longer needed.</td>
</tr>
<tr>
<td>CCtiOsSession</td>
<td>DestroySilentMonitorManager()</td>
<td>This method decrements the reference count of the passed in object.</td>
</tr>
<tr>
<td>CCtiOsSession</td>
<td>CreateSilentMonitorManager()</td>
<td>The client application must call DestroySilentMonitorManager() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td>CCtiOsSession</td>
<td>SetCurrentSilentMonitor()</td>
<td>This method increments the reference count on the passed in object. The previous &quot;current&quot; silent monitor's reference count is decremented.</td>
</tr>
<tr>
<td>CCtiOsSession</td>
<td>GetCurrentCall(),</td>
<td>The client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td></td>
<td>GetCurrentSilentMonitorManager(),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetAllCalls(),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetAllSkillGroups(),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetAllAgents(),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetCurrentAgent(),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetValue(),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetObjectFromObjectID()</td>
<td></td>
</tr>
<tr>
<td>CCtiOsSession</td>
<td>SetAgent()</td>
<td>This method increments the reference count on the passed in object. If the passed in object is NULL, then this method decrements the current Agent object's reference count.</td>
</tr>
</tbody>
</table>
### Methods That Call AddRef

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Method Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSilentMonitorManager</td>
<td>GetSessionInfo(),</td>
<td>The client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td></td>
<td>GetIPPhoneInfo(),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetSMSSessionList()</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2: CiosClient.dll (COM)**

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Method Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAgent</td>
<td>GetSkillGroups()</td>
<td>This method increments the reference count for every SkillGroup object, adds them to a safe array and then returns the safe array.</td>
</tr>
<tr>
<td>IAgent</td>
<td>GetMonitoredAgent(),</td>
<td>The client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td></td>
<td>GetMonitoredCall()</td>
<td></td>
</tr>
<tr>
<td>IAgent</td>
<td>GetValue(),</td>
<td>The client application must call Release() on the returned object (second argument) when it is no longer needed.</td>
</tr>
<tr>
<td></td>
<td>GetValueArray(),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetElement()</td>
<td></td>
</tr>
<tr>
<td>IAgent</td>
<td>GetAllProperties()</td>
<td>The client application must call Release() on the returned object (first argument) when it is no longer needed.</td>
</tr>
<tr>
<td>ISkillGroup</td>
<td>GetValue(),</td>
<td>The client application must call Release() on the returned object (second argument) when it is no longer needed.</td>
</tr>
<tr>
<td></td>
<td>GetValueArray(),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetElement()</td>
<td></td>
</tr>
<tr>
<td>ISkillGroup</td>
<td>GetAllProperties()</td>
<td>The client application must call Release() on the returned object (first argument) when it is no longer needed.</td>
</tr>
<tr>
<td>ICall</td>
<td>GetCallContext(),</td>
<td>The client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td></td>
<td>GetCallData()</td>
<td></td>
</tr>
<tr>
<td>ICall</td>
<td>GetValue(),</td>
<td>The client application must call Release() on the returned object (second argument) when it is no longer needed.</td>
</tr>
<tr>
<td></td>
<td>GetValueArray(),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetElement()</td>
<td></td>
</tr>
<tr>
<td>Object Name</td>
<td>Method Name</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ICall</td>
<td>GetAllProperties()</td>
<td>The client application must call Release() on the returned object (first argument) when it is no longer needed.</td>
</tr>
<tr>
<td>ISilentMonitorManager</td>
<td>SetMonitor()</td>
<td>This method increments the reference count of the passed in object and decrements the reference count of the previous monitor.</td>
</tr>
<tr>
<td>ISilentMonitorManager</td>
<td>GetMonitor()</td>
<td>The client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td>ISilentMonitorManager</td>
<td>GetSessionInfo(), GetIPPhoneInfo(), GetSMSSessionList(), GetValue(), GetValueArray(), GetElement()</td>
<td>The client application must call Release() on the returned object (second argument) when it is no longer needed.</td>
</tr>
<tr>
<td>ISilentMonitorManager</td>
<td>GetAllProperties()</td>
<td>The client application must call Release() on the returned object (first argument) when it is no longer needed.</td>
</tr>
<tr>
<td>ISession</td>
<td>SetAgent()</td>
<td>This method increments the reference count on the passed in object. If the passed in object is NULL, then this method decrements the current Agent object's reference count.</td>
</tr>
<tr>
<td>ISession</td>
<td>GetCurrentAgent(), GetCurrentCall()</td>
<td>The client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td>ISession</td>
<td>GetAllCalls()</td>
<td>This method increments the reference count for every Call object, adds them to a safe array and then returns the safe array.</td>
</tr>
<tr>
<td>ISession</td>
<td>GetAllAgents()</td>
<td>This method increments the reference count for every Agent object, adds them to a safe array and then returns the safe array.</td>
</tr>
<tr>
<td>Object Name</td>
<td>Method Name</td>
<td>Explanation</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ISession</td>
<td>GetAllSkillGroups()</td>
<td>This method increments the reference count for every SkillGroup object, adds them to a safe array and then returns the safe array.</td>
</tr>
<tr>
<td>ISession</td>
<td>GetValue()</td>
<td>The client application must call Release() on the returned object (second argument) when it is no longer needed.</td>
</tr>
<tr>
<td>ISession</td>
<td>GetValueArray(), GetElement()</td>
<td></td>
</tr>
<tr>
<td>ISession</td>
<td>GetAllProperties()</td>
<td>The client application must call Release() on the returned object (first argument) when it is no longer needed.</td>
</tr>
<tr>
<td>ISession</td>
<td>GetObjectFromObjectID()</td>
<td>The client application must call Release() on the returned object (second argument) when it is no longer needed.</td>
</tr>
<tr>
<td>ISession</td>
<td>CreateSilentMonitorManager()</td>
<td>The client application must call DestroySilentMonitorManager() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td>ISession</td>
<td>DestroySilentMonitorManager()</td>
<td>This method call decrements the reference count on the passed in object.</td>
</tr>
<tr>
<td>ISession</td>
<td>GetCurrentSilentMonitorManager()</td>
<td>The client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
</tbody>
</table>

Table 3: CitosComArguments.dll (COM)

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Method Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IArg</td>
<td>Clone()</td>
<td>The client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td>IArg</td>
<td>GetValueArray()</td>
<td>The client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td>IArg</td>
<td>GetValue()</td>
<td>If ARG_TYPE = ARG_ARRAY, the client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
</tbody>
</table>
Where to Go from Here

Subsequent chapters in this manual contain the following information:

- For information about CIL coding conventions, see CIL Coding Conventions, on page 19
- For information about building an application using the CIL, see Building Your Custom CTI Application, on page 33

Table 4: ArgumentsLib (C++)

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Method Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IArguments</td>
<td>GetValueArray(), GetValue(), GetElement()</td>
<td>The client application must call Release() on the returned object (second argument) when it is no longer needed.</td>
</tr>
<tr>
<td>IArguments</td>
<td>Clone()</td>
<td>The client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td>Arg</td>
<td>CreateInstance(), GetValueArray(), operator=</td>
<td>The client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td>Arguments</td>
<td>CreateInstance(), Clone(), GetValue(), GetValueArg, GetValueArray(), GetElement(), GetElementArg()</td>
<td>The client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td>Arguments</td>
<td>SetValue()</td>
<td>If the returned object is of type Arg or of type Arguments, the client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
<tr>
<td>Arguments</td>
<td>SetElement()</td>
<td>If the returned object is of type Arg or of type Arguments, the client application must call Release() on the returned object when it is no longer needed.</td>
</tr>
</tbody>
</table>
• For a description and syntax of the CIL programming interfaces, see Chapters 8 through 13.
CIL Coding Conventions

- CTI OS CIL Data Types, on page 19
- Asynchronous Program Execution, on page 20
- CIL Error Codes, on page 21
- COM Error Codes, on page 25
- Generic Interfaces, on page 26
- UniqueObjectID Variable-Length String, on page 28
- UniqueObjectID to Obtain Pointer or Reference, on page 29
- Button Enablement Masks, on page 30

CTI OS CIL Data Types

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

- **STRING**: A variable-length string variable. If a maximum length exists, it is listed with the parameter description.

- **INT**: A 32-bit wide integer.

- **UNSIGNED INT**: A 32-bit wide unsigned integer.

- **SHORT**: A 16-bit wide short integer.

- **UNSIGNED SHORT**: A 16-bit wide unsigned short integer.

- **BOOL**: A logical true or false variable. Different implementations use variables of different sizes to represent this type. In COM, the VARIANT_BOOL is used. Tests of variables of this data type must be against VARIANT_TRUE and VARIANT_FALSE and not simply against 0 or 1.

- **ARGUMENTS**: A custom data structure used by CTI OS, which holds a variable-length set of key-value pairs.

- **ARG**: An individual element (value), which can be stored in an ARGUMENTS structure.

The following table describes the appropriate language specific types to which the documented type are associated.
Asynchronous Program Execution

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

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

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

To determine the success or failure of the underlying telephony operation requested, the CTI programmer must wait for an event confirming the success or failure of the request. To generalize the message flow model, most requests made at the CTI OS CIL are answered with a confirmation message and/or an event message. See the object interface reference in Chapters 8-12 for details on each particular request. This type of response is called asynchronous—it can arrive at any time after the request is made, but typically requests are services in sub-second timeframes.

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

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

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

# CIL Error Codes

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

The possible values of the error code returned from C++ and Java CIL methods are defined in the following table.

> **Note**
The numeric values listed in the following table are subject to change. Use the error code enumerations to check a given error code, rather than rely on a specific numeric value.

<table>
<thead>
<tr>
<th>CIL Error Code</th>
<th>Numeric Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIL_OK</td>
<td>1</td>
<td>The method succeeded. The request to silent monitor the call was successfully initiated.</td>
</tr>
<tr>
<td>CIL_FAIL</td>
<td>0</td>
<td>The method failed.</td>
</tr>
<tr>
<td>E_CTIOS_METHOD_NO_IMPLEMENTED</td>
<td>-99</td>
<td>There is no implementation available for this method.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_PROPERTY</td>
<td>-100</td>
<td>One or more properties are invalid.</td>
</tr>
<tr>
<td>E_CTIOS_MODE_CONFLICT</td>
<td>-101</td>
<td>A conflict when setting session mode.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_EVENTID</td>
<td>-102</td>
<td>The Event ID is not valid.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_ARGUMENT</td>
<td>-103</td>
<td>The Argument is not valid.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_SESSION</td>
<td>-104</td>
<td>The Session is not valid.</td>
</tr>
<tr>
<td>E_CTIOS_UNEXPECTED</td>
<td>-105</td>
<td>An unexpected error has occurred.</td>
</tr>
<tr>
<td>CIL Error Code</td>
<td>Numeric Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_CTIOS_OBJ_ALLOCATION_FAILED</td>
<td>-106</td>
<td>There is not enough memory available and the creation of CCtiOsObject failed.</td>
</tr>
<tr>
<td>E_CTIOS_ARRAYREF_ALLOCATION_FAILED</td>
<td>-107</td>
<td>There is not enough memory available and the creation of an array of references to objects of type CCtiOsObject failed.</td>
</tr>
<tr>
<td>E_CTIOS_ARGUMENT_ALLOCATION_FAILED</td>
<td>-108</td>
<td>There is not enough memory available and the creation of an object of type Arguments failed.</td>
</tr>
<tr>
<td>E_CTIOS_TARGET_OBJECT_NOT_FOUND</td>
<td>-109</td>
<td>There are no CTIOS Objects capable of processing an incoming event.</td>
</tr>
<tr>
<td>E_CTIOS_PROP_ATTRIBUTES_ACCESS_FAILED</td>
<td>-110</td>
<td>An error occurred while accessing a property's attributes, System may be running out of memory.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_OBJECT_TYPE</td>
<td>-111</td>
<td>The object type is not one of the following predefined types CAgent, CCall, CSkillGroups, or CWaitObject.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_AGENT</td>
<td>-112</td>
<td>No valid agent.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_CALL</td>
<td>-113</td>
<td>No valid call.</td>
</tr>
<tr>
<td>E_CTIOS_IN_FAILOVER</td>
<td>-114</td>
<td>The session is recovering from a connection failure and started the Fail Over procedure.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_DESKTOP_TYPE</td>
<td>-115</td>
<td>Indicates that the desktop type specified in the request for DeskSettings download is neither Agent or Supervisor.</td>
</tr>
<tr>
<td>E_CTIOS_MISSING_ARGUMENT</td>
<td>-116</td>
<td>Missing a required argument.</td>
</tr>
<tr>
<td>E_CTIOS_CALL_NOT_ON_HOLD</td>
<td>-117</td>
<td>Call is not on hold.</td>
</tr>
<tr>
<td>E_CTIOS_CALL_ALREADY_ON_HOLD</td>
<td>-118</td>
<td>Call is already on hold.</td>
</tr>
<tr>
<td>E_CTIOS_CALL_NOT_ALERTING</td>
<td>-119</td>
<td>Call is not in alert state, it can not be answered.</td>
</tr>
<tr>
<td>E_CTIOS_AGENT_NOT_LOGIN</td>
<td>-120</td>
<td>Agent is not logged in.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_METHOD_PARAMETER</td>
<td>-121</td>
<td>The input parameter is invalid.</td>
</tr>
<tr>
<td>E_CTIOS_UNKNOWN</td>
<td>-122</td>
<td>The cause of this error is unknown.</td>
</tr>
<tr>
<td>E_CTIOS_OUT_OF_MEMORY</td>
<td>-123</td>
<td>Failed to allocate new memory.</td>
</tr>
<tr>
<td>E_CTIOS_PORT_UNAVAILABLE</td>
<td>-124</td>
<td>The specified port is not available for use.</td>
</tr>
<tr>
<td>CIL Error Code</td>
<td>Numeric Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_CTIOS_SM_SESSION_TERMINATED_ABNORMALLY</td>
<td>-125</td>
<td>The Silent Monitor session was terminated abnormally.</td>
</tr>
<tr>
<td>E_CTIOS_SM_REJECTED_ALREADY_IN_SESSION</td>
<td>-126</td>
<td>The request was rejected because there is an active silent monitor session in progress.</td>
</tr>
<tr>
<td>E_CTIOS_SM_PACKET_SNIFFER_NOT_INSTALLED</td>
<td>-127</td>
<td>The packet sniffer is not present in the system; verify installation.</td>
</tr>
<tr>
<td>E_CTIOS_PACKET_SNIFFER_FAILED</td>
<td>-128</td>
<td>An error occurred in the packet sniffer.</td>
</tr>
<tr>
<td>E_CTIOS_SOCKET_CALL_FAILED</td>
<td>-129</td>
<td>A CTI OS socket call failed.</td>
</tr>
<tr>
<td>E_CTIOS_MEDIA_TERMINATION_NOT_INSTALLED</td>
<td>-130</td>
<td>EVVBU Media Termination component in the system, verify installation.</td>
</tr>
<tr>
<td>E_CTIOS_MT_UNKNOWN_CODEC</td>
<td>-131</td>
<td>Specified CODEC is not supported.</td>
</tr>
<tr>
<td>E_CTIOS_MEDIA_TERMINATIONFAILED</td>
<td>-132</td>
<td>An error occurred in the Media Termination Packet Decoder.</td>
</tr>
<tr>
<td>E_CTIOS_SNIFFER_NO_PACKETS_RECEIVED</td>
<td>-133</td>
<td>The Sniffer has not received any IP packets.</td>
</tr>
<tr>
<td>E_CTIOS_SNIFFER_FAILED_TO_OPEN_DEVICE</td>
<td>-134</td>
<td>The Sniffer failed to open the networking device.</td>
</tr>
<tr>
<td>E_CTIOS_SNIFFER_FAILED_TO_SET_FILTER</td>
<td>-135</td>
<td>The Sniffer failed when setting the packet filter.</td>
</tr>
<tr>
<td>E_CTIOS_ERROR_IN_PACKET_FILTER</td>
<td>-136</td>
<td>The packet filter expression is incorrect.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_MONITORED_IP_ADDRESS</td>
<td>-137</td>
<td>The IP Address specified for the monitored device (IP Phone) is not valid.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_SNIFFER_OBJECT</td>
<td>-138</td>
<td>Invalid Sniffer object.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_DECODER_OBJECT</td>
<td>-139</td>
<td>Invalid Decoder object.</td>
</tr>
<tr>
<td>E_CTIOS_NO_SM_SESSION_IN_PROGRESS</td>
<td>-140</td>
<td>There are no Silent Monitor Sessions in progress.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_SILENT_MONITOR_SESSION</td>
<td>-141</td>
<td>The specified Silent Monitor session does not exist.</td>
</tr>
<tr>
<td>E_CTIOS_FAILED_REMOVING_SILENT_MONITOR_SESSION</td>
<td>-142</td>
<td>Silent Monitor Session was not removed from the collection.</td>
</tr>
<tr>
<td>E_CTIOS_IP_PHONE_INFORMATION_NOT_AVAILABLE</td>
<td>-143</td>
<td>There is no information available about the IP Phone.</td>
</tr>
<tr>
<td>CIL Error Code</td>
<td>Numeric Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_CTIOS_PEER_NOT_ENABLED_FOR_SILENT_MONITOR</td>
<td>-144</td>
<td>The peer application is not enabled for Silent Monitor.</td>
</tr>
<tr>
<td>E_CTIOS_NOT_ENABLED_FOR_SILENT_MONITOR</td>
<td>-145</td>
<td>This application is not enabled for Silent Monitor.</td>
</tr>
<tr>
<td>E_CTIOS_NO_PENDING_REQUEST</td>
<td>-146</td>
<td>There are no pending requests to be processed.</td>
</tr>
<tr>
<td>E_CTIOS_ALREADY_IN_SESSION</td>
<td>-147</td>
<td>There is already an established session.</td>
</tr>
<tr>
<td>E_CTIOS_MODE_SET_ALREADY</td>
<td>-148</td>
<td>The session mode is already set.</td>
</tr>
<tr>
<td>E_CTIOS_MODE_NOT_SET</td>
<td>-149</td>
<td>The session mode is not set yet.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_OBJECT_STATE</td>
<td>-150</td>
<td>The object is not in the correct state.</td>
</tr>
<tr>
<td>E_CTIOS_INVALID_SILENT_MONITOR_MODE</td>
<td>-151</td>
<td>This error occurs when a request to initiate CTI OS silent monitor is made and CTI OS is configured to use CCM silent monitor. This error also occurs when a request to initiate CCM silent monitor is made and CTI OS is configured to use CTI OS silent monitor.</td>
</tr>
<tr>
<td>E_CTIOS_COM_OBJ_ALLOCATION_FAILED</td>
<td>-200</td>
<td>CoCreateInstance failed to create a COM object wrapper for a CIL Object (Session, Agent, Call, Skill, etc.).</td>
</tr>
<tr>
<td>E_CTIOS_COM_CORRUPTED_REGISTRY</td>
<td>-201</td>
<td>A COM component failed to access data from the registry.</td>
</tr>
<tr>
<td>E_CTIOS_COM_DIALPAD_FAIL_TO_LOAD</td>
<td>-202</td>
<td>The Dial Pad common dialog was not created and CoCreateInstance failed.</td>
</tr>
<tr>
<td>E_CTIOS_COM_CONV_COMPTR_TO_CPPPTR_FAILED</td>
<td>-203</td>
<td>Failed converting COM pointer to C++ pointer.</td>
</tr>
<tr>
<td>E_CTIOS_COM_NOT_INITIALIZED</td>
<td>-204</td>
<td>The MS COM library is not initialized. Invoke CoInitialize(...).</td>
</tr>
<tr>
<td>E_CTIOS_SESSION_DISCONNECT_PENDING</td>
<td>-300</td>
<td>A disconnect is already pending.</td>
</tr>
<tr>
<td>E_CTIOS_SESSION_NOT_CONNECTED</td>
<td>-301</td>
<td>The session is not connected.</td>
</tr>
<tr>
<td>E_CTIOS_SESSION_NOT_DISCONNECTED</td>
<td>-351</td>
<td>The call to Connect failed because the session is not in a disconnected state. The session may be connected or a previous call to Disconnect may not yet be complete.</td>
</tr>
<tr>
<td>CIL Error Code</td>
<td>Numeric Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E_CTIOS_AGENT_ALREADY_IN_SESSION</td>
<td>-900</td>
<td>An object for this agent already exists in the session.</td>
</tr>
<tr>
<td>E_CTIOS_SET_AGENT_SESSION_DISCONNECT_REQUIRED</td>
<td>-901</td>
<td>Session must be disconnected before operation.</td>
</tr>
<tr>
<td>E_CTIOS_SERVICE_SEND_MESSAGE_FAILED</td>
<td>-902</td>
<td>Could not send message. Session may not be connected.</td>
</tr>
<tr>
<td>E_CTIOS_CALL_ALREADY_CURRENT_IN_SESSION</td>
<td>-903</td>
<td>An object for this call is already set as current in the session.</td>
</tr>
<tr>
<td>E_CTIOS_LOGIN_INCONSISTENT_ARGUMENTS</td>
<td>-904</td>
<td>The AgentID and/or PeripheralID provided to a Login call do not match the properties set on the Agent object when SetAgent() was called.</td>
</tr>
</tbody>
</table>

**Note**

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

## COM Error Codes

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

```cpp
// COM Example in C++
int errorCode = 0;
HRESULT hr = pCall->Answer(&errorCode);
if (errorCode==CIL_FAILED)
    printf("An error has occurred while answering the call.")
```

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

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

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

The complete set of HRESULT values is defined by Microsoft in the header file `winerror.h`. The most common HRESULT values that CTI OS developers see are listed in the following table:
<table>
<thead>
<tr>
<th>COM Error Code</th>
<th>Numeric Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_OK</td>
<td>0x00000000</td>
<td>The method succeeded.</td>
</tr>
<tr>
<td>S_FALSE</td>
<td>0x00000001</td>
<td>The method succeeded, but something unusual happened.</td>
</tr>
<tr>
<td>E_FAILED</td>
<td>0x80000008</td>
<td>The method failed.</td>
</tr>
<tr>
<td>REG_DB_E_CLASSNOTREG</td>
<td>0x80040143</td>
<td>The class was not found in the registry. You must run regsvr32.exe on the DLL file to register it.</td>
</tr>
</tbody>
</table>

## Generic Interfaces

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

## Arguments

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

Example of using Arguments in a Visual Basic MakeCall request:

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

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

Java example:

```java
Arguments args = new Arguments();
args.SetValue(CtiOs_IkeywordIDs.CTIOS_DIALEDNUMBER, "12345");
```
**GetValue Method to Access Properties and Parameters**

CTI OS makes extensive use of generic data abstraction. The CTI OS CIL objects, as well as the Arguments structure, store all data by key-value pair. Properties and data values in CTI OS are accessible through a generic mechanism called GetValue. For a list of the different GetValue methods, see CtiOs Object, on page 291 or Helper Classes, on page 441. The GetValue mechanism provides for the retrieval of any data element based on its name. This enables the future enhancement of the data set provided for event parameters and object properties without requiring any interface changes to support new parameters or properties. GetValue supports use of string keywords, as shown in the following examples:

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

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

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

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

**SetValue Method to Set Object Properties and Request Parameters**

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

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

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

```vbnet
Visual Basic
Dim a As Agent
```
UniqueObjectID Variable-Length String

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

The UniqueObjectID is a variable-length string that can uniquely identify the object within the current context of the CTI OS Server and the Unified CCE and CTI Interlink Advanced. The UniqueObjectID comprises an object type (for example, call, agent, skillgroup, and so on), and two or more additional identifying fields. The following table explains the composition of the UniqueObjectID.

Table 8: UniqueObjectID Components

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Sample UniqueObjectID</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call Object</td>
<td>call.5000.202.23901</td>
<td>The Call object is uniquely identified by its PeripheralID (5000, generated by Unified ICM), ConnectionCallID (202, generated by the PBX/ACD), and its ConnectionDeviceID (23901, generated by the PBX/ACD).</td>
</tr>
<tr>
<td>Agent Object</td>
<td>agent.5000.22866</td>
<td>The Agent object is uniquely identified by its PeripheralID (5000, generated by Unified ICM), and its agent ID.</td>
</tr>
<tr>
<td>Device Object (for events only; no CIL object)</td>
<td>device.5000.23901</td>
<td>The device object is uniquely identified by its PeripheralID (5000, generated by Unified ICM), and its instrument number (configured by the PBX/ACD).</td>
</tr>
<tr>
<td>SkillGroup Object</td>
<td>skillgroup.5000.77</td>
<td>The skill group object is uniquely identified by its PeripheralID (5000, generated by Unified ICM), and its SkillGroupNumber (configured by the PBX/ACD).</td>
</tr>
<tr>
<td>Team Object (for events only; no CIL object)</td>
<td>team.5000.5001</td>
<td>The team object is uniquely identified by its PeripheralID (5000, generated by Unified ICM), and its TeamID (5001, also generated by Unified ICM).</td>
</tr>
</tbody>
</table>
The CTI OS UniqueObjectID is not the same as the Unified ICM globally unique 64 bit key used in the ICM historical databases (called the ICMEnterpriseUniqueID), which exists only for calls. The ICMEnterpriseUniqueID stays with the call even when the call is transferred between call center sites, while the UniqueObjectID for a call is specific to its site (by PeripheralID, ConnectionCallID, and ConnectionDeviceID).

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

```
"icm.routercallkeyday.routercallkeycallid"
```

where routercallkeyday is the field Day in the Unified ICM Route_Call_Detail and Termination_Call_Detail tables, and routercallkeycallid is the field RouterCallKey in the Unified ICM Route_Call_Detail and Termination_Call_Detail tables.

The CTI OS server enables certain types of monitor mode applications that track the pre-call notification event (eTranslationRouteEvent or eAgentPreCallEvent) and seeks to match the call data with the arrival of an eCallBeginEvent.

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

- eCallBeginEvent
- eCallDataUpdateEvent
- eSnapshotCallConf
- eCallEndEvent

### UniqueObjectID to Obtain Pointer or Reference

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

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

The GetObjectFromObjectID method performs an AddRef() on the pointer before it is returned to the programmer.
C++ example:

```cpp
string sUniqueObjectID = "call.5000.101.23901";
Ccall * pCall = NULL;
m_pSession->GetObjectFromObjectID(sUniqueObjectID,
    (CCtiOsObject**)&pCall);
pCall->Clear();
pCall->Release(); // release our reference to this object
pCall = NULL;
```

Java example:

```java
String sUID = "call.5000.101.23901";
Call rCall = (Call) m_Session.GetObjectFromObjectID(sUID);
```

### Button Enablement Masks

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

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

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

### Visual Basic.NET Example

```vbnet
Private Sub m_session_OnAgentStateChange(ByVal pIArguments As Cisco.CTIOSCLIENTLib.Arguments) Handles m_session.OnAgentStateChange
    Dim bitmask As Integer
    'Determine the agent's button enablement and update the buttons on the
    form
    bitmask = m_Agent.GetValueInt("EnablementMask")
    btnReady.Enabled = False
    btnNotReady.Enabled = False
    btnLogout.Enabled = False
    btnStartMonitoring.Enabled = False
```
If bitmask And Cisco.CTIOSCLIENTLib.enumCTIOS_EnablementMasks.ENABLE_READY Then
    btnReady.Enabled = True
End If
If bitmask And Cisco.CTIOSCLIENTLib.enumCTIOS_EnablementMasks.ENABLE_NOTREADY Then
    btnNotReady.Enabled = True
End If
If bitmask And Cisco.CTIOSCLIENTLib.enumCTIOS_EnablementMasks.ENABLE_NOTREADY_WITH_REASON Then
    btnNotReady.Enabled = True
End If
If bitmask And Cisco.CTIOSCLIENTLib.enumCTIOS_EnablementMasks.ENABLE_LOGOUT Then
    btnLogout.Enabled = True
End If
If bitmask And Cisco.CTIOSCLIENTLib.enumCTIOS_EnablementMasks.ENABLE_LOGOUT_WITH_REASON Then
    btnLogout.Enabled = True
End If
End Sub

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

Note

- You must use the button enablement mask generated by CTI Server in all cases where Cisco provides button enablement masks. This prevents application impact if changes are made to the event flow.
- Cisco makes no guarantees that the event flow will remain consistent across versions of software.

Warning

The button enablement feature is intended for use in agent mode applications and not for monitor mode applications.

For any given event, the CTI Server calculates the appropriate button enablement bitmask and sends it to the CIL with the event parameters. The button enablement bit masks are discussed in detail in Event Interfaces and Events, on page 167 You can use these masks to write a custom softphone-type application without writing custom code to enable and disable buttons. This approach is also used internally for the CTI ActiveX softphone controls.
CHAPTER 4

Building Your Custom CTI Application

- System Requirements for Building Custom Applications, on page 33
- Environment Set Up for .NET, on page 34
- Integration Between Your Application and CTI OS via CIL, on page 35
- CTI Application Testing, on page 37
- Developer Sample Applications, on page 38
- CTI OS ActiveX Controls, on page 39
- COM CIL in Visual Studio, on page 43
- C++ CIL and Static Libraries, on page 46
- Java CIL Libraries, on page 52
- .NET CIL Libraries, on page 53
- CTI OS Server Connection, on page 53
- Agent Login and Logout, on page 64
- Calls, on page 72
- Making Requests, on page 76
- Events, on page 77
- Agent Statistics, on page 78
- Skill Group Statistics, on page 83
- Silent Monitoring, on page 87
- Agent Greeting, on page 96
- Deployment of Custom CTI OS Applications, on page 96
- Supervisor Applications, on page 102
- Sample Code in CTI OS Toolkit, on page 115

System Requirements for Building Custom Applications

Use the following development tools for building custom applications:

<table>
<thead>
<tr>
<th>Development Tool</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Visual Studio</td>
<td>2015</td>
</tr>
<tr>
<td>.NET Framework</td>
<td>4.7.1</td>
</tr>
<tr>
<td>Java</td>
<td>1.8 Update 161</td>
</tr>
</tbody>
</table>
Environment Set Up for .NET

The Cisco CTI OS Toolkit introduces support for application development using Microsoft Visual Studio and .NET framework. You need not modify the existing .NET framework controls to run .NET CLR. Cisco CTI OS Toolkit provides a native .NET class library (.NET CIL) and runtime callable wrappers (RCWs) for COM CIL and the CTI OS ActiveX controls. The CTI OS Toolkit consists of a set of production ready desktops and five software development kits.

The setup program installs the .NET CIL and the RCWs in the Global Assembly Cache (GAC) making all the components available to the sample included in the toolkit and any new application in development. Use the CTI OS toolkit for environment settings for building .NET applications. Additional configuration steps are available for integration with the development environment.

The Production Ready Contact Center Desktop applications are the CTI OS Toolkit Agent Desktop, CTI OS Toolkit IPCC Supervisor Desktop, the CTI OS Toolkit Outbound Option Desktop, and the default client desktops for Cisco CTI OS used by call center agents and supervisors. These desktop applications are built using the COM CIL and the CTI OS ActiveX controls. These applications are implemented using Visual Basic .NET (VB.NET) and Microsoft Visual Studio.

Microsoft Visual Studio

Microsoft Visual Studio 2015 offers a wider spectrum of development possibilities and an advanced design experience. In addition to Service Pack 3 it also provides:

• Microsoft .NET Framework 4.7.1 application development
• New processor support (for example, Core Duo) for code generation and profiling
• Additional support for project file based Web applications
• Secure C++ application development

To access the .NET CIL and the RCWs directly from Visual Studio, add the following configuration to your environment.

Add CTI OS Toolkit Components to Add Reference Dialog Box

In Microsoft Visual Studio, you can select class libraries and assemblies from the .NET tab of the Add Reference dialog box. This facilitates the development process and ensures you can always use the correct version of the components.

To enable the .NET CIL class libraries to appear on the Add References dialog box, follow the steps described in https://msdn.microsoft.com/en-us/library/wkze6zky(v=VS.100).aspx.

Set a registry key that specifies the location of assemblies to appear.

To do this, add one of the following registry keys, where <AssemblyLocation> is the directory of the assemblies that you want to appear in the Add Reference dialog box:

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\<version>\AssemblyFoldersEx\MyAssemblies]@="<AssemblyLocation>"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\<version>\AssemblyFoldersEx\MyAssemblies]@="<AssemblyLocation>"
Creating the registry key under the HKEY_LOCAL_MACHINE node allows all users to see the assemblies in the specified location in the Add Reference dialog box. Creating the registry key under the HKEY_CURRENT_USER node affects only the setting for the current user.

For example, if you want to add:

- Cisco .NET CIL to the Add Reference dialog box
  
  ```
  [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\$.NETFramework\<version>\AssemblyFoldersEx\MyAssemblies]@="<AssemblyLocation>"
  ```

- Cisco CTI OS RCWs to the Add Reference dialog box
  
  ```
  [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\$.NETFramework\v2.0.50727\AssemblyFoldersEx\CiscoCTIOSRCWs]@="C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32CIL\$.NETInterops"
  ```

### Add Cisco CTI OS ActiveX Controls to Toolbox

The Microsoft Visual Studio IDE allows visual editing of Windows Forms based applications using the toolbox of available visual components. Because Windows Forms applications are native, the visual components are also native. You can still use ActiveX controls and include them in the toolbox.

Adding CTI OS ActiveX controls to the toolbox provides pre-packaged CTI functionality such as Agent Login, Make Call, Transfer Call, Barge In, and so on. The ActiveX controls use COM CIL as the API to provide call center and telephony services. These components are used in rapid software development. You can drag and drop selected components into your project and immediately gain the selected CTI functionality. These components are used in development environments such as: Microsoft Visual Studio, .Net Framework, and Java.

To use the Cisco CTI OS ActiveX controls in Microsoft Visual Studio, you must configure the Cisco CTI OS RCWs:

**Procedure**

**Step 1**
From Visual Studio's View menu, choose Add/Remove Toolbox Items.

**Step 2**
From the Customize Toolbox dialog box, select the .NET Framework Components tab.

**Warning**
If you click the COM Components tab from the Customize Toolbox dialog box or select the CTI OS ActiveX controls, you cause Microsoft Visual Studio to automatically generate a set of private RCWs that are not optimized and approved by Cisco, which can result in application failure.

**Step 3**
From the list, select the CTI OS RCW that corresponds to the CTI OS ActiveX Control you want to add to the toolbox. For example, for the Agent State Control select AxAgentStateCtl.

**Step 4**
To add more CTI OS ActiveX controls, repeat steps 1 to 3.

### Integration Between Your Application and CTI OS via CIL

Creating an integration between your application and CTI OS via the CIL is straightforward. The first step is to articulate the desired behavior, and to create a complete design specification for the integration.
Integration Planning and Design

Good design depends on understanding how CTI fits into your application and workflow. Your requirements analysis and design process should address the following points, as they relate to your specific application:

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

• **Study agent workflow.** What are the points where CTI can make the workflow easier and faster? Build a business case for the CTI integration.

• **Evaluate what CTI will do for your application.** A good approach is to make a list based on the priority (for example, screen pop, then call control) and then design and implement features in that order.

• **Design how CTI should work within your application.** What are the interaction points? Get specifications as to which screens interact, and which data values should be sent between your application and the CTI OS platform.

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

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

Language and Interface

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

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

• **ActiveX Controls.** The CTI OS ActiveX controls are the appropriate choice for creating a rapid drag and drop integration of CTI and third-party call control with an existing desktop application. The CTI OS ActiveX controls are an appropriate choice for developing a CTI integration with any fully ActiveX-compliant container, or any other container that fully supports ActiveX features (for example, Powerbuilder, Delphi, and many third-party CRM packages). The ActiveX controls are the easiest to implement in graphical environments, and help achieve the fastest integrations by providing a complete user interface. All CTI OS ActiveX components are distributed via dynamic link library files (.dll), which you only have to register once to work on any Microsoft Windows platform. These components are not appropriate for non-Windows environments. You can use the CTI OS ActiveX controls in Windows Forms .NET applications only if the Runtime Callable Wrappers (RCWs) provided with the CTI OS Toolkit are a part of the project. For more information, see CTI OS ActiveX Controls, on page 39.

• **COM.** The CTI OS Client Interface Library for Microsoft's Component Object Model (COM) is the appropriate choice for developing a CTI integration with any COM-compliant container, or any other container that supports COM features, such as Microsoft Internet Explorer or Visual Basic for Applications scripting languages. The COM CIL is the easiest to implement in scripting environments, and helps
achieve the fastest integrations requiring a custom or non-graphical user interface. All CTI OS components are distributed via dynamic link library files (.dll), which you only have to register once to work on any Microsoft Windows platform. These components are not appropriate for non-Windows environments. You can use the COM CIL in Windows Forms .NET applications only if the Runtime Callable Wrappers (RCWs) provided with the CTI OS Toolkit are a part of the project. For more information, see Hook for Screenpops, on page 41.

• **C++**. The CTI OS Client Interface Library for C++ is the appropriate choice for building a high-performance application running on a Windows platform in a C++ development environment. The C++ CIL is distributed as a set of header files (.h) that specify the class interfaces to use and statically linked libraries (.lib) that contain the compiled implementation code.

• **Java**. The CTI OS Java Client Interface Library (Java CIL) is an appropriate choice for non-Microsoft (typically UNIX) operating systems, as well as for browser based applications.

• **.NET CIL class libraries**. This section covers the steps required to reference the .NET CIL components in a C# and Visual Basic .NET project files.

**CTI Application Testing**

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

**Test Plan Development**

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

Perform (at minimum) the following test phases:

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

• **Integration Testing**. In an integration test, you ensure that the new components work together properly. The physical connections and data passing between the layers and servers involved in the system are tested. An example of an integration test is testing your client application with the CTI OS server, to ensure that you can pass data correctly through the components.

• **System Testing**. In a system test, you ensure that the correct application behavior is exhibited. An example of a system test is to make a phone call to a VRU, collect the appropriate caller information, transfer the call to an agent, and ensure that the screen pop arrives correctly.

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

The CTI OS Software Development Toolkit (SDK) CD media includes a CTIServerSimulator that you can use for application development and demonstration purposes. It can roughly simulate a Lucent PBX/ACD or a Cisco Unified Contact Center environment. Documentation on how to configure and use the simulator is available in the Tools\Simulator directory.

Note

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

Developer Sample Applications

The CTI OS Software Development Toolkit (SDK) is distributed with a rich set of Developer Sample Applications (DSAs) for Cisco Unified CCE customers and similar Production Class Applications for Unified ICM customers.

The DSAs are provided as tools for Unified CCE customers to accelerate development efforts. The DSAs demonstrate several basic working applications that use varying implementations of the CTI OS Client Interface Library API. The samples are organized by programming language and demonstrate the syntax and usage of the API. For many developers, these DSAs form the foundation of your custom application. The samples are available for you to customize and distribute as a part of your finished product.

For Unified ICM ACD types (such as Avaya, Aspect, and so on), you can deploy some DSAs as Production Class Applications. Cisco certifies and supports the out-of-the-box CTI OS Agent Desktop application in a production environment when used in conjunction with a supported Unified ICM ACD. Refer to the ACD Supplement, Cisco Unified Intelligent Contact Management (Unified ICM) ACD PG Supportability Matrices for the current list of supported ACD types.

For Unified CCE, these same DSAs are generally not intended for production use "as-is". They are neither certified nor supported by Cisco as working out-of-the-box applications.

The following table lists the sample programs in the CTI OS Toolkit.

Table 9: CTI OS Toolkit Sample Programs

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI Toolkit</td>
<td>CTIOS Toolkit\Win32 CIL\Samples\CTI Toolkit Outbound Desktop</td>
<td>A softphone application that demonstrates Outbound Option (formerly Blended Agent) functionality.</td>
</tr>
<tr>
<td>Outbound Desktop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All Agents Sample</td>
<td>CTIOS Toolkit\dotNetCIL\Samples\All Agents Sample.NET</td>
<td>A Microsoft C# program demonstrating a monitor mode application. This program lists all agents in a grid along with current state updates.</td>
</tr>
<tr>
<td><strong>Program Name</strong></td>
<td><strong>Location</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>All Calls Sample.NET</td>
<td>CTIOS Toolkit\dotNetCIL\Samples\All Calls Sample.NET</td>
<td>Similar to AllAgents but lists calls instead of agents.</td>
</tr>
<tr>
<td>CTI Toolkit Combo Desktop.NET</td>
<td>CTIOS Toolkit\dotNetCIL\Samples\CTI Toolkit Combo Desktop.NET</td>
<td>A Microsoft C# program that interfaces to CTI OS via the .NET CIL interface. The program demonstrates how to build a multi-functional contact center desktop that contains Agent, Supervisor and Outbound Option features.</td>
</tr>
<tr>
<td>CtiOs Data Grid.NET</td>
<td>CTIOS Toolkit\dotNetCIL\Samples\CtiOs Data Grid.NET</td>
<td>Microsoft C# program that implements a Tabular Grid used by the CTI Toolkit Combo Desktop.NET to show calls and statistics.</td>
</tr>
<tr>
<td>CTI Toolkit Agent Desktop</td>
<td>CTTIOS Toolkit\Win32 CIL\Samples\CTI Toolkit AgentDesktop</td>
<td>A Visual Basic .NET program using the CTI OS ActiveX controls. The application is the source code used by the out of the box CTI Toolkit Agent Desktop.</td>
</tr>
<tr>
<td>CTI Toolkit Supervisor Desktop</td>
<td>CTTIOS Toolkit\Win32 CIL\Samples\CTI Toolkit SupervisorDesktop</td>
<td>A Visual Basic .NET program using the CTI OS ActiveX controls. The application is the source code used by the out of the box CTI Toolkit Supervisor Desktop.</td>
</tr>
<tr>
<td>C++Phone</td>
<td>CTIOS Toolkit\Win32 CIL\Samples\CTI Toolkit C++Phone</td>
<td>A softphone written in C++ linking to the static C++ libraries. Sending requests and event handling as well as the use of the wait object is demonstrated.</td>
</tr>
<tr>
<td>JavaPhone</td>
<td>CTIOS Toolkit\Java CIL samples</td>
<td>A Java counterpart to the C++phone sample programs.</td>
</tr>
<tr>
<td>AllAgents</td>
<td>CTIOS Toolkit\Java CIL samples</td>
<td>A Java counterpart to the Visual Basic all agents program.</td>
</tr>
</tbody>
</table>

**CTI OS ActiveX Controls**

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

**Build Simple Softphone with ActiveX Controls**

To use the CTI OS ActiveX controls, you must copy the ActiveX controls on the target system and register with Windows. You accomplish this with the CTI OS toolkit install, as well as the CTI OS Agent and Supervisor installs. For more information, see [Deployment of Custom CTI OS Applications](#), on page 96.

After you launch Visual Basic .NET, you can use the ActiveX controls by selecting them via the Customized Toolbox dialog (Tools > Add/Remove Toolbox Items via the menu).
Note: If the CTI OS ActiveX controls are not listed as shown in the following figure the files are either not copied on the target system or the controls were not properly registered.

Figure 7: Customize Toolbox in Visual Basic .Net Listing CTI OS ActiveX Controls Runtime Callable Wrappers

After you select the CTI OS ActiveX controls in the .NET Framework Components tab, you should see them in the Visual Basic .NET ToolBox. You can now drag and drop the CTI OS ActiveX RCWs components onto the Windows Form. For a softphone application, it is useful to start with the CallAppearanceCtl (see the following figure).
On the very left, the Toolbox is visible showing some of the CTI OS ActiveX RCWs icons. On the form, the AxCallGrid has been dragged and dropped.

For a complete description of the ActiveX controls see CTI OS ActiveX Controls, on page 119. The following figure shows the CTI OS Toolkit Agent Desktop application, which is also included as a sample on the CTI OS CD.

Figure 9: CTI OS Toolkit Agent Desktop (See CD) Built with CTI OS ActiveX Controls

Once all ActiveX controls are placed on the phone, you can create an executable in Visual Basic .NET via Build > Build Solution or selecting <F7>.

Hook for Screenpops

This agent desktop application did not require any Visual Basic .NET coding. You can choose to add some custom code to add a hook for screenpops. For example, you may want to retrieve CallVariables, which are passed along with certain call events.
CTI OS SessionResolver

A CTI OS Client application connects to CTI OS with a Session object (see Session Object, on page 307). Depending on the application, a client can use one or more Session objects. For most agent desktop applications, however, it is useful to employ only a single Session object.

If you choose to write a program not using ActiveX controls, you can create a Session object and use it directly (see CTI Toolkit AgentDesktop at the Win32 CIL samples).

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

VB .NET Code Sample to Retrieve Common Session

The following sample VB .NET code retrieves the common session and listens for a CallEstablishedEvent occurring in that session. If a CallEstablishedEvent occurs, it retrieves CallVariable 1 and puts it in the Windows Clipboard (from where you can retrieve it via CTRL-v or used by other applications).

This code uses the COM CIL Interfaces and therefore, needs the following references: Cisco.CTIOSCLIENTLib, Cisco.CTIOSARGUMENTSLib, Cisco.CTIOSSESSIONRESOLVERLib. The references are shown in the following figure (in Visual Basic .NET, select Project > Add Reference...).

Figure 10: CTI OS COM CIL RCWs References Needed for Visual Basic .NET COM Programming

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

Private Sub Form_Initialize_Renamed()
' instantiate the sessionresolver
Set m_sessionresolver = New Cisco.CTIOSSESSIONRESOLVERLib.SessionResolver

' CTI OS ActiveX controls use the session named "" - blank
' since the CTI OS ActiveX controls do the connection and login,
' all we do is listen for events
Set m_session = m_sessionresolver.GetSession(""")
End Sub

Private Sub Form_Terminate_Renamed()
Call m_sessionresolver.RemoveSession(""")
End Sub

Private Sub m_Session_OnCallEstablished(ByVal pIArguments
As Cisco.CTIOSCLIENTLib.Arguments)
' Handles m_Session.OnCallEstablished
   GetCallVariable1 pIArguments
End Sub

Function GetCallVariable1(ByVal pIArguments As CTIOSCLIENTLib.IArguments)

    Dim m_uid As String
    m_uid = pIArguments.GetValueString("Uniqueobjectid")
    Dim m_call As Cisco.CTIOSCLIENTLib.Call
    Set m_call = m_session.GetObjectFromObjectID(m_uid)

    ' retrieve callvar1
    Dim m_callvar1 As String
    m_callvar1 = m_call.GetValueString("Callvariable1")

    ' copy call variable1 to the clipboard
    Clipboard.SetText m_callvar1
End Function

---

**Note**
Visual Basic 6.0 is no longer supported.

---

**COM CIL. in Visual Studio**

**COM CIL.**

Use this API in development environments that support COM/DCOM and OLE Automation. Examples: Microsoft Visual Studio, Borland Delphi, Power Builder, etc. COM CIL is an adaptor interface that uses C++ CIL as kernel. The API is deployed as a group of Dynamic Linked Libraries (DLLs).

**Note**
You must use Microsoft Visual Studio to build C++ applications using COM CIL. Applications using COM CIL built with Visual C++ 8.0(1) are not supported.
Building Your Custom CTI Application
Add COM Support to Your Application

For building a custom Win32 (Console or Windows) CTI application in Microsoft Visual Studio with COM,
you must create COM components in Microsoft Visual Studio.
Client applications of this type are more complex to build, and more powerful and faster in execution, than
scripting clients (for example, Visual Basic). CIL components for COM are distributed as COM Dynamic
Link Libraries (COM DLL).
COM components must be registered with Windows to be accessible to COM containers including Micsosoft
Visual Studio. The components required for programming in Microsoft Visual Studio are:
• CTI OS Client library (CTIOSClient.dll). This is the main CIL library for COM. The objects available
in this library are described in Chapters 8 through 11.
• CTI OS Arguments Library (arguments.dll). The Arguments helper class is used extensively in CTI
OS, and is described in Helper Classes, on page 441.
• CTI OS Session Resolver (ctiossessionresolver.dll). This object allows multiple applications or controls
to use a single CTI OS Session object. You require this object when building an application that includes
the CTI OS ActiveX controls.

Add COM Support to Your Application
Your application must support COM to use these objects in your CTI application. To add COM support to
your application, you must use one of the following:
• Microsoft Foundation Classes (MFC). The following header files are required for MFC applications to
use COM: afxwin.h, afxext.h, afxdisp.h, and afxdtctl.h. If you build an application using the Microsoft
Visual C++ application wizard, these files are included automatically.
• Microsoft's ActiveX Template Library (ATL). To use ATL, include the standard COM header file:
atlbase.h.

Important Note About COM Method Syntax
In this manual, the syntax used to describe method calls in COM shows standard COM data types such as
BSTR, VARIANT and SAFEARRAY. Be aware that these data types can be encapsulated by wrapper classes
proper to the environment depending on the development environment, tools, and how the COM CIL is
included in your project application.
For example, in a Microsoft Visual C++ project a VARIANT type can be either a CComVariant or _variant_t,
and a BSTR type can be either a CComBSTR or _bstr_t.
For more information, see the documentation for your development environment.

Use CIL Dynamic Link Libraries
Next, you must import the COM Dynamic Link Libraries into your C++ application. The following code
sample (which you might put into your StdAfx.h file) depicts how to use a COM Dynamic Link Library in
C++:
#import "..\..\Distribution\COM\ctiossessionresolver.dll" using namespace
CTIOSSESSIONRESOLVERLib;
#import "..\..\Distribution\COM\ctiosclient.dll" using namespace CTIOSCLIENTLib;

CTI OS Developer Guide for Cisco Unified ICM, Release 12.0(1)
44


You must register three DLLs, but you do not need to import the arguments.dll into your project because it is imported by the ctiosclient.dll type library.

Create COM Object at Run Time

Only the apartment threading model is supported.

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

```cpp
// Create SessionResolver and Session object
hRes = m_pSessionResolver.CreateInstance(OLESTR("CTIOSSessionResolver.SessionResolver"));
if (m_pSessionResolver)
{
    m_pSession = m_pSessionResolver->GetSession(_bstr_t(""));
}
```

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

COM Events in C++

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

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

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

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

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

```cpp
// Add event sink as event listener for the _IallEvents interface
HRESULT hRes = AtlAdvise(m_pSession, m_EventSink.GetIDispatch(FALSE), __uuidof(_IAllEvents), &m_dwEventSinkAdvise);
```
When the program run is complete, the client must unsubscribe from the event source, using the AtlUnadvise() method:

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

### Additional Information

- For more information about the CTI OS client startup and shut down sequence, see section Disconnect from CTI OS Server Before Shutdown, on page 63.
- For more information about the CTI OS Client Interface Library objects, see Chapters 8 through 12.

The C++ Client Interface Library (C++ CIL) application is a programming interface (API) you can use to build high performance CTI enabled desktop or server-to-server integration that use Cisco CTI OS. The API is deployed as a set of C++ static libraries that you can use to build Win 32 or console based applications.

- For more information about a sample application that uses the CIL COM interface written in C++, see the Comphone sample application on the CTI OS CD.

### C++ CIL and Static Libraries

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

The CIL interface for C++ is a set of C++ header files (.h), and static libraries compiled for the Win32 platform (Windows 2010). The header files required to access the class definitions are located on the CTI OS SDK media in the CTIOSToolkit\Include\ directory, and the static libraries are located in the CTI OS Toolkit\Win32\CIL\Libs directory.

---

**Note**

Use Microsoft Visual Studio to build C++ applications using C++ CIL Visual Studio. Applications that are built using Visual Studio Enterprise 2015 are supported.

### Header Files and Libraries

The header files you most likely require are all included in the main CIL header file, CIL.h, which you would include in your application:

```c
#include <Cil.h>
```

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

- **ConnectionLibSpd.lib.** This library contains the connection-layer services for CIL.
- **ServiceLibSpd.lib.** This library contains the service-layer services for CIL.
• **SessionLib.lib**. This library contains the object-interface services for CIL.

• **UtilLibSpd.lib**. This library contains helper classes for CIL.

• **ArgumentsLibSpd.lib**. This library contains the Arguments data structure for CIL.

• **SilentMonitorLib.lib**. This library contains all the services required to establish and control silent monitor sessions.

• **SecuritySpd.Lib**. This library contains the services required to establish secure connections with CTI OSServer.

• **SilentMonitorClient.lib**. This library is used by the CIL to communicate with the silent monitor service.

• **SilentMonitorCommon.lib** and **ServiceEventHandler.lib**. These libraries contain support classes for SilentMonitorClient.lib.

---

**Note**

The preceding are the release versions of the libraries. The Debug equivalent libraries use the same library name with the appended “d” instead of Spd; for example, for ArgumentsLibSpd, the Debug library is ArgumentsLibd.lib.

In addition to the aforementioned CTI OS CIL libraries, your application requires:

• The standard Microsoft sockets library, Wsock32.lib

• The standard multimedia library, winmm.lib

• The OpenSSL standard libraries:
  - libeay32d.lib
  - ssleay32d.lib (Debug) and libeay32d.lib
  - ssleay32r.lib (Release)

A console C++ application with C++ CIL needs to use the following in stdafx.h:

```cpp
#pragma once
#define WIN32_LEAN_AND_MEAN // Exclude rarely-used stuff from Windows headers
#include <iostream>
#include <tchar.h>
```

Use the following libraries in linker in addition to the CIL libraries:

• ws2_32.lib

• Winmm.lib

• odbc32.lib

• odbc32.lib

---

**Configure Project Settings for Compiling and Linking**

You must configure some program settings to set up your Visual C++ application.
Procedure

**Step 1**  
You access the Program Setting in Visual C++ under the **Project > Properties** menu.

**Step 2**  
In the **Property Pages** dialog box, under **C/C++**, select **General** and then select the **Additional Include Directories**. Provide either the absolute or relative path to find the header files (.h) required for your application. This path points to the CTIOSToolkit\Win32CIL\Include directory, where the CIL header files are installed.

**Step 3**  
Next, under **C/C++**, select **Code Generation**. For a Debug Mode program, the setting for Runtime Library is Multi-threaded Debug DLL (/MDd). For a Release Mode program, the setting is Multi-threaded DLL (/MD).
Step 4  Under **Preprocessor**, set the **Preprocessor Definitions**. You need to provide the compiler with the following define constants: _USE_NUMERIC_KEYWORDS=0; _WIN32_WINNT=0x0500; _WIN32_LEAN_AND_MEAN in addition to the defaults.

Step 5  In the Preprocessor Definitions for the C++ compiler, add these macros:

-  _CRT_SECURE_CPP_OVERLOAD_STANDARD_NAMES=1
-  _CRT_SECURE_CPP_OVERLOAD_STANDARD_NAMES_COUNT=1
Step 6 In the Language settings for the C++ compiler, set the parameter "Treat wchar_t as Built-in Type" to No (/Zc:wchar_t-).

Step 7 For the Precompile Headers for the C++ compiler, set to Not Using Precompile Headers.

Step 8 Under Linker, set the link settings for your project. You must list all the static libraries for your program to link with the settings described in Configure Project Settings for Compiling and Linking. The libraries required for CIL (in addition to the default libraries) are described in Header Files and Libraries, on page 46.

Step 9 Finally, expand Link, select General. Set the Additional Library Directories: to the location of the CTIOSToolkit\Win32 CIL\Libs directory.
Step 10

After specifying all the Project Settings required for CTI OS, click OK, and save your project settings.

Event Subscription in C++

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

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

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

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

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

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

The Cisco CTI OS Toolkit no longer uses STLPort. The toolkit now uses Microsoft's version of STL, which removes any special configuration of the build environment.

Additional Information

- For more information about the CTI OS client start up and shut down sequence, see the section Disconnect from CTI OS Server Before Shutdown, on page 63.
- For more information about the CTI OS Client Interface Library objects, see Chapters 6 through 11.
- For a complete sample application that uses the CIL interface with C++ static libraries, see the C++ phone sample application on the CTI OS CD.

Java CIL Libraries

The Java CIL provides a powerful cross-platform library for developing Java CTI applications. This Java API allows the creation of multiplatform client application that you can execute either in MS Windows or Linux. JavaTM CIL is built to support the 1.8 Java Development Kit (JDK) and JRE. It is built using a similar architecture to the C++ CIL. The interface is similar to C++ with minor differences. A developer porting a C++ CIL application to Java or working between a Java and C++ should find it easy to switch between the two.

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

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

The Java CIL ships with a GUI TestPhone application that provides most of the functionality found on the CTI OS Agent and Supervisor Desktops. The distribution also includes samples that are Java versions of some of the C++/COM/VB sample applications. For more information, see Developer Sample Applications, on page 38.

The CTI OS Java Test Phone was updated and compiled with CTI OS Java CIL 8.0(1) using the JDK/JRE 1.6_01 for Linux and was functionally tested on Red Hat Linux Enterprise.

Additional Information

- For more information about differences between the C++ and Java event publishing, see Event Interfaces and Events, on page 167 and Keywords, on page 499.
- For more information about differences in method calls and syntax for those classes between C++ and Java, see CtiOs Object, on page 291.
• For more information about differences between C++ and Java tracing, see CTI OS Client Logs (COM and C++), on page 503.

**.NET CIL Libraries**

The .NET CIL provides native .NET class libraries for developing native .NET Framework applications. It is built using the same architecture as the Java CIL and the interface is similar to C++. A developer porting a C++ CIL application to .NET CIL between a .NET and Win32 should find it easy to switch between the two. The .NET Client Interface Library (.NET CIL) API provides native support for the Microsoft .NET Framework Common Language Runtime 4.7 (CLR). You can use the API with all major .NET Programming languages (C#, VB.NET, Managed C++, ASP.NET, etc). The API is deployed as .NET Assemblies that are registered in the system Global Assembly Cache (GAC).

The .NET CIL consists of two class libraries: NetCil.dll and NetUtil.dll that must be added as references on the build project. See the CTIOSToolkitComboDesktop sample.

To deploy the client application, use the Global Assembly Cache Tool (gacutil.exe) that is included with Microsoft Visual Studio. Use the Microsoft .NET Framework 4.7 Configuration Manager to install the NetCil.dll and NetUtil.dll class libraries on the host Global Assembly Cache (GAC). The .NET CIL libraries include sample programs that explain how to use APIs in a .NET programming environment. For more information, see Developer Sample Applications, on page 38.

---

**Note**

In addition to NetCil.dll and NetUtil.dll, the .NET Combo sample requires the CTIOSVideoCtl.dll, which is in: \C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\dotNet CIL\Controls.

---

**Additional Information**

• For more information about the differences between the C++, and .NET and Java event publishing, see Event Interfaces and Events, on page 167 and CTI OS Client Logs (COM and C++), on page 503.

• For more information about the differences in method calls and syntax for those classes between C++ and Java, see CtiOs Object, on page 291.

---

**CTI OS Server Connection**

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

1. Create a session instance, described below.
2. Set the event listener and subscribe to events, described below.
3. Set connection parameters, described below.
4. Call the Connect() method, described on Connect Session to CTI OS Server, on page 55.
5. Set the connection mode, described on Connection Mode, on page 57. This section also describes how to deal with connection failures, on Connection Failures, on page 55.
Although the Cisco Security Agent (CSA) is now in end-of-life status and no longer supported, if your system is a duplexed Unified CCE PG with a CSA installed and one side of the CTIO server is not running, CSA does not respond to login requests on the CTIO server port. This triggers a time-out (20 second delay) before you attempt to connect to the active CTIOS server in the CTIO client machine TCP stack. On start-up or login, the CTIO client randomly chooses a CTIO server side to connect and it may connect to the server side that is not running.

To avoid this delay/time-out, you must:

• Start the inactive CTIO server side.
• Disable CSA (temporarily) and reconfigure the CTIO desktop for a simplex operation.
• Upgrade the version of the CTIO server to CTIO 12.0 (the desktop does not appear frozen though the delay persists).

**Connect to CTIO Server**

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

The following line shows this in **Java**:

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

**Session Object Lifetime (C++ Only)**

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

For example:

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

The client application holds a reference to the Session object as long as it is in use, but the client programmer must release the last reference to the object to prevent a memory leak when the object is no longer needed.

During application cleanup, you must dispose the Session object only by invoking the CCtiOsSession::Release() method. This ensures proper memory cleanup.

For example:

```c++
m_pSession->Release();
```

**Set Event Listener and Subscribe to Events**

Before making any method calls with the Session instance, you must set the session as an event listener for the desktop application and subscribe to events.

The following lines show this in **Java**:

```
rSession.AddEventListener(this, CtiOs_Enums.SubscriberList.eAllInOneList);
```

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

To set connection parameters:

Procedure

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>Create an instance of the Arguments class.</td>
</tr>
<tr>
<td>Step 2</td>
<td>Set values for the CTI OS servers, ports, and the heartbeat value.</td>
</tr>
</tbody>
</table>

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

The following example demonstrates this task in **Java**:

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

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

**Note** The Arguments.setValue() methods return a boolean value to indicate whether the method succeeded (true) or not (false).

Connect Session to CTI OS Server

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

The following line shows this in **Java**:

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

For more information about the possible values and meanings of the int value returned by the Connect() method in the Java CIL, see Connection Attempt Error Codes in Java and .NET CIL, on page 56.

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

Connection Failures

This section contains the following information:
Connection Failure Events

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

- OnConnectionRejected() event indicates that an unsupported version mismatch was found.
- OnCTIOSFailure() indicates that the CTIOSServer requested in the Connect() method is down. If an OnConnectionFailure() event is generated, the application is in Failover and the CIL continues to attempt to connect until the connection succeeds or until the application calls Disconnect(). The Arguments parameter for the event includes the following keywords:
  - FailureCode
  - SystemEventID
  - SystemEventArg1
  - ErrorMessage

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

Connection Attempt Error Codes in Java and .NET CIL

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

- CIL_OK - The connection process has successfully begun. The CIL either fires the OnConnection() event to indicate that the CIL successfully connected or fires the OnConnectionFailure() event and goes into failover mode. If the latter occurs, the CIL continues to attempt to connect, alternating between hosts CTIOS_CTIOSA and CTIOS_CTIOSB, until the connection succeeds, at which point the CIL fires the OnConnection() event.
- E_CTIOS_INVALID_ARGUMENT - A null Arguments parameter was passed to the Connect() method. The connection failed. No events are fired.
- E_CTIOS_MISSING_ARGUMENT - The Arguments parameter did not contain values for both CTIOS_CTIOSA and CTIOS_CTIOSB. At least one of these values must be provided. The connection failed. No events are fired.
- E_CTIOS_IN_FAILOVER - A previous connection attempt failed and the CIL is currently in failover and attempting to establish a connection. This continues until a connection is established, at which point the CIL fires an OnConnection() event indicating that the previous Connect() method succeeded. To attempt to connect again with different parameters, the application must first use the Disconnect() method.
- E_CTIOS_SESSION_NOT_DISCONNECTED - The Session is not disconnected (i.e. a previous Connect() method is in progress, or the Session is already connected). The application must call the Disconnect() method before attempting to establish another connection. The CIL may fire an
OnConnection() event for the to previous call to the Connect() method if the connection was in progress, but will not fire one corresponding to this method call.

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

---

**Note**

After the application receives a Connect return code of CIL_OK, it does not call Connect again on that session until it receives an OnConnectionClosed event after a call to Disconnect.

---

**Configure Agent to Automatically Log In After Failover**

If you are using CTI OS in an Unified Contact Center Enterprise (Unified CCE) environment, you can configure the agent to automatically relogin in the event of a failover.

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

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

For more information on logging in an agent, see Log In an Agent, on page 64.

---

**Stop Failover Procedure**

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

---

**Connection Mode**

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

- Agent mode
- Monitor mode

---

**Set Connection Mode in OnConnection() Event Handler**

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

---

**Caution**

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

---

**Agent Mode**

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

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

**Procedure**

**Step 1**
Set properties for the agent.

*Note* The properties required for the agent depend on the type of ACD you use. The following example demonstrates the required properties for Unified CCE users.

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

*Note* In the Java CIL only: If the SetAgent() method is called on a session in which the current agent is different than the agent parameter in the SetAgent() method, the Java CIL automatically calls the Disconnect() method on the current session instance, generating an OnCloseConnection() event, then attempts to reconnect, generating an OnConnection() event. Then the new agent is set as the current agent.

The following example, which assumes the Session object has been created and connected to the CTI OS Server, demonstrates this task in Java:

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

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

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

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

Monitor Mode

Use Monitor Mode for applications that need to receive all events that CTI OS Server publishes or a specified subset of those events. Monitor Mode applications may receive events for calls, multiple agents, or statistics. The session receives specific events based on the event filter specified when setting the session to Monitor Mode.
Monitor Mode, as the name implies, is intended for use in applications that passively listen to CTI OS server events. Monitor Mode is not intended for use in applications that actively control the state of calls or agents. Such applications include but are not limited to the following:

- Applications that log in agents and change their state
- Applications that make or receive calls and change their state
- Applications that silently monitor agents

When a Monitor Mode session is initialized, the CTI OS Server performs a CPU intensive sequence of operations to provide the application with a snapshot of the state of the system. A large number of Monitor Mode applications connecting to CTI OS server at the same time, such as in a fail-over scenario, can cause significant performance degradation on CTI OS Server. Therefore, minimize the number of Monitor Mode applications connecting to CTI OS Server to two (2).

You can only use the button enablement feature in agent mode sessions and is not intended for Monitor Mode applications.

**Monitor Mode Filters**

**Overview Monitor Mode Filters**

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

**Warning**

By default the CTIOS server does a snapshotting which results in sending the info about all agents to the monitor mode connection. You control the behavior using the CTIOS_MONITORSESSIONSNAPSHOTMODE argument in the messagefilter args.

Use filter arg Enum_CtiOs.CTIOS_MONITORSESSIONSNAPSHOTMODE, 1 to turn off the snapshot.

**Filter String Syntax**

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

The general syntax for the filter String is as follows:

```
“key1=value1, value2, value3;key2=value4, value5, value6”
```
The filter string may also contain an asterisk (*), which is used as a wildcard to indicate any possible value. In addition, you can use a prefix to * to narrow the results. For example, using 10* matches 1001, 1002, 10003. However, CTI OS ignores any characters that follow the asterisk. For example, using 10*1 matches both 1001 and 1002.

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

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

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

For example, a filter string could be as follows:

```
S_MESSAGEID + "=" + CtiOsEnums.EventID.eAgentStateEvent + ";" + S_AGENTID + "=5128";
```

This example works as follows:

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

Filter Keys

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

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

For example, in Java:

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

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

Filters for Events for Monitored Calls

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

- Creates the filter
- Checks events to verify that the CTIOS_MONITORED parameter is present and is TRUE
Select Monitor Mode

To select Monitor mode for the connection:

Procedure

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

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

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

Note Always include the OnCtiOsFailure() event in the message filter so that the application can detect when a system component is online or offline.

Caution

A Monitor mode application that monitors any Call-related events must also monitor the OnCallEnd() event, as described on OnCallEnd() Event Monitoring, on page 78.

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

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

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

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

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

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

- With Unified CCE only, OnQueryAgentStateConf() for each team and each agent logged in
- OnSnapshotDeviceConf() for each device
- OnSnapshotCallConf()
- OnMonitorModeEstablished()

Deal with Failover in Monitor Mode

The CTI OS CIL does not support failover for Monitor Mode. Agents in Monitor Mode cannot recover their state after a failover. Furthermore, after a failover, the CTI OS CIL may leak Call objects.
To deal with failover in Monitor Mode:

**Procedure**

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

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

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

**Step 3**
Follow the steps starting at the beginning of the section Silent Monitoring, on page 87 to:

a) Create a session instance.
b) Set the event listener.
c) Set connection parameters.
d) Call the Connect() method.
e) Set the connection mode in the OnConnection() event handler.

### Settings Download

One of the many useful features of CTI OS is the ability to configure Agent Desktop settings after what is on the server and have them available to all agent desktops via the RequestDesktopSettings() method. You can make any changes after what is on the server instead of changing each and every desktop. Settings download are considered as part of the process of setting up a connection that the client application uses.

Desktop settings are stored in the registries on the machines running CTI OS Server. Centralizing the desktop settings on the server streamlines the process of changing or updating the agent desktop. A settings download occurs every time a client application connects and ensures that all the desktops are based on the same settings.

You can downloading settings from CTI OS Server after connecting and setting the mode via the RequestDesktopSettings() method on the Session object. The OnGlobalSettingsDownloadConf event indicates success and also returns the settings which are now available to the client application in the form of properties on the Session object. You can access these properties via the GetValue() methods. Refer to Chapter 9 for a list of all the properties of the Session object.

You can make the request for desktop settings either in the OnConnection event or in the OnSetAgentModeEvent event (if Agent mode has been specified). Sample code:

```vbnet
Private Sub m_Session_OnConnection(ByVal pDispParam As Object)
    'Issue a request to the server to send us all the Desktop 'Settings
    m_Session.RequestDesktopSettings eAgentDesktop
End Sub
```

The OnGlobalSettingsDownloadConf event passes back the settings and you can access them via the Session object. For example, the following snippet checks for Sound Preferences and specifically to see if the Dial Tone is Mute or not:

```vbnet
Private Sub _m_session_OnGlobalSettingsDownloadConf(ByVal pDispParam As Object)
    Dim SoundArgs As CTIOSARGUMENTSLib.Arguments
```

CTI OS Developer Guide for Cisco Unified ICM, Release 12.0(1)
' check if "SoundPreferences is a valid property

    If m_session.IsValid("SoundPreferences ") = 1 Then
        Set SoundArgs = m_session.GetValue("SoundPreferences")
        Dim DialToneArgs As CTIOSARGUMENTSLib.Arguments
        If Not SoundArgs Is Nothing Then
            If SoundArgs.IsValid("DialTone") = 1 Then
                Set DialToneArgs = SoundArgs.GetValue("DialTone")
            End If
        End If
        Dim Mute As Integer
        If Not DialToneArgs Is Nothing Then
            If DialToneArgs.IsValid("Mute") = 1 Then
                Mute = DialToneArgs.GetValueInt("Mute")
                If Mute = 1 Then
                    MsgBox "Dial Tone MUTE" //Your logic here
                Else
                    MsgBox "Dial Tone NOT MUTE" //Your logic here
                End If
            End If
        End If
    End If
End Sub

Disconnect from CTI OS Server Before Shutdown

Disconnecting from CTI OS Server (via the Disconnect() method) before shutting down is an important part of the client application functionality. The Disconnect() method closes the socket connection between the client application and CTI OS. On most switches, it does not log the agent out. If no logout request was issued before the Disconnect(), then on most switches the agent stays logged into the instrument even after the client application has shut down.

Disconnect is a higher priority method than all others. Before calling Disconnect, ensure that all prior requests have completed lest the call to Disconnect abort these requests. For example, calling Disconnect immediately after calling Logout can result in an agent not being logged out.

Upon Disconnect(), each object maintained by the Session (Call, Skillgroup, Wait) is released and no further events are received. Cleaning up the Agent object is the developer's responsibility because it was handed to the Session (via the SetAgent() method).

Code sample:

In the C++ and COM CIL only, to disconnect from CTI OS Server when the session mode has not yet been established by means of calling either CCtiOsSession::SetAgent(...) or CCtiOsSession::setMessageFilter(...), you must call for disconnect with an Arguments array containing the CTIOS_FORCEDDISCONNECT set to True.

    m_session.Disconnect
    // Perform disconnect
    if(m_ctiSession->GetValInt(CTIOS_CONNECTIONMODE) == eSessionModeNotSet )
    { // If the session mode has not yet been set by SetAgent or
      // SetSessionMode at the time of the disconnect.
      // we need to indicate the session that a disconnect needs to
      // be forced
        bool bAllocOk = true;
        Arguments * pDisconnectArgs = NULL;
Agent Login and Logout

Log In an Agent

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

Note

Before attempting to log in an agent, you typically request global configuration data to correctly handle a duplicate log in attempt. For more information, see Get Registry Configuration Values to Desktop Application, on page 67.

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

Procedural

Step 1
Create an instance of the Arguments class.

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

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

Step 3
Log in the agent.

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

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

    /* 2. Set log in values.*/
    rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AGENTID, "275");
    rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_PERIPHERALID, "5002");
    rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AGENTINSTRUMENT, "5002");
    rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AGENTPASSWORD, "********");
    rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AUTOLOGIN, "1");
```
/* 3. Log in the agent. */
int returnValue = agent.Login(rArgs);
}

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

The Login() method generates the following events:
- QueryAgentStateConf()
- AgentStateEvent(), if the agent is unknown or is logged out.

Note: The client application receiving these events must check both the ENABLE_LOGOUT and ENABLE_LOGOUT_WITH_REASON bitmasks. For more information, see In the OnButtonEnablementChange() Event, on page 74.

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

---

### Duplicate Login Attempts

#### Overview of Duplicate Login Attempts

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

You can handle duplicate log in attempts in three ways:
- Allow the Duplicate Log In with No Warning
- Allow the Duplicate Log In with a Warning
- Do not allow a duplicate log in

You control how duplicate log in attempts are handled in two ways:
- By configuring how duplicate log in attempts are handled on a global basis by creating custom values in the CTI OS Server Registry. By using custom values in the CTI OS Server registry to control how duplicate log in attempts are handled and downloading these settings to your desktop application as described in Log Out an Agent, on page 69, you can enable flexibility without having to modify your desktop application code.
- By implementing code in your desktop application to detect and to handle the duplicate log in attempt error according to the custom values in the CTI OS Server Registry. You can write code to handle duplicate log in attempts in each of the three ways listed above. When you need to change how such attempts are handled, you simply change the registry settings; you would not have to change the desktop application code.
Create Values in CTI OS Server Registry to Control Duplicate Sign In Attempts

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

⚠️ Warning

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

Create two custom values:

- custom_WarnIfAgentLoggedIn
- custom_RejectIfAgentLoggedIn

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

The following table lists the settings to control how duplicate login attempts are handled:

<table>
<thead>
<tr>
<th>Goal</th>
<th>custom_WarnIfAgentLoggedIn</th>
<th>custom_RejectIfAgentLoggedIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>To warn the agent of the duplicate log in attempt, but to allow the agent to proceed.</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>To allow the agent to proceed with the duplicate log in attempt with no warning.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>To not allow the agent to proceed with a duplicate log in attempt.</td>
<td>0 or 1</td>
<td>1</td>
</tr>
</tbody>
</table>

To create keys to control duplicate log in attempts:

**Procedure**

---

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

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

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

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

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

**Step 6**
Repeat steps 2 through 5 for the value custom_RejectIfAgentLoggedIn.
Agent Login with Incorrect Credentials

To prevent another agent log in with incorrect credentials, use the `SendIdentifyClientRequest` method to identify and detect the log in request.

Set the Method Argument to `Nil`. To invoke this method, use the session object.

The following examples demonstrate the method in:

- **C++:** `int SendIdentifyClientRequest()`
- **.NET:** `CilError SendIdentifyClientRequest()`
- **Java:** `int SendIdentifyClientRequest()`

Following is an example of how to use the method:

```java
if (CIL_OK != SessionObj.SendIdentifyClientRequest())
{
    LOG(CRITICAL, "CtiOsSession::SetAgent(...), SendIdentifyClientRequest: authentication will fail, aborting..");
    ReportError(CIL_FAIL);
    return CIL_FAIL;
}
```

Get Registry Configuration Values to Desktop Application

To get CTI OS registry configuration values to your desktop application to handle duplicate login attempts correctly, you must request global configuration settings, then extract the custom settings from the event. You typically do this task before attempting to log in an agent, in the `OnConnection()` event.

**Procedure**

1. **Create an instance of the Arguments class.**
2. **In the Arguments instance, set the value for the CTIOS_DESKTOPTYPE key to either:**
   - `CtiOs_Enums.DesktopType.eAgentDesktop`
   - `CtiOs_Enums.DesktopType.eSupervisorDesktop`
   
   **Note** Although the Arguments object must have one of these fields as a value for the CTIOS_DESKTOPTYPE key, this version of CTI OS does not utilize the desktop type parameter when sending global configuration data to a desktop application. Regardless of which field you use in defining the Arguments object, CTI OS returns all global configuration data with the `OnGlobalSettingsDownloadConf()` event. The desktop type indicators, through currently required, are reserved for future use.
3. **Request desktop settings for the session using the RequestDesktopSettings() method. This results in a OnGlobalSettingsDownloadConf() event.**

   The following example demonstrates steps 1 through 3 in Java:

   ```java
   /* 1. Create Arguments object*/
   Arguments rArgs = new Arguments();
   ```
/* 2. Set the desktop type.*/
rArgs.SetValue("CTIOS_DESKTOPTYPE",
    CtiOsEnums.DesktopType.eAgentDesktop);

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

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

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

Step 6
Get the Integer instances for the custom values you established for the key in the CTI OS Server registry.

Step 7
For convenience, get the int values for those Integers to test with, as described in the section Detect
Duplicate Login Attempts, on page 65.

The following example demonstrates steps 4 through 7 in Java:

```java
void OnGlobalSettingsDownloadConf(Arguments rArgs) {
    /* 4. Get the Arguments instance for the Login configuration
        information from the event Arguments parameter.*/
    Arguments loginArgs = rArgs.getValueArray(CTIOS_LOGIN);

    /* 5. Get the Arguments instance for the Connection Profile
        from the Login Arguments instance. */
    Arguments connectionProfilesArgs = loginArgs.GetValueArray(CTIOS_CONNECTIONPROFILES);

    /* 6. Get the Arguments instance for the specific switch from the Connection
         Profiles instance */
    Arguments IPCCLogInArgs = connectionProfilesArgs.GetValueArray("SoftACD")

    /* 7. Get the Integer instances for the custom values you entered in the CTI OS Server
        registry.*/
    Integer warningIntObj = IPCCLogInArgs.GetValueIntObj("custom_WarnIfAgentLoggedIn");
    Integer rejectIntObj = IPCCLogInArgs.GetValueIntObj("custom_RejectIfAgentLoggedIn");

    /* 8. Get the int values for those object to test later.*/
    custom_WarnIfAgentLoggedIn = warningIntObj.intValue();
    custom_RejectIfAgentLoggedIn = rejectIntObj.intValue();
}
```

**Detect Duplicate Login Attempt in Desktop Application**

You detect the duplicate log in attempt in the OnQueryAgentStateConf() event, which is sent after the
application calls SetAgent():
**Procedure**

**Step 1**
Get the agent state value from the Arguments instance passed to the event.

**Step 2**
Test the agent state value in the CtiOs Enums.AgentState interface, as follows.

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

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

The following example demonstrates this task in Java:

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

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

**Handle Duplicate Login Attempts in Desktop Application**

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

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

**Log Out an Agent**

To log out an agent:

**Procedure**

**Step 1**
Create an instance of the Arguments class.

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

**Note**
Unified CCE requires a reason code to log out. Other switches may have different requirements.

**Step 3**
Log out the agent.

The following example demonstrates this task in Java:

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

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

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

---

**Typical Logout Procedure**

When the Logout button is clicked the following actions need to happen:

1. **Call Logout request on your current agent.**

   You need to call Logout and not use SetAgentState(eLogout), because Logout provides additional logic to support pre-Logout notification, Logout failure notification, and resource cleanup.

   Here is the sample code for the same:

   ```cpp
   if(m_ctiAgent)
   {
       Arguments &rArgAgentLogout = Arguments::CreateInstance();
       //add reason code if needed
       rArgAgentLogout.AddItem(CTIOS_EVENTREASONCODE, reasonCode);
       int nRetVal = m_ctiAgent->Logout(rArgAgentLogout);
       rArgAgentLogout.Release();
   }
   ```

2. **Receive a response for the Logout request.**

   You can expect the following events in response to a Logout request:

   - **OnAgentStateChange** (with Logout agent state).
     
     or
     
     **OnControlFailure** (with the reason for the failure).

   - **OnPostLogout** (you additionally receive this event if the Logout request succeeds).

   **Note**

   You can disable statistics either prior to issuing the Logout request or upon receipt of the OnAgentStateChange to logout state. Use the OnPostLogout event to trigger session disconnect. This guarantee that all event listeners can make CTIOS server requests in response to the logout OnAgentStateChange event.

   See the following example code:

   ```cpp
   void CMyAppEventSink::OnPostLogout(Arguments & rArguments )
   {
       // Do not Disconnect if the reason code is Forced Logout
       // (particular failover case):
       int nAgentState = 0;
       if ( rArguments.GetValueInt(CTIOS_AGENTSTATE, &nAgentState) )
       {
           if (nAgentState == eLogout)
           {
               int nReasonCode = 0;
               if ( rArguments.GetValueInt(CTIOS_EVENTREASONCODE,
&nReasonCode) 
{
    if (CTIOS_FORCED_LOGOUT_REASON_CODE == (unsigned short)nReasonCode) 
    {
        return;
    }
}
}

// Disconnect otherwise
if( IsConnected() ) // if session is connected
{
    if(m_ctiSession)
    {
        m_ctiSession->Disconnect();
    }
}

3. If you are not concerned with whether the agent is successfully logged out prior to disconnect, issue a session Disconnect request without a Logout request.

4. Additionally, you must wait for OnConnectionClosed before destroying Agent and Session objects. This guarantee that the CIL has completed cleanup of the Session object prior to your calling Release on these objects.

5. Ensure that the Agent Object is set to NULL in the session before you Release the session object. For example, whenever your application is exiting and you are disconnecting the session object (for example, when the user closes your application window) do something similar to the code below:

```c
if (m_ctiSession)
{
    m_ctiSession->Disconnect();

    // stop all events for this session
    int nRetVal =
        m_pctiSession->RemoveAllInOneEventListener((IAllInOne *)
            m_pmyEventSink);

    // The application is closing, remove current agent from session
    CAgent * pNullAgent = NULL;
    m_Session->SetAgent(*pNullAgent);
    m_Session->Release();
    m_Session = NULL;
}
```

```c
if(m_ctiAgent)
{
    m_ctiAgent->Release();
    m_ctiAgent = NULL;
}
```

```c
if (m_pmyEventSink)
{
    m_pmyEventSink->Release();
    m_pmyEventSink = NULL;
}
```
Calls

Multiple Call Handling

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

- Apply incoming events to the correct call
- Select the correct call on which to make method calls (for example, telephony requests)

It is not necessary to maintain a set of Call objects to do this. Instead, the application can store the string UniqueObjectID of each call (keyword CTIOS_UNIQUEOBJECTID). CTIOS_UNIQUEOBJECTID is always included in the args parameter for each call event. You can obtain the actual Call object with the Session object's GetObjectFromObjectID() method to make a method call.

Current Call

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

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

The Current Call is set according to the following rules:

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

Get Call Object from Session

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

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

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

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

The following line shows this in Java:

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

Call Wrapup

The agent/supervisor desktop behaves differently at the end of a call depending on the following factors:

- The direction of the call (inbound or outbound)
- Configuration of Unified CCE or the ACD (whether wrapup data is required, optional, or not allowed)
- Configuration of CTIOS server

The CTI Toolkit Combo Desktop .NET sample shows how to use this information to display a wrapup dialog box that allows the agent to select from a set of pre-configured wrapup strings after an inbound call goes into wrapup state (see ProcessOnAgentStateEvent in SoftphoneForm.cs). On an agent state change event, if the state changes to WorkReady or WorkNotready state, this indicates that the agent has transitioned to call wrapup state. The CTIOS server provides the following key/value pairs in the event arguments to determine whether wrapup data is associated with the call and whether that data is required or optional.

CTIOS_INCOMINGOROUTGOING indicates the direction of the call. The defined values are:

0 = the direction of the call is unknown
1 = the call is an incoming call and the agent may enter wrapup data
2 = the call is an outgoing call and the agent may not enter wrapup data

You can use the GetValueInt method to obtain this value on the Agent object.

CTIOS_WRAPUPPOKENABLED indicates whether wrapup data is required for the recently ended call. A value of false indicates that wrapup data is not required. A value of true indicates that wrapup data is required. (In the Combo Desktop sample, this value is used as a boolean to determine whether the "Ok" button on the wrapup dialog box is enabled when no wrapup information is selected.) You can use the GetValueBool method to obtain this value on the Agent object.

The wrapup strings that are configured on CTIOS server are sent to the client during the login procedure and are stored under the keyword CTIOS_INCOMINGWRAPUPSTRINGS as an Arguments array within the Agent object. You can use the GetValueArray method to obtain the wrapup strings on the Agent object. For more information about how to configure wrapup strings on CTIOS server see the CTIO System Manager Guide for Cisco Unified ICM.

Logout and NotReady Reason Codes

Depending on the configuration of Unified CC or the configuration of CTIOS server, the agent/supervisor desktop may be required to supply a reason code when requesting an agent state change to Logout or NotReady state. The CTI Toolkit Combo Desktop .NET sample provides examples of how to implement reason codes in an agent/supervisor desktop. (See the btnLogout_Click and btnNotReady_Click methods in SoftphoneForm.cs.)
CTIOS server informs the CTIOS client of this configuration during the login process and the information is stored in the following properties on the Agent object:

**CTIOS_LOGOUTREASONREQUIRED** - This boolean value indicates whether a reason code is required for logout. A value of true indicates that a reason code is required. A value of false indicates that a reason code is not required. You can use the GetValueBool method to get this value on the Agent object.

**CTIOS_LOGOUTREASONCODES** - This Arguments array provides a list of the logout reason codes configured on CTIOS server. You can use the GetValueArray method to get this value on the Agent object.

**CTIOS_NOTREADYREASONREQUIRED** - This boolean value indicates whether a reason code is required when setting an agent to NotReady state. A value of true indicates that a reason code is required. A value of false indicates that a reason code is not required. You can use the GetValueBool method to obtain the value on the Agent object.

**CTIOS_NOTREADYREASONCODES** - This Arguments array provides a list of the not ready reason codes configured on CTIOS server. You can use the GetValueArray method to obtain the value on the Agent object.

### Applications and OnButtonEnablementChange() Event

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

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

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

### In the OnButtonEnablementChange() Event

To see if a button should be enabled, do a bitwise “AND” with the appropriate value listed in the Table included under the OnButtonEnablementChange event in Chapter 6.

The following examples shows this in Java:

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

### Not Ready Bitmasks in OnButtonEnablementChange() Event

A client application receiving the OnButtonEnablementChange() event must check both the ENABLE_NOTREADY and ENABLE_NOTREADY_WITH_REASON bitmasks in the event.
Failure to check both the ENABLE_NOTREADY and ENABLE_NOTREADY_WITH_REASON bitmasks can lead to problems properly displaying a NotReady control to the agent.

The following example shows this in Java:

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

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

    final long bitMask = LMask.longValue();

    /* Transfer modification of the GUI objects to the EventDispatchThread or we could have a thread sync issue. We're currently on the CtiOsSession's event thread.*/
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            /* Enable a button if it's bit is turned on. Disable it if not.*/
            m_appFrame.m_btnAnswer.setEnabled (((bitMask & ENABLE_ANSWER) > 0));
            m_appFrame.m_btnConference.setEnabled
                (((bitMask & ENABLE_CONFERENCE_COMPLETE) > 0));
            m_appFrame.m_btnCCConference.setEnabled
                (((bitMask & ENABLE_CONFERENCE_INIT) > 0));
            m_appFrame.m_btnHold.setEnabled (((bitMask & ENABLE_HOLD) > 0));
            m_appFrame.m_btnLogin.setEnabled (((bitMask & ENABLE_LOGIN)> 0));
            m_appFrame.m_btnLogout.setEnabled
                (((bitMask & (ENABLE_LOGOUT | CtIoS_Enums.ButtonEnablement.ENABLE_LOGOUT_WITH_REASON)) >
                0));
            m_appFrame.m_btnMakeCall.setEnabled
                (((bitMask & ENABLE_MAKECALL) > 0));
            m_appFrame.m_btnNotReady.setEnabled
                (((bitMask & (ENABLE_NOTREADY | ENABLE_NOTREADY_WITH_REASON)) > 0));
            m_appFrame.m_btnRelease.setEnabled
                (((bitMask & ENABLE_RELEASE)> 0));
            m_appFrame.m_bntRetrieve.setEnabled
                (((bitMask & ENABLE_RETRIEVE) > 0));
            m_appFrame.m_btnSSTransfer.setEnabled
                (((bitMask & ENABLE_SINGLE_STEP_TRANSFER) > 0));
            m_appFrame.m_btmSSTransfer.setEnabled
                (((bitMask & ENABLE_SINGLE_STEP_CONFERENCE) > 0));
            m_appFrame.m_btmTransfer.setEnabled
                (((bitMask & ENABLE_TRANSFER_COMPLETE) > 0));
            m_appFrame.m_btnCCTransfer.setEnabled
                (((bitMask & ENABLE_TRANSFER_INIT) > 0));
        }
    });
} // OnButtonEnablementChange
```
OnButtonEnablementChange() Event in Supervisor Desktop Applications

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

Making Requests

Telephony requests are made through either an Agent object or a Call object by calling the appropriate API methods listed in Chapters 9 and 10. It is important to ensure that a user cannot make multiple duplicate requests before the first request has a chance to be sent to the switch and the appropriate events sent back to the application, because this results in either multiple failures or unexpected results.

Multiple Duplicate Requests

Because it is important for a custom application to prevent a user from making a number of duplicate requests, the user should not be able to click the same button multiple times. A custom application should disable a clicked button until it is all right for the user to click it again.

Some examples of when Sample softphones re-enable a button that was clicked and disabled are listed below:

- Re-enable Connect/LoginBtn when:
  - LoginDlg canceled
  - ControlFailure or CTIOSFailure when login is in progress
  - In ProcessOnConnectionClosed()

- Re-enable Logout/DisconnectBtn when:
  - Logout ReasonCodes are required & Dlg pops up, but user clicks Cancel

- Re-enable NotReadyBtn when:
  - NotReady ReasonCodes are required & Dlg pops up, but user clicks Cancel

- Re-enable DialBtn, TransferBtn or ConferenceBtn when:
  - DialPad was closed with Cancel rather than Dial, depending on which was originally clicked

- Re-enable TransferBtn & ConferenceBtn when:
  - Received ControlFailure with MessageType parameter set to eConsultationCallRequest

- Re-enable EmergencyBtn when:
  - Received ControlFailure with MessageType parameter set to eEmergencyCallRequest

- Re-enable SupervisorAssistBtn when:
  - Received ControlFailure with MessageType parameter set to eSupervisorAssistRequest
• Re-enable any AgentStateBtn when:
  • Received ControlFailure with MessageType parameter set to eSetAgentStateRequest & lastAgentStateBtnClicked was the appropriate one

• Re-enable any of the buttons when:
  • Received OnButtonEnablementMaskChange indicating the button should be enabled.

Events

Event Order

A desktop application using the CTI OS API must handle events in the order they are sent by CTI OS.

⚠️ Warning
Because many events include agent state data and button enablement data indicating valid agent state transitions, if events are handled out of order agents may not be presented with valid options.

Coding Considerations for CIL Event Handling

The CTI OS CIL fires events to the application in a single thread. Keep the amount of time spent in a particular CIL event handler to a minimum to ensure timely delivery of subsequent CIL events. If a screenpop based on a call event (such as the OnCallDelivered event or the OnCallDataUpdate event) takes longer than a few seconds (for example, remote database lookup), delegate this operation to a separate thread or separate process so as not to block CTI OS event handling.

⚠️ Note
The order of arrival of CIL events is highly dependent on the ACD that is in use at the customer site. Therefore, CIL event order is not guaranteed. Do not write your event handling code in a manner that relies on a particular event order.

If an application calls a CIL API method from a CIL event callback routine it must ensure that the method call is made on the same thread as the CIL event callback. This rule applies to the following methods:

• SetCurrentCall
• SetAgent

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

When handling events in the browser using JavaScript, keep event processing time to a minimum because all other JavaScript execution (e.g., handling of button clicks) may be blocked during handling of the event.
OnCallEnd() Event Monitoring

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

Warning
The Call object in the CTI OS CIL is only deleted when the OnCallEnd() event is received. If the OnCallEnd() and OnCallDataUpdate() events are not monitored, Call objects accumulate and cause a memory leak.

Agent Statistics

Overview of Agent Statistics

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

- Set Up Agent Application to Receive Agent Statistics, on page 78
- Set Up Monitor Mode Application to Receive Agent Statistics, on page 79
- Agent Statistics Access, on page 82
- Agent Statistics Configuration, on page 83
- Agent Statistics Computed by Sample CTI OS Desktop, on page 83

Set Up Agent Application to Receive Agent Statistics

To set up an Agent application to receive agent statistics:

Procedure

Step 1 Create an instance of the Session class, as described on Connect to CTI OS Server, on page 54.
Step 2 Subscribe for events for the session, as described on Set Event Listener and Subscribe to Events, on page 54.

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

Step 3 Set connection parameters, as described on Set Connection Parameters for Session, on page 55.
Step 4 Connect the desktop application to the CTI OS Server, as described on Connect Session to CTI OS Server, on page 55.
Step 5 In the OnConnection() event handler, set the Agent for the session, as described on Select Agent Mode, on page 58.
Step 6 Log in the agent, as described on Log In an Agent, on page 64.
Step 7 Enable agents statistics using the EnableAgentStatistics() method.
Though the EnableAgentStatistics() method requires an Arguments parameter, there are no parameters to set for agent statistics; you can therefore send an empty Arguments instance as a parameter.

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

To disable agents statistics, use the DisableAgentStatistics() method.

The following example demonstrates this task in Java:

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

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

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

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

public void OnConnection(Arguments rArgs) {
    /* 5. Set agent for the session. */
    errorCode = rSession.SetAgent(agent);

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

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

### Set Up Monitor Mode Application to Receive Agent Statistics

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

CTI OS has a limitation in providing monitor-mode support to build agent desktop call-control applications, as well as having the ability to rely on button enablement messages.

**Procedure**

**Step 1** Create an instance of the Session class, as described on Connect to CTI OS Server, on page 54.

**Step 2** Subscribe for events for the session, as described on Set Event Listener and Subscribe to Events, on page 54.

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

**Step 3** Set connection parameters, as described on Set Connection Parameters for Session, on page 55.

**Step 4** Connect the desktop application to the CTI OS Server, as described on Connect Session to CTI OS Server, on page 55.

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

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

**Step 6** Set the message filter as described on Filters for Events for Monitored Calls, on page 60.

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

**Note** The “*:,” indicates all events for that agent.

b) Create an instance of the Arguments class.

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

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

**Step 7** Wait for any event for the agent, to ensure that the Agent instance exists for the Session.

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

**Note** This example uses a Wait object to wait.

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

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

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

**Caution** The agent must be logged in before you can use the EnableAgentStatistics() method.
Step 10

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

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

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

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

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

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

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

/*7. Wait for agent events.*/

rArgs = new Arguments();

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

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

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

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

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

Overview of Agent Statistics Access

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

- By implementing the `eOnNewAgentStatisticsEvent()` (in Java) or the `OnAgentStatistics()` event (in C++, COM, or VB 6.0)

Caution

The name of the event to access agent statistics is different in Java when compared to other languages supported by CTI OS.

- Through the Agent instance itself

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

**eOnNewAgentStatisticsEvent() in Message Filter (JAVA)**

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

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

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

For more information about message filters, see Monitor Mode Filters, on page 59.

**OnAgentStatistics() Event in Message Filter (C++ COM and VB)**

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

For more information about message filters, see Monitor Mode Filters, on page 59.

**Get Agent Statistics Through Agent Instance**

After you use the `EnableAgentStatistics()` method for the agent, agent statistics are available through that Agent instance.

To get the agent statistics perform the following procedure:

**Procedure**

1. Get the Arguments instance containing statistics from the Agent instance using the `GetValueArray()` method.
2. Parse the Arguments instance as needed to get specific statistics.

The following example demonstrates this task in Java:

```java
/* 1. Get Arguments instance.*/
Arguments rArgs = agent.GetValueArray(CtiOs_IKeywordIDs.CTIOS_STATISTICS);
```
Agent Statistics Configuration

You can change which agent statistics are sent to applications by modifying the registry on the CTI OS Server. For more information about how to change which agent statistics are sent to applications by default, see the CTI OS System Manager Guide for Cisco Unified ICM.

Agent Statistics Computed by Sample CTI OS Desktop

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

- \( \text{AvgTalkTimeToday} = \frac{\text{AgentOutCallsTalkTimeToday} + \text{HandledCallsTalkTimeToday}}{\text{AgentOutCallsToday} + \text{HandledCallsToday}} \)
- \( \text{CallsHandledToday} = \text{AgentOutCallsToday} + \text{HandledCallsToday} \)
- \( \text{TimeLoggedInToday} = \text{LoggedOnTimeToday} \)
- \( \text{TimeTalkingToday} = \text{AgentOutCallsTalkTimeToday} + \text{HandledCallsTalkTimeToday} \)
- \( \text{TimeHoldingToday} = \text{AgentOutCallsHeldTimeToday} + \text{IncomingCallsHeldTimeToday} \)
- \( \text{TimeReadyToday} = \text{AvailTimeToday} \)
- \( \text{TimeNotReadyToday} = \text{NotReadyTimeToday} \)
- \( \text{AvgHoldTimeToday} = \frac{\text{AgentOutCallsHeldTimeToday} + \text{IncomingCallsHeldTimeToday}}{\text{AgentOutCallsToday} + \text{HandledCallsToday}} \)
- \( \text{AvgHandleTimeToday} = \frac{\text{AgentOutCallsTimeToday} + \text{HandledCallsTimeToday}}{\text{AgentOutCallsToday} + \text{HandledCallsToday}} \)
- \( \text{AvgIdleTimeToday} = \frac{\text{NotReadyTimeToday}}{\text{AgentOutCallsToday} + \text{HandledCallsToday}} \)
- \( \text{PercentUtilizationToday} = \frac{\text{AgentOutCallsTimeToday} + \text{HandledCallsTimeToday}}{\text{LoggedOnTimeToday} + \text{NotReadyTimeToday}} \)

Skill Group Statistics

Overview of Skill Group Statistics

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

- Set Up Monitor Mode Application to Receive Skill Group Statistics, on page 84
Set Up Monitor Mode Application to Receive Skill Group Statistics

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

Procedure

**Step 1** Create an instance of the Session class, as described on Connect to CTIOSServer, on page 54.

**Step 2** Subscribe for events for the session, as described on Set Event Listener and Subscribe to Events, on page 54.

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

**Step 3** Set connection parameters, as described on Set Connection Parameters for Session, on page 55.

**Step 4** Connect the desktop application to the CTIOSServer, as described on Connect Session to CTIOSServer, on page 55.

**Step 5** Set the message filter as described on Filters for Events for Monitored Calls, on page 60.

a) Create String for the filter using the keyword S_FILERTARGET as the name and the event keyword (enum or number) eOnNewSkillGroupStatisticsEvent (numeric value 536871027) as the value.

b) Create an instance of the Arguments class.

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

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

a) Create an instance of the Arguments class.

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

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

**Attention** You must pass a value of "0" for the Skill Group Priority.

**Caution** The application must know the Skill Group ID, and the skill group's Peripheral ID, for any skill group to monitor. The application cannot dynamically get these values from CTIOSServer.
c) Use the Arguments instance as a parameter for the session’s EnableSkillGroupStatistics() method.
d) Repeat steps b and c for each skill group for which you want to receive events.

**Step 7**

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

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

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

c) Use the Arguments instance as a parameter for the session's DisableSkillGroupStatistics() method.

The following example demonstrates this task in Java:

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

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

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

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

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

/* 6. Enable statistics. */
rArgs = new Arguments();
rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_SKILLGROUPNUMBER, sgNumber);
rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_PERIPHERALID, peripheralID);
rSession.EnableSkillGroupStatistics(rArgs);
```
Skill Group Statistics Access

Overview of Skill Groups Statistics Access

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

- In Java, eOnNewSkillGroupStatisticsEvent()
- In C++, COM, or VB, OnSkillGroupStatisticsUpdated()

⚠️ **Caution**

The name of the event through which to access skill group statistics is different in Java from other languages supported by CTI OS.

**eOnNewSkillGroupStatisticsEvent() in Message Filter (JAVA)**

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

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

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

For more information about message filters, see Monitor Mode Filters, on page 59.

**eOnNewSkillGroupStatisticsEvent() in Message Filter (C++ COM and VB)**

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

For more information about message filters, see Monitor Mode Filters, on page 59.

Skill Group Statistics Sent to Desktop Application

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

For more information about how to change which skill group statistics are sent to desktop applications, see the CTI OS System Manager Guide for Cisco Unified ICM.

Skill Group Statistics Computed by Sample CTI OS Desktop

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

- AvgCallsQTimeNow = CallsQTimeNow/CallsQNow
Silent Monitoring

There are two (mutually exclusive) silent monitoring methods:

- CTI OS based silent monitoring
- Cisco Unified Communications Manager (Unified CM) based silent monitoring

For more information, see the CTI OS System Manager Guide for Cisco Unified ICM. For more information about how to enable silent monitor in your application, see CTI OS Based Silent Monitoring, on page 88 or Unified CM-Based Silent Monitoring in Your Application, on page 92, as applicable.
CTI OS Based Silent Monitoring

CTI OS Silent Monitor functionality is only available in the C++ and COM CILs.

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

The first thing a client application should do is to create a silent monitor object instance. The application should then set this object instance as the current manager in the session object. The CIL provides the interface to this functionality. A client application can work in one of two possible modes:

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

Silent Monitor does not work until you set the session mode using one of the following function calls:

- `Session.SetAgent()` for an Agent mode application
- `Session.SetMessageFilters()` for a Monitor mode application

Create a Silent Monitor Object

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

The following VB 6.0 code sample demonstrates how to create a SilentMonitorManager object with COM CIL and make it the current manager in the Session object:

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

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

'Make the object the current manager
errorcode = m_Session.SetCurrentSilentMonitor(m_SMManager)
```

Session Mode

After you set this new object as the current object, set the manager's work mode to Monitoring for the monitoring client and Monitored for the monitored client. The following sections provide code examples. For more information about syntax of the StartSMMonitoringMode and SMMonitoredMode methods, see SilentMonitorManager Object, on page 483.
Monitoring Mode

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

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

```vbnet
Dim m_Args As New Arguments
' Assemble arguments to set the work mode
m_Args.AddItem("HeartbeatInterval", 5)
m_Args.AddItem("HeartbeatTimeout", 15)
' Address or hostname of the silent monitor service
m_Args.AddItem("SMSAddr", "localhost")
' Port on which silent monitor service is listening
m_Args.AddItem("SMSListenPort", 42228)
' QoS setting when sending messages to the silent monitor service
m_Args.AddItem("SMSTOS", 0)
' Milliseconds between heartbeats
m_Args.AddItem("SMSHeartbeats", 5000)
' Number of missed heartbeats before the connection to the
'silent monitor service is considered disconnected
m_Args.AddItem("SMSRetries", 3)
' Port number where audio will be listened for
m_Args.AddItem("MediaTerminationPort", 4000)
' Set the working mode to monitoring
m_SMManager.StartSMMonitoringMode(args)
```

Monitored Mode

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

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

```vbnet
Dim m_Args As New Arguments
' Assemble arguments to set the work mode
m_Args.AddItem("HeartbeatInterval", 5)
m_Args.AddItem("HeartbeatTimeout", 15)
' Address or hostname of the silent monitor service
m_Args.AddItem("SMSAddr", "localhost")
' Port on which silent monitor service is listening
m_Args.AddItem("SMSListenPort", 42228)
' QoS setting when sending messages to the silent monitor service
m_Args.AddItem("SMSTOS", 0)
' Milliseconds between heartbeats
m_Args.AddItem("SMSHeartbeats", 5000)
' Number of missed heartbeats before the connection to the
'silent monitor service is considered disconnected
m_Args.AddItem("SMSRetries", 3)
' Extension number of the IP Phone to monitor
m_Args.AddItem("MonitoringDeviceID", 1234)
' Set the working mode to monitored
m_silentMonitor.StartSMMonitoredMode(args)
```

Silent Monitor Session

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

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

MrsClient Code Sample

Private Sub btnStartSM_OnClick()
Dim m_Args As New Arguments

'Agent to monitor
m_Args.AddItem("AgentID", "23840")
m_Args.AddItem("PeripheralID", "5000")
m_Args.AddItem("HeartbeatInterval", 5)
m_Args.AddItem("HeartbeatTimeout", 15)

'If MonitoringIPPort is not specified, port 39200 will be used by 'default.
m_Args.AddItem("MonitoringIPPort", 39200)

'Request silent monitor session to start
m_SMManager.StartSilentMonitorRequest(m_Args, m_nSMSessionKey)
End Sub

Private Sub m_session_OnSilentMonitorStatusReportEvent(ByVal pIArguments As CTIOSCLIENTLib.IArguments)
Dim strAgent As String
Dim nMode As Integer

nMode = pIArguments.GetValueInt("StatusCode")
If nMode = eSMStatusMonitorStarted Then strAgent = pIArguments.GetValueString("MonitoredUniqueObjectID")
    MsgBox "Silent Monitor Status",,
    "Started Monitoring Agent: " & strAgent
Else
    MsgBox "Silent Monitor Status",,
    "Request Failed with code = " & nMode
End If
End Sub

Private Sub tmrScreening_Timer()
'After listening the conversation for 30 sec, drop monitoring session

'Assemble arguments for stop request
'Agent to monitor
m_Args.AddItem "SMSessionKey", m_nSMSessionKey

'Request silent monitor session to stop
m_SMManager.StopSilentMonitorRequest(m_Args, m_nSMSessionKey)
End Sub
Monitored Client Code Sample

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

Private Sub m_session_OnSilentMonitorStopRequestedEvent(ByVal pIArguments As CTIOSCLIENTLib.IArguments, ByVal bDoDefaultProcessing)
    Dim strRequestInfo As String
    strRequestInfo = pIArguments.DumpArgs
    MsgBox "Request to Stop Silent Monitor Received", , strRequestInfo
End Sub

Silent Monitor Manager Shutdown

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

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

'Stop Silent Monitor ModeRequest
m_SMManager.StopSilentMonitorMode

'Remove silent monitor manager object from session
errorCode = m_session_SetCurrentSilentMonitor(Nothing)

'Destroy silent monitor manager object
errorCode = m_session.DestroySilentMonitorManager()

CTI OS Silent Monitor Management in Monitor Mode

CTI OS Silent Monitor is configured, initiated, and ended the same in monitor mode as it is in agent mode. There is one additional step in monitor mode. You must include the OnCallRTPStarted and OnCallRTPStopped events in the filter used by the monitor mode application. An example follows.

// 116 = OnCallRTPStarted
// 117 = OnCallRTPStopped
m_session.SetMessageFilter("MessageID = 116, 117")

Note

For more information, see Session Modes, on page 10.
Unified CM-Based Silent Monitoring in Your Application

CCM-Based Silent Monitor Overview

CCM based silent monitor is the Call Manager implementation of silent monitor. When CCM based silent monitor is used, silent monitor is implemented as a call. After initiating silent monitor, the supervisor is able to hear agent conversations using their phone.

Agents can only be silent monitored by one supervisor at a time.

CCM based silent monitor is supported in all CILs.

The CTI Toolkit Combo Desktop .Net sample includes CCM based silent monitor source code.

The following section describes how to enable CCM based silent monitor in custom CTI OS applications.

CTI OS Monitor Mode Applications

CCM based silent monitor is not supported for CTI OS monitor mode applications.

CCM-Based Silent Monitor Request

Before you initiate CCM based silent monitor, ensure that you configure CCM based silent monitor. For more information, see Determine if CCM-Based Silent Monitoring Is Enabled, on page 95.

CCM based silent monitoring is initiated through the SuperviseCall() method associated with the supervisor's Agent object. To start silent monitor:

- Set the SupervisoryAction parameter to eSupervisorMonitor.
- Set the AgentReference parameter to the unique object ID of the agent to be silent monitored.
- Set the AgentCallReference parameter to the unique object ID of the call to be silent monitored.

When the request is successfully initiated and the silent monitor call is established, the supervisor and agent applications receive the OnSilentMonitorStartedEvent. You can use this event to trigger application specific logic.

The following figure illustrates the messaging that occurs between the CIL and CTI OS Server after an application initiates a CCM based silent monitor request using Agent.SuperviseCall().
C# Code Sample for Initiating Silent Monitor Session

```csharp
Agent curAgent = session.GetCurrentAgent();
Agent monAgent = curAgent.GetMonitoredAgent();
Call monCall = curAgent.GetMonitoredCall();

string monAgentID;
monAgent.GetStringValue(
    Enum_CtiOs.CTOS_UNIQUEOBJECTID,
    out monAgentID);

string monCallID;
monCall.GetStringValue(
    Enum_CtiOs.CTOS_UNIQUEOBJECTID,
    out monCallID);

Arguments args = new Arguments();
args.SetValue(Enum_CtiOs.CTOS_AGENTREFERENCE, monAgentID);
args.SetValue(Enum_CtiOs.CTOS_AGENTCALLREFERENCE, monCallID);
args.SetValue(
    Enum_CtiOs.CTOS_SUPERVISORYACTION,
    SupervisoryAction.eSupervisorMonitor);
```
Current Agent Being Silently Monitored

If an application needs to determine if the current agent is being silently monitored, then compare the current agent unique object ID against the silent monitor target agent unique ID carried in the SilentMonitorStartedEvent.

**Code Sample for Determining if Current Agent Is Target of Silent Monitor Call**

The parameter `args` carries the payload of an OnSilentMonitorStartedEvent.

```csharp
public bool IsCurrentAgentTargetAgent(Arguments args)
{
    bool isTarget = false;

    if (m_ctiSession != null)
    {
        Agent rAgent = m_ctiSession.GetCurrentAgent();
        if (rAgent != null)
        {
            string curAgentUID;
            rAgent.GetValueString(Enum_CtiOs.CTIOS_UNIQUEOBJECTID, out curAgentUID);
            if (curAgentUID != null)
            {
                string targetAgentUID;
                args.GetValueString(Enum_CtiOs.CTIOS_SILENMONITOR_TARGET_AGENTUID, out targetAgentUID);
                if (targetAgentUID != null)
                {
                    isTarget = curAgentUID == targetAgentUID;
                }
            }
        }
    }
    return isTarget;
}
```

**CCM-Based Silent Monitor Request End**

CCM based silent monitoring is stopped using the SuperviseCall method associated with the supervisor's Agent object. To stop silent monitor, set the SupervisoryAction parameter to eSupervisorClear. Set the AgentReference parameter to the unique object ID of the agent currently silent monitored. Set the AgentCallReference parameter to the unique object ID of the call that resulted from the initiation of silent monitor (Agent.SuperviseCall[eSupervisorMonitor]). The application receives the SilentMonitorStopRequestedEvent event when the stop silent monitoring request is processed.

The following figure illustrates the message flow.
You can also release the silent monitor call using the Call.Clear() method.

**Code Sample for Ending Silent Monitor Session**

```csharp
Agent curAgent = session.GetCurrentAgent();

string monAgentID;
curAgent.GetValueString(
    Enum_CtiOs.CTIOS_SILENTMONITOR_TARGET_AGENTUID,
    out monAgentID);

string monCallID;
curAgent.GetValueString(
    Enum_CtiOs.CTIOS_SILENTMONITOR_CALLUID,
    out monCallID);

Arguments args = new Arguments();
args.SetValue(Enum_CtiOs.CTIOS_AGENTREFERENCE, monAgentID);
args.SetValue(Enum_CtiOs.CTIOS_AGENTCALLREFERENCE, monCallID);
args.SetValue(
    Enum_CtiOs.CTIOS_SUPERVISORYACTION,
    SupervisoryAction.eSupervisorClear);

CilError ret = curAgent.SuperviseCall(args);
```

**Determine if CCM-Based Silent Monitoring Is Enabled**

To determine if CCM based silent monitoring is enabled, use the Session.IsCCMSilentMonitor() method if the application uses the C++, Java, or .Net CIL. Use the CCMBasedSilentMonitor value stored in the session object if the application uses the COM CIL:

```csharp
/// <summary>
/// Determines if CCM Based Silent Monitor is enabled
/// </summary>
public bool IsCCMSilentMonitor()
{
    if ( _ctiSession == null )
    {
        return false ;
    }
}
Agent Greeting

There are several ways to control the behavior of the Agent Greeting feature. You can enable or disable Agent Greeting for the duration of the Agent's login session. Note that when an Agent logs in, the feature is automatically enabled.

Code example:

```cpp
Arguments &rArgAgentAction = Arguments::CreateInstance();
rArgAgentAction.AddItem("AgentAction", commandRequested);
int nRetVal = m_pCtiAgent->SetAgentGreetingAction(rArgAgentAction);
rArgAgentAction.Release();
```

Where "commandRequested" is an int with the value 1 (to disable) or 2 (to enable).

Deployment of Custom CTI OS Applications

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

Application Deployment Using ActiveX Controls

ActiveX controls need all the components for COM deployment plus the components listed in the following table.

Table 11: ActiveX Control DLLs

<table>
<thead>
<tr>
<th>DLL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentSelectCtl</td>
<td>AgentSelect ActiveX control</td>
</tr>
<tr>
<td>agentstateCtl.dll</td>
<td>Agentstate ActiveX control</td>
</tr>
<tr>
<td>AlternateCtl.dll</td>
<td>Alternate ActiveX control</td>
</tr>
<tr>
<td>answerctl.dll</td>
<td>Answer/Release ActiveX control</td>
</tr>
<tr>
<td>arguments.dll</td>
<td>Arguments COM class</td>
</tr>
<tr>
<td>badlinectl.dll</td>
<td>Badline ActiveX control</td>
</tr>
<tr>
<td>buttoncontrol.dll</td>
<td>Basic Button ActiveX control</td>
</tr>
<tr>
<td>ccnsmt.dll</td>
<td>Cisco EVVBU Media Termination ActiveX control</td>
</tr>
<tr>
<td>chatctl.dll</td>
<td>Chat ActiveX control</td>
</tr>
<tr>
<td>conferencectl.dll</td>
<td>Conference ActiveX control</td>
</tr>
</tbody>
</table>
You must copy and register ActiveX controls using the regsvr32 Windows utility. Some ActiveX controls are dependent on others. For example, all Button type controls (for example, AgentStatectl.dll) depend on (buttoncontrol.dll) and all Grid type controls (for example, Ctiocallappearance.dll) depend on gridcontrol.dll. The following table means that for a dll listed in the left column to work properly, all dlls listed in the right column (Dependencies) need to be available (copied and registered).

The following table lists the dependencies of CTI OS ActiveX controls.

### Table 12: Dependencies of CTI OS ActiveX Controls

<table>
<thead>
<tr>
<th>DLL File</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agentselectctl</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll</td>
</tr>
</tbody>
</table>

**Note**  
When used in a .NET application must include: AxInterop.AgentSelectCtl.dll AxInterop.AgentSelectCtl.dll
<table>
<thead>
<tr>
<th>DLL File</th>
<th>Dependencies</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>agentstatectl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommondlgs.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a.NET application must include: AxInterop.AgentStateCtl.dll Interop.AgentStateCtl.dll</td>
</tr>
<tr>
<td>AlternateCtl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a.NET application must include: AxInterop.AlternateCtl.dll Interop.AlternateCtl.dll</td>
</tr>
<tr>
<td>answerctl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a.NET application must include: AxInterop.AnswerCtl.dll Interop.AnswerCtl.dll</td>
</tr>
<tr>
<td>arguments.dll</td>
<td>ATL80.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a.NET application must include: Cisco.CTIOSARGUMENTSLib.dll</td>
</tr>
<tr>
<td>badlineectl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a.NET application must include: AxInterop.BadLineCtl.dll Interop.BadLineCtl.dll</td>
</tr>
<tr>
<td>buttoncontrol.dll</td>
<td>ATL80.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a.NET application must include: AxInterop.ButtonControl.dll Interop.ButtonControl.dll</td>
</tr>
<tr>
<td>ccnsmt.dll</td>
<td>Traceserver.dll, LIBG723.dll</td>
<td></td>
</tr>
<tr>
<td>chatctl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a.NET application must include: AxInterop.ChatCtl.dll Interop.ChatCtl.dll</td>
</tr>
<tr>
<td>conferencectl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommondlgs.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a.NET application must include: AxInterop.ConferenceCtl.dll Interop.ConferenceCtl.dll</td>
</tr>
<tr>
<td>cticommondlgs.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a.NET application must include: Cisco.CTICOMMONDLGSLib.dll</td>
</tr>
</tbody>
</table>

Building Your Custom CTI Application

Application Deployment Using ActiveX Controls
<table>
<thead>
<tr>
<th>DLL File</th>
<th>Dependencies</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTIOSAgentStatistics.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, Gridcontrol.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a .NET application must include: AxInterop.CTIOSAgentStatistics.dll</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctioscallappearance.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommondlg.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a .NET application must include: AxInterop.CTIOSCallAppearance.dll</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctiosclient.dll</td>
<td>ATL80.dll, arguments.dll, ctiosracetext.exe, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a .NET application must include: Cisco.CTIOSCLIENTLib.dll</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctiossessionresolver.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a .NET application must include: Cisco.CTIOSSESSIONRESOLVERLib.dll</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTIOSSkillGroupStatistics.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, Gridcontrol.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a .NET application must include: AxInterop.CTIOSSkillGroupStatistics.dll</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctiosstatusbar.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, cticommondlg.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a .NET application must include: AxInterop.CTIOSStatusBar.dll</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EmergencyAssistCtl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a .NET application must include: AxInterop.EmergencyAssistCtl.dll</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gridcontrol.dll</td>
<td>ATL80.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a .NET application must include: AxInterop.GridControl.dll</td>
</tr>
<tr>
<td>DLL File</td>
<td>Dependencies</td>
<td>Note</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>holdctl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll,</td>
<td>When used in a .NET application must include:AxInterop.HoldCtl.dll</td>
</tr>
<tr>
<td></td>
<td>MSVCR80.dll</td>
<td>Interop.HoldCtl.dll</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: When used in a .NET application must include:AxInterop.HoldCtl.dll</td>
</tr>
<tr>
<td>IntlResourceLoader.dll</td>
<td>ATL80.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>Note: When used in a .NET application must include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cisco.INTLRESOURCELOADERLib.dll</td>
</tr>
<tr>
<td>makecallctl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommndlgs.dll,</td>
<td>When used in a .NET application must include:AxInterop.MakeCallCtl.dll</td>
</tr>
<tr>
<td></td>
<td>MSVCP80.dll, MSVCR80.dll</td>
<td>Interop.MakeCallCtl.dll</td>
</tr>
<tr>
<td>ReconnectCtl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll,</td>
<td>When used in a .NET application must include:AxInterop.ReconnectCtl.dll</td>
</tr>
<tr>
<td></td>
<td>MSVCR80.dll</td>
<td>Interop.ReconnectCtl.dll</td>
</tr>
<tr>
<td>recordctl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll,</td>
<td>When used in a .NET application must include:AxInterop.RecordCtl.dll</td>
</tr>
<tr>
<td></td>
<td>MSVCR80.dll</td>
<td>Interop.RecordCtl.dll</td>
</tr>
<tr>
<td>SilentMonitorCtl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, ccnsmt.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a .NET application must include:AxInterop.SilentMonitorCtl.dll</td>
</tr>
<tr>
<td>SubclassForm.dll</td>
<td>ATL80.dll, MSVCP80.dll, MSVCR80.dll</td>
<td>When used in a .NET application must include:AxInterop.SubclassForm.dll</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Interop.SubclassForm.dll</td>
</tr>
<tr>
<td>SupervisorOnlyCtl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll,</td>
<td>When used in a .NET application must include:AxInterop.SupervisorOnlyCtl.dll</td>
</tr>
<tr>
<td></td>
<td>MSVCR80.dll</td>
<td>Interop.SupervisorOnlyCtl.dll</td>
</tr>
<tr>
<td>transferctl.dll</td>
<td>ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommndlgs.dll,</td>
<td>When used in a .NET application must include:AxInterop.TransferCtl.dll</td>
</tr>
<tr>
<td></td>
<td>MSVCP80.dll, MSVCR80.dll</td>
<td>Interop.TransferCtl.dll</td>
</tr>
</tbody>
</table>
**Application Deployment Using COM (but Not ActiveX Controls)**

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

- **CTIOSClient.dll**
  
  When used in a .NET application must include: Cisco.CTIOSCLIENTLib.dll

- **Arguments.dll**
  
  When used in a .NET application must include: Cisco.CTIOSARGUMENTSLib.dll

- **CtiosSessionresolver.dll** (only if used – see previous discussion)
  
  When used in a .NET application must include: Cisco.CTIOSSESSIONRESOLVERLib.dll

- **ATL80.dll** (only if not already available on target system)
  
  If the client application uses silent monitoring in monitoring mode, ccnsmt.dll is needed. If the client application uses silent monitoring in monitored mode, wpcap.dll is also a dependency.

You must copy and register the dll files on the target system. To register, use the Windows utility regsvr32.exe providing the dll name (for example, regsvr32 ctiosclient.dll).

ATL80.dll is a Microsoft Dynamic Link Library implementing the Active Template Library used by CTIOS. It is usually available on most Windows client systems in a windows system directory (for example, \winnt\sysct32 on Windows 2000). Because CTIOS depends on this DLL, you must copy and register it if it is not already available at the target system.

**Application Deployment Using C++ CIL**

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

**Application Deployment Using .NET CIL**

Applications built with the .NET CIL class libraries require the following assemblies to be distributed with the custom application.

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetCil.dll</td>
<td>.NET CIL Class library, contains the CTIOS object classes</td>
</tr>
<tr>
<td>NetUtil.dll</td>
<td>.NET Util Class library, contains helper and utility classes used in conjunction with .NET CIL</td>
</tr>
</tbody>
</table>
In addition to NetCil.dll and NetUtil.dll, the .NET Combo sample requires the CTIOSVideoCtl.dll, which is located in C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\dotNet CIL\Controls.

You can install both assembly libraries in the Global Assembly Cache (GAC) at the application host computer or they can be at the working directory of the custom client application.

**Custom Application and CTI OS Security**

A custom application that launches the SecuritySetupPackage.exe program to create CTI OS client certificate request needs to add the **InstallDir** registry value under the following registry key:

```
SOFTWARE\CiscoSystems\CTI Desktop\CtiOs
```

If the **InstallDir** registry value does not exist, then the setup program fails and aborts the installation process, otherwise the program uses the **InstallDir** registry value to create and copy the security files to the right place after it appends Security directory to it.

For example, if the **InstallDir** registry value is

```
<drive>:\Program Files\Cisco Systems\CTIOS Client
```

then the security files should be under

```
<drive>:\Program Files\Cisco Systems\CTIOS Client\Security
```

**Supervisor Applications**

This section describes how to build a supervisor desktop for Unified CCE. The following documentation references the source of the CTI OS Toolkit Combo Desktop when describing how to build a supervisor desktop. This section also references a class called CTIObject. The CTI OS Toolkit Combo Desktop uses this class to wrap CIL functionality.

The source code for the Combo Desktop is found in the following directories.

- `<Install Drive>\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\dotNet CIL\Samples\CTI Toolkit Combo Desktop.NET`
- `<Install Drive>\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\dotNet CIL\Samples\CtiOs Data Grid.NET`

In the following section, string keys are used as keys to method calls. This is for the sake of readability. A developer writing an application can use either string or integer based keys.

**General Flow**

The general flow of a supervisor application is as follows:

1. Request the supervisor's teams.
2. Start monitoring the supervisor's team.
3. Select a team member and start monitoring the selected team member's activity.
4. Perform supervisory actions on the currently monitored call.

These steps illustrate the layers of a supervisor application. First, the application gets the team. After the team is retrieved, the supervisor application can monitor agents. This generates more events/information allowing the supervisor application to monitor agent calls.

**Monitored and Unmonitored Events**

When writing a supervisor application, developers are confronted with two types of events: monitored events and unmonitored events.

Unmonitored events are received for agent, call, and button enablement events associated with the supervisor. Monitored events are received to notify the supervisor of agent, call, and button enablement events corresponding to an agent or call that is currently monitored by the supervisor. These events carry a field named CTIOS_ISMONITORED. This field is set to true.

For example, if a supervisor changes state to ready, the supervisor receives an AgentStateEvent. If a supervisor is monitoring an agent and the monitored agent changes state, the supervisor receives an OnMonitoredAgentState event. Call events behave in a similar manner. When the supervisor puts a call on hold, the supervisor receives an OnCallHeld event. When the supervisor is monitoring an agent and that agent puts a call on hold, the supervisor receives an OnMonitoredCallHeld event.

Button enablement events behave differently. When the supervisor is monitoring agents on the supervisor's team, the agent receives OnButtonEnablementChange events for the monitored agent. It is important for the application not to apply these events to elements of the application that control the supervisor's or any of the supervisor's calls state. For example, if a monitored agent changes state to ready, the supervisor receives a ButtonEnablementChange event. The supervisor should not apply this event since the event does not apply to the supervisor's state.

To determine if an event is monitored, check the payload of the event for the “Monitored” field. If the field exists and is set to true, the event is a monitored event.

**Supervisor Application Flow to Request and Monitor Team**

This section discusses steps 1 and 2 in the flow of a supervisor application. The methods and events listed below are used to request and monitor the team.

**Methods Called:**

Agent.RequestAgentList(Arguments args)
Agent.StartMonitoringAgentTeams(Arguments args)

**Events Processed:**

OnNewAgentTeamMember
OnMonitoredAgentStateChange
OnMonitoredAgentInfo
OnSkillInfo

The following diagram illustrates the flow of messages between the application and CTI OS Server when the supervisor application requests its team and then requests to monitor the team. Because logging in a supervisor...
is the same as logging in an agent, this diagram picks up at the first AgentStateEvent after the agent has logged in.

**Figure 13: Message Flow Between the Application and the CTI OS Server**

The requests leading up to and including Agent.StartMonitoringAgent() is in CTIObject.StartMonitoringAgent(). When writing a supervisor application, the developer should call Agent.RequestAgentTeamList() and Agent.StartMonitoringAllAgentTeams(). The developer should call these methods after the supervisor logs in. In the CTI OS Toolkit Combo Desktop, this is done when processing the eAgentStateEvent in the SupervisorUIManager class’ ProcessAgentStateEvent() method. SupervisorUIManager checks to see that the current agent is a supervisor. If so and if CTIObject.StartMonitoringTeams() has not already been called, CTIObject.StartMonitoringTeams() is called. CTIObject.StartMonitoringTeams() then calls Agent.RequestAgentTeamList() and Agent.StartMonitoringAllAgentTeams().

If these requests are successful, the desktop begins receiving OnNewAgentTeamMember, OnMonitoredAgentStateChange, and MonitoredAgentInfoEvent events. The next sections describe how to handle each of these events.

**OnNewAgentTeamMember Events**

Process OnNewAgentTeamMember events as follows.

The OnNewAgentTeamMember event is received for two possible reasons:
1. After the application calls Agent.RequestAgentTeamList(), OnNewAgentTeamMember events are sent for each member of the supervisor's team.

2. An agent has been added or removed from the supervisor's team.

To address point 2 above, examine the field “ConfigOperation” in the payload of the OnNewAgentTeamMember event. If this flag does not exist or exists and is set to TeamConfigFlag.CONFIG_OPERATION_ADDAGENT (1), add the agent to the grid. If the flag exists and is not set to TeamConfigFlag.CONFIG_OPERATION_ADDAGENT, remove the agent from the grid.

In supervisor applications, use the value in the UniqueObjectID field of the event to uniquely reference/track each agent in the supervisor's team. This ID uniquely identifies each agent cached on the CIL.

**OnNewAgentTeamMember Events and Supervisors**

Because the supervisor is considered part of the team, an OnNewAgentTeamMember event is sent for the supervisor logged in to the application. If the developer does not want to include the supervisor in the agent team grid, compare the current agent ID to the ID of the agent carried in the OnNewAgentTeamMember event. If the values are equal, do not add the supervisor to the grid.

If the developer does not want to add primary supervisors to the grid, retrieve the Agent object stored in the CIL using the Session.GetObjectFromObjectID() method. When calling Session.GetObjectFromObjectID(), set the value in the “UniqueObjectID” (Enum_CtiOs.CTIOS_UNIQUEOBJECTID) field of the OnNewAgentTeamMember event as the key (first parameter to this method). This method returns an Agent object. Check the properties of the Agent object for the field “AgentFlags” (Enum_CtiOs.CTIOS_AGENTFLAGS). If the field exists with the TeamConfigFlag.AGENT_FLAG_PRIMARY_SUPERVISOR (0x01) bit set, the agent is a primary supervisor and should not be added to the grid.

It is possible for an agent to be team supervisor while not being a member of the team. Some supervisor applications, including the combo desktop, may not want to add this type of supervisor to the agent select grid. This is tricky because supervisors that are not part of the team generate OnMonitoredAgentStateChange events. The agent select grid normally updates when the OnMonitoredAgentStateChangeEvent is received. To prevent this, supervisors who are not members of the team that they are supervising need to be marked as such. You can use this information to avoid updates when an OnMonitoredAgentStateChangeEvent is received for a supervisor that is not part of the team. To accomplish this, the application leverages the following:

1. OnNewAgentTeamMember events are not received for supervisors that are not part of the team.

2. The CIL keeps a cache of all the agents and supervisors that it knows about. Agents in this cache have properties that can be modified by applications built on top of the CIL.

Knowing this, the application marks every agent that is included in an OnNewAgentTeamMember event as a member of this supervisor's team. When OnMonitoredAgentStateChangeEvent events are received, the agent select grid only updates when the agent that is represented by the event is marked as a member of the team. In short, any agent that does not send an OnNewAgentTeamMember event to the CIL is not displayed in the agent select grid. This is illustrated in the SupervisorUIManager.ProcessMonitoredAgentStateChange() method.

**OnMonitoredAgentStateChange Events**

OnMonitoredAgentStateChange events are sent when an agent in the supervisor's team changes state. Supervisor applications, like the CTI OS Toolkit Combo Desktop, use this event to update structures that store the
supervisor's team (the agent team grid). This event is processed similar to OnNewAgentTeamMember. However, there is one subtle difference. Instead of using the Arguments object carried with the event, the application uses the arguments associated with the Agent object cached by the CIL. This is done to correctly handle skill group membership changes related to dynamic reskilling. The CIL contains logic that processes the OnMonitoredAgentStateChange and determines whether or not an agent has been added or removed from a skill group. The changes in the agent's skill group membership are reflected in the Agent object's properties.

**OnMonitoredAgentInfo Event**

You can use this event to populate the following agent information:

- AgentID
- AgentFirstName
- AgentLastName
- LoginName

**Time in State**

If your application needs to track an agent's time in state, it can be done as follows. The algorithm is contained in AgentSelectGridHelper.cs. The first part of the algorithm resides in the AgentData.UpdateData() method. This method decides if the agent's state duration is known or unknown. An agent's state duration is unknown if the agent was just added to the grid or if the agent's state has not changed since being added to the grid. If a state change is detected after the agent was added to the grid, the time of the state change is marked.

Second, there is a timer callback that the AgentSelectGridHelper class starts when the grid is initialized. The timer callback fires every ten seconds. When the callback fires, the method AgentSelectGridHelper.m_durationTimer_Tick() cycles through all of the rows in the grid. Each row whose Time in State column is not unknown has its value set to the time the agent changed state minus the current time.

**OnSkillInfo Event**

OnSkillInfo events are sent to the CIL when skillgroup statistics are enabled using the Agent.EnableSkillGroupStatistics() method. These events are used to populate the fields in the Skill Name column of the team grid. OnSkillInfo events carry the ID of a skill group and its corresponding name. The AgentSelectGridHelper processes this event by storing a mapping of skill group IDs to skill group names. After the map is updated, each field in the Skill Name column is updated to reflect the new skill name.

**Agent Team Information Displayed in Grid Format**

If your application would like to display agent team information in a grid similar to the one used by the CTI OS Toolkit Combo Desktop, the following table illustrates which events supply information for each column.

Please refer to CtiOsDataGrid.AgentSelectGridHelper as an example of handling the OnNewAgentTeamMember event.
Table 14: Agent Grid Data Population

<table>
<thead>
<tr>
<th>Column</th>
<th>Event</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>OnNewAgentTeamMember OnMonitoredAgentStateChange OnMonitoredAgentInfoEvent</td>
<td>Enum_CtiOs.CTIOS_AGENTFIRSTNAME Enum_CtiOs.CTIOS_AGENTLASTNAME</td>
</tr>
<tr>
<td>Login Name</td>
<td>OnMonitoredAgentStateChange OnMonitoredAgentInfoEvent</td>
<td>Enum_CtiOs.CTIOS_LOGINNAME</td>
</tr>
<tr>
<td>Agent ID</td>
<td>OnNewAgentTeamMember OnMonitoredAgentStateChange OnMonitoredAgentInfoEvent</td>
<td>Enum_CtiOs.CTIOS_AGENTID</td>
</tr>
<tr>
<td>Agent State</td>
<td>OnNewAgentTeamMember OnMonitoredAgentStateChange</td>
<td>Enum_CtiOs.CTIOS_STATE</td>
</tr>
<tr>
<td>Time in State</td>
<td>OnMonitoredAgentStateChange</td>
<td>For more information, see Time in State, on page 106.</td>
</tr>
<tr>
<td>Skill Group</td>
<td>OnMonitoredAgentStateChange</td>
<td>Enum_CtiOs.CTIOS_NUMSKILLGROUPS</td>
</tr>
<tr>
<td>Skill Name</td>
<td>OnSkillInfoEvent</td>
<td>For more information, see OnSkillInfoEvent, on page 106.</td>
</tr>
<tr>
<td>Available for Call</td>
<td>OnNewAgentTeamMember</td>
<td>Enum_CtiOs.CTIOS_AGENTAVAILABILITYSTATUS</td>
</tr>
</tbody>
</table>

The Skill Group column lists the field from the Arguments object as CTIOS_NUMSKILLGROUPS. This field tells the developer how many skill groups the agent belongs to. To obtain information about each of the agent's skill groups the developer should construct the following loop to get information about each of the agent's skill groups (code taken from the sample source file CtiOsDataGrid\AgentSelectGridHelper.cs).

```csharp
// Check to see if the event carries an array of skillgroups
// (OnNewAgentTeamMember)
int numGroups;
if ( args.GetValueInt(Enum_CtiOs.CTIOS_NUMSKILLGROUPS, out numGroups) )
{
    CtiOsDataGrid.Trace(
        Logger.TRACE_MASK_METHOD_AVG_LOGIC,
        methodName,
        "Found skillgroup numbers") ;
    m_skillGroupNumbers.Clear() ;
    for ( int j = 1 ; j <= numGroups ; j++ )
    {
        CtiOsDataGrid.Trace(
            Logger.TRACE_MASK_METHOD_AVG_LOGIC,
```
methodName, 
string.Format("Looking for skillgroup at position {0}", j));

string unknownStr = string.Format(
    AgentSelectGridHelper.STRING_UNKNOWN_SG_FORMAT, j);

// Keys for individual skillgroups are formatted as SkillGroup[{index}]
//
// string sgKey = string.Format(
//    AgentSelectGridHelper.STRING_SKILLGROUP_FORMAT, j);

// Each element of the array is an argument containing
// skillgroup information.
//
// Arguments sgInfo;
if (!args.GetValueArray(sgKey, out sgInfo))
{
    CtiOsDataGrid.Trace(
        Logger.TRACE_MASK_WARNING, 
        methodName, 
        string.Format("No skillgroup info at position {0}", j));
    m_skillGroupNumbers.Add(unknownStr);
}
else
{
    string sgStr;
    if (sgInfo.GetValueString(
        Enum_CtiOs.CTIOS_SKILLGROUPNUMBER, 
        out sgStr))
    {
        CtiOsDataGrid.Trace(
            Logger.TRACE_MASK_METHOD_AVG_LOGIC, 
            methodName, 
            string.Format("Found skillgroup number {0} at position {1}", sgStr, j));
        m_skillGroupNumbers.Add(sgStr);
    }
    else
    {
        CtiOsDataGrid.Trace(
            Logger.TRACE_MASK_WARNING, 
            methodName, 
            string.Format("No skillgroup number at position {0}", j));
        m_skillGroupNumbers.Add(unknownStr);
    }
}


Supervisor Application Flow to Monitor an Agent

This section discusses step 3 in the flow of a supervisor application. The methods and events listed below are used to monitor an agent.

Methods Called:
Agent.StartMonitoringAgent(Arguments args)

Events Processed:
OnSupervisorButtonChange
OnStopMonitoringAgent
OnMonitoredAgentStateChange
OnMonitoredCallBegin
OnMonitoredCallCleared
OnMonitoredCallConferenced
OnMonitoredCallConnectionCleared
OnMonitoredCallDataUpdate
OnMonitoredCallDelivered
OnMonitoredCallDequeued
OnMonitoredCallDiverted
OnMonitoredCallEstablished
OnMonitoredCallFailed
OnMonitoredCallHeld
OnMonitoredCallOriginated
OnMonitoredCallQueued
OnMonitoredCallReachedNetwork
OnMonitoredCallRetrieved
OnMonitoredCallServiceInitiated
OnMonitoredCallTransferred
OnMonitoredCallTranslationRoute
OnMonitoredCallEnd

After a supervisor application is informed of an agent team member via the OnNewAgentTeamMember event, the supervisor can monitor the agent via the Agent.StartMonitoringAgent() method. The following sequence diagram illustrates the call to StartMonitoringAgent() and the events sent upon successful completion of the call.
Figure 14: Sequence Diagram for StartMonitoringAgent() and Successful Call Completion

The requests leading up to and including Agent.StartMonitoringAgent() is in the CTIObject.StartMonitoringAgent() method. When calling the Agent.StopMonitoringAgent(), the Agent object associated with the supervisor (the current agent) is the target of the method. The parameter is an Arguments object set as follows:

Table 15: Agent.StopMonitoringAgent Parameter

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentReference</td>
<td>The UniqueObjectID of the currently monitored agent.</td>
</tr>
</tbody>
</table>

When calling Agent.StartMonitoringAgent(), the Agent object associated with the supervisor (the current agent) is the target of the method. The parameter is an Arguments object set as follows:

Table 16: Agent.StartMonitoringAgent Parameter

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentReference</td>
<td>The UniqueObjectID of the agent to begin monitoring.</td>
</tr>
</tbody>
</table>
**OnSupervisorButtonChange**

This event is delivered to define the operations that the supervisor can successfully execute. The operations included in this event are as follows:

- Log out an agent on the team
- Make an agent on the team ready
- Enable silent monitor
- Enable barge-in on agent
- Enable intercept call

The application uses the bitmask carried by this event to enable or disable the functionality listed above. The `ProcessSupervisorButtonChange()` method in SupervisorUIManager illustrates how to process this event.

**Monitored Call Events**

The majority of events listed with `StartMonitoringAgent()` are monitored call events. These events inform the supervisor of monitored agent calls beginning, ending, and changing. The combo desktop uses these events to populate its monitored calls grid.

**Supervisor Application Makes Agent Ready or Logs Agent Out**

When `StartMonitoringAgent()` is called for a given agent, the supervisor application begins receiving `OnSupervisorButtonChange` events. This event can indicate that the monitored agent is in a state where the supervisor can make the agent ready or log the agent out. The following section describes how a supervisor application can make an agent on the supervisor's team ready or log the agent out.

To make an agent ready, the desktop calls the method `Agent.SetAgentState()`. When calling this method, the Agent object representing the monitored agent is used as the target of the method. The parameter is an `Arguments` object populated with the following key/value pairs:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupervisorID</td>
<td>The ID of the supervisor who is making the agent ready. This value is the value of the AgentID key associated with the current agent (the current agent is the agent passed into the call to <code>Session.SetAgent()</code> when first logging in the agent).</td>
</tr>
<tr>
<td>AgentState</td>
<td>The state to which to set the agent. In this case, the state is ready (integer with value 3).</td>
</tr>
</tbody>
</table>

To log out an agent, the desktop calls the method `Agent.SetAgentState()`. When calling this method, the Agent object representing the monitored agent is used as the target of the method. The parameter is an `Arguments` object populated with the following key/value pairs:
Table 18: Agent.SetAgentState Parameter (Logout) A

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupervisorID</td>
<td>The ID of the supervisor who is making the agent ready. This value is the value of the AgentID key associated with the current agent (the current agent is the agent passed into the call to Session.SetAgent() when first logging in the agent).</td>
</tr>
<tr>
<td>AgentState</td>
<td>The state to which to set the agent. In this case, the state is ready (integer with value 3).</td>
</tr>
<tr>
<td>EventReasonCode</td>
<td>The value associated with this key is 999. The value 999 indicates to the rest of Unified CCE that the agent was logged out by their supervisor.</td>
</tr>
</tbody>
</table>

An agent involved in a call is not logged out until the agent is disconnected from the call. Both the out-of-the-box desktop and the combo desktop warn the supervisor of this behavior. To do this, check the state of the currently monitored agent. If the agent's state is talking, hold, or reserved, the monitored agent is involved in one or more calls and is not logged out until the agent is disconnected from all calls. This is illustrated in SupervisorUIManager.m_btnMonLogoutAgentClick().

Successfully calling Agent.SetAgentState() should be followed by one or more SupervisorButtonChange and MonitoredAgentEvents reflecting the change in the monitored agent's state.

**Supervisor Application Flow to Monitor a Call**

This section discusses step 4 in the flow of a supervisor application. The methods and events listed below are used to monitor a call.

**Methods Called**

Agent.StartMonitoringCall()
Agent.SuperviseCall()

**Events Processed**

Events Processed
OnSupervisorButtonChange
AgentStateEvents
CallEvents
MonitoredCallEvents

**MonitoredCallEvents**

As stated in the “Monitoring Agents” section, calling Agent.StartMonitoringAgent() triggers MonitoredCallEvents for the agent specified in Agent.StartMonitoringAgent(). The MonitoredCallEvents received by the supervisor desktop inform the desktop of the state of the monitored agent's calls. The combo desktop uses these events to populate and update the monitored calls grid. For more information see the SupervisorUIManager and CallAppearanceHelper classes.

To monitor a call, the supervisor calls the Agent.StartMonitoringCall() method. The target of the call is the current agent (Agent object representing the supervisor). StartMonitoringCall() takes an Arguments object
with the CallReference key set to the UniqueObjectID of the call to be monitored. This is illustrated in the CTIOBJECT.StartMonitoringCall() method.

**Barging into Calls**

The following sequence diagram illustrates a request to barge into an agent's call. In this sequence diagram, the supervisor application is divided into four components to illustrate the different events that affect the different pieces of a supervisor application.

*Figure 15: Sequence Diagram for Barging into a Call*

*Figure 16:*

After Agent.StartMonitoringCall() is called for a specific call, the application begins receiving SupervisorButtonChangeEvent events. When a call is being monitored, the SupervisorButtonChange event can carry a bitmask indicating that the call can be barged into. To barge-in on a call, the application calls the Agent.SuperviseCall() method. The target of the SuperviseCall() method is the current agent (the Agent object that represents the supervisor). The parameter to the method is an Arguments object with the following key/value pairs:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentReference</td>
<td>The UniqueObjectID of the currently monitored agent</td>
</tr>
<tr>
<td>CallReference</td>
<td>The UniqueObjectID of the currently monitored call</td>
</tr>
<tr>
<td>SupervisoryAction</td>
<td>The value 3. For the .NET CIL, this is SupervisoryAction.eSupervisorBargeIn</td>
</tr>
</tbody>
</table>

When successfully calling this method, the application receives many events because this method not only changes the state of the monitored call, but also delivers a call to the supervisor which changes the supervisor's state. When an OnButtonEnablementChangeEvent event is received, be sure to check the monitored flag. If the flag
does not exist or exists and is set to false, apply the event to any application specific logic or UI to control the supervisor's state. This is illustrated in SoftphoneForm.OnEvent(). Notice that this method discards any event that is monitored.

One or more OnSupervisorButtonChange events are received by the application. These events notify the application that it is now possible to intercept the agent's call.

The trickiest piece of handling the events that result from a successful call to Agent.SuperviseCall() is handling the resulting Call and MonitoredCall events. You should apply all CallEvents to whatever application specific object and/or UI element is managing calls directly connected to the supervisor's device (SoftphoneForm in the combo desktop). You should apply all MonitoredCallEvents to whatever application specific object and/or UI element is managing calls connected to the supervisor's team members/monitored agents (SupervisorUIManager in the combo desktop).

Calling SuperviseCall() with the SupervisoryAction set to barge-in initiates a consultative conference between the caller, agent, and supervisor. This means that whatever UI elements and/or objects that handle monitored calls has to be able to handle the set of events that set up a consultative conference. In general, this is not too difficult. The consultative call is joined to the conference call by sending a MonitoredCallEndEvent to end the consultative call. Then a MonitoredCallDataUpdateEvent is used to change the ID of the call to the conference. The MonitoredCallEndEvent takes care of cleaning up the consultative call. The trick is to check OnMonitoredCallDataUpdateEvents for the OldUniqueObjectID key. If this key exists, it means that the UniqueObjectID of a call has changed. OldUniqueObjectID stores the old/obsolete ID of the call. UniqueObjectID stores the new ID of the call. This new ID is carried in all future events for the call. Application logic must be updated based on this information or new events for the call are not tracked correctly.

### Intercepting Calls

After a supervisor has barged into an agent's call, the supervisor can intercept the call. This is done by calling the Agent.SuperviseCall() method. The target of the SuperviseCall() method is the current agent (the Agent object that represents the supervisor). The parameter to the method is an Arguments object with the following key/value pairs:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentReference</td>
<td>The UniqueObjectID of the currently monitored agent</td>
</tr>
<tr>
<td>CallReference</td>
<td>The UniqueObjectID of the currently monitored call</td>
</tr>
<tr>
<td>SupervisoryAction</td>
<td>The value 4. For the .NET CIL, this is SupervisoryAction.eSupervisorIntercept</td>
</tr>
</tbody>
</table>

Calling this method removes the agent from the call. This means that OnMonitoredEndCall events are received for the agent. Also, OnSupervisorButtonChange events are sent to reflect the current state of the monitored agent.

### Monitored Call Data

Setting monitored call data is similar to setting call data on an agent's call. The only difference is that the monitored call is the target of the Call.SetCallData() method. You can retrieve the currently monitored call by calling Agent.GetMonitoredCall() where the current agent (the Agent object that represents the supervisor) is the target of the Agent.GetMonitoredCall() method.
Sample Code in CTI OS Toolkit

The CTI OS Toolkit provides several samples that illustrate how to use the various CTI OS CILs in custom applications. These samples are categorized according to the CIL (.NET, Java, or Win32) that they use.

.NET Samples

Of all the samples provided in the CTI OS toolkit, the .NET sample applications provide the most complete set of coding examples. Therefore, use the .NET samples as the reference implementation for custom CTI OS application development regardless of which language you plan to use in your custom development.

Use the Java and Win32 samples as secondary references to highlight syntactic differences as well as minor implementation differences between the CILs.

CTI Toolkit Combo Desktop .NET

The CTI Toolkit Combo Desktop .NET sample illustrates how to use the .NET CIL to build a fully functional agent or supervisor desktop. Though this sample is written in C#, it is a good reference for how to make CIL requests and handle CIL events in an agent mode CIL application. This sample illustrates the following CIL programming concepts:

• Agent mode connection to CTI OS
• Agent desktop functionality (call control, agent state control, statistics)
• Supervisor desktop functionality (team state monitoring, barge-in, intercept)
• Outbound option functionality
• Button enablement
• Failover

CTI Toolkit Combo Desktop Configuration

The .NET CTI Combo desktop is configured via an XML file found in the current working directory of the desktop.

The name of the file used to configure the CTI Toolkit Combo Desktop is “CTIOSSoftphoneCSharp.exe.config”. The desktop attempts to find the file in the current directory. If the file is not found, the desktop creates the file and displays the following error message.

The user can now edit the file to fill in the appropriate values.

Following is an example configuration file.
The configuration file is composed of the following elements. These elements are as follows:

classification – This element contains the configuration for the desktop.

appSettings – This section defines configuration settings that are shared by every Windows user that logs in to the system. A system administrator needs to configure these values for the appropriate CTI OS Servers and ports. Each of this element's sub-elements defines key value pairs used to configure the desktop.

  - LogFilePath – The value for this key is the path to the log file as well as the prefix of the name of the log file. The name of the Windows user, the log file's creation time, and the extension “.log” are appended to form the complete name of the log file. For example, if the desktop was run at 11:58 AM on May 23, 2005, the log file would have the name CtiOsClientLog.JoeUser.050523.11.58.04.5032.log.

  - CtiOsA – The name or IP address of one of the CTI OS Server peers.

  - CtiOsB – The name or IP address of the other CTI OS Server peer.

  - PortA – The port used to connect to the CTI OS Server specified by the CtiOsA key.

  - PortB – The port used to connect to the CTI OS Server specified by the CtiOsB key.

  - configSections – This section defines Windows user specific sections of the configuration file.

These sections are defined using the section element. Note that in the sample configuration file there is a section element under configSections corresponding to the element tagged with the Window's user name “JoeUser” under the configuration element. You should not have to manually modify this section. As different Windows users use the desktop, this section is modified to include section elements for each of the users.

The rest of the configuration file comprises elements that define configuration specific to different Windows users. For each section element in the configSections element, there is a corresponding element under the configuration element. These elements are used to store information specific to given users such as trace mask, agent login ID, dialed numbers, and so on. Most of the attributes in this element do not need to be modified. The one attribute that may need modification is the TraceMask attribute. This attribute is used to control the amount of information logged to the log file.

**CtiOs Data Grid.NET**

This sample is a set of helper classes that are used in other .NET CIL samples.
**All Agents Sample.NET**

This sample illustrates how to use the .NET CIL to build a monitor mode application that monitors agents. Though this sample is written in C#, it is a good reference in general for how to create a monitor mode CIL application. This sample illustrates the following CIL programming concepts:

- Monitor mode connection to CTI OS
- When to enable connect and disconnect buttons for a monitor mode application
- How to handle failover in monitor mode.
- Filtering for agent events

**All Calls Sample.NET**

This sample illustrates how to use the .NET CIL to build a monitor mode application that monitors calls. This sample illustrates the following CIL programming concepts:

- Monitor mode connection to CTI OS
- Connect and Disconnect error handling
- Filtering for call events
- Filtering for silent monitor call events

---

**Note**

For CCM based silent monitoring only. Filtering for silent monitor calls is only applicable to CCM based silent monitoring.

---

**Java CIL Samples**

**AllAgents** - This sample illustrates how to use the Java CIL to build a monitor mode application that monitors calls.

**JavaPhone** - This sample illustrates how to use the Java CIL to create a rudimentary agent mode application.

**Win32 Samples**

**CTI Toolkit AgentDesktop** - This sample illustrates how to use the Win32 COM CIL’s ActiveX controls to create an agent desktop using VisualBasic .NET.

**CTI Toolkit SupervisorDesktop** - This sample illustrates how to use the Win32 COM CIL’s ActiveX controls to create a supervisor desktop using VisualBasic .NET.

**CTI Toolkit Outbound Desktop** - This sample illustrates how to use the Win32 COM CIL’s ActiveX controls to create an outbound option desktop using VisualBasic .NET.

**CTI Toolkit C++Phone** - This sample illustrates how to use the C++ CIL to create a rudimentary agent mode application.
CHAPTER 5

CTI OS ActiveX Controls

The CTI OS Developer Toolkit includes a set of ActiveX controls to enable rapid application development. ActiveX controls are typically UI components (there are also ActiveX controls that are invisible at run time) that enable easy drag-and-drop creation of custom CTI applications in a variety of container applications. Container applications include: Microsoft Visual Basic 6.0, Microsoft Internet Explorer, Microsoft Visual C++, Borland Delphi, Sybase Powerbuilder and other applications supporting the OC96 ActiveX standard.

The CTI OS Agent Desktop and CTI OS Supervisor Desktop for Unified CCE applications are both Visual Basic applications based on the CTI OS ActiveX controls.

For more information, see CTI OS Agent Desktop User Guide for Cisco Unified ICM and the CTI OS Supervisor Desktop User Guide for Cisco Unified ICM.

The following table lists the ActiveX controls included with CTI OS. As seen in the table, you can group CTI OS Controls into Agent Related Controls, Call Related Buttons, Statistics Controls, and Supervisor Controls.

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Agent Related Controls</strong></td>
<td></td>
</tr>
<tr>
<td>AgentGreetingCtl</td>
<td>Provides a UI for turning the Agent Greeting feature on or off.</td>
</tr>
<tr>
<td>AgentStateCtl</td>
<td>Provides UI to for login, logout, ready, not ready and wrapup requests, also enables the use to specify reason codes for logout and Not_Ready (if supported and configured).</td>
</tr>
</tbody>
</table>
## Control

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChatCtl</td>
<td>Provides UI to send text messages to a supervisor or (if allowed) to other agents.</td>
</tr>
<tr>
<td>EmergencyAssistCtl</td>
<td>Provides UI to place Emergency and Supervisor Assist calls. These calls allow agents to conveniently contact a supervisor if they need help. You must configure the corresponding Unified ICM scripts for this control to work.</td>
</tr>
<tr>
<td>Record GreetingCtl</td>
<td>Provides a UI for recording and managing agent greeting messages by placing a Make Call request to a hard coded “RecordAgentGreeting” Dialed Number and setting the Placement type (CPT_RECORD_AGENT_GREETING) to 7.</td>
</tr>
</tbody>
</table>

### Call Related Controls

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlternateCtl</td>
<td>Provides UI for alternate requests. If an agent has Call A active and Call B on hold, alternate puts call A on hold and makes Call B active. Useful during consult calls.</td>
</tr>
<tr>
<td>AnswerCtl</td>
<td>Provides UI to answer a call. Only a call with state “LCS_Alerting” (Ringing) can be answered.</td>
</tr>
<tr>
<td>BadLineCtl</td>
<td>Provides a UI to report a Bad Line, such as bad voice quality or equipment problems.</td>
</tr>
<tr>
<td>CallAppearanceCtl</td>
<td>A grid control displaying call information, including call status and context data.</td>
</tr>
<tr>
<td>ConferenceCtl</td>
<td>Provides UI to place a conference call in single step or consultative mode.</td>
</tr>
<tr>
<td>HoldCtl</td>
<td>Provides UI to put calls on hold and retrieve them.</td>
</tr>
<tr>
<td>MakeCallCtl</td>
<td>Provides UI to enter a telephone number and place a make call request.</td>
</tr>
<tr>
<td>ReconnectCtl</td>
<td>Provides a UI for reconnect requests. If an agent has Call A active and Call B on hold, reconnect hangs up call A and makes Call B active. Useful during consult calls to return to the original call.</td>
</tr>
<tr>
<td>StatusBarCtl</td>
<td>Visually displays information about the logged on agent (id, instrument, extension, current state).</td>
</tr>
<tr>
<td>RecordCtl</td>
<td>Provides UI for Call Recording requests (start/stop recording), the requests are forwarded to CTI Server, so they care handled by a configured call recording service.</td>
</tr>
</tbody>
</table>
### Control and Description

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransferCtl</td>
<td>Provides UI to transfer a call in single step or consultative mode.</td>
</tr>
</tbody>
</table>

#### Statistics Controls

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentStatisticsCtl</td>
<td>A grid control displaying real-time agent statistics. Columns are configurable at CTIOS server (for more information, see <em>CTIOS System Manager Guide for Cisco Unified ICM</em>).</td>
</tr>
<tr>
<td>SkillgroupStatisticsCtl</td>
<td>A grid control displaying real time skill group statistics. Columns are configurable at CTIOS server (for more information, see <em>CTIOS System Manager Guide for Cisco Unified ICM</em>).</td>
</tr>
</tbody>
</table>

#### Supervisor Controls

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

### Property Pages

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

### Button Controls and Grid Controls

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

- StatusbarControl
- ChatCtl
- Utility controls (such as CtiCommonDlgs and SubClassFormCtl)
The Utility controls, such as CtiCommonDlg and SubClassFormCtl (used by the CTI OS Agent and Supervisor desktops), are for Internal Use Only.

As such they share common principles.

The following table describes button enablement scenarios only for call control, agent state control and supervisor assist in Unified CCE.

The video control button (under the tools group) is not included in standard Unified CCE desktops. This button is related to controls exercised on the supervisor desktop.

Table 22: Button Enablement in a Standard CTI OS Desktop for Unified CCE

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Buttons enabled and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent is not logged in to a desktop</td>
<td>Only the Log-in button is enabled.</td>
</tr>
<tr>
<td>Agent in the Not ready state</td>
<td>The agent is in the Not Ready state.</td>
</tr>
<tr>
<td></td>
<td>The following buttons are enabled: Ready, Supervisor Assist, Emergency Assist, Statistics and Chat Control, and Make Call Control.</td>
</tr>
<tr>
<td></td>
<td>Note The Make Call Control button allows you to dial out only in the Not Ready (NR) state.</td>
</tr>
<tr>
<td>Agent in the Ready state</td>
<td>The agent is in the Ready state.</td>
</tr>
<tr>
<td></td>
<td>The following buttons are enabled: Not Ready, Supervisor Assist, Emergency Assist, Statistics and Chat Control, and Make call Control.</td>
</tr>
<tr>
<td></td>
<td>Note The Make Call Control button allows you to dial out only in the Not Ready state.</td>
</tr>
<tr>
<td>Agent gets an incoming call that is alerted on the agent desktop</td>
<td>The agent is in the Reserved state.</td>
</tr>
<tr>
<td></td>
<td>The following buttons are enabled: Statistics, Chat Control, and Bad Line Ctrl.</td>
</tr>
<tr>
<td>Scenarios</td>
<td>Buttons enabled and Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Agent answers the call</td>
<td>The agent is in the <strong>Talking</strong> state.</td>
</tr>
<tr>
<td></td>
<td>The following buttons are enabled: Hold, Release, Supervisor Assist, Emergency Assist, Conference, transfer, Statistics, Chat ctl, and Badline control.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>The Ready, Not Ready, and wrap-up buttons are enabled based on the agent desktop settings configured in the <strong>Configuration Manager</strong>.</td>
</tr>
<tr>
<td>Agent puts an answered call on hold</td>
<td>The agent is in the <strong>On hold</strong> state.</td>
</tr>
<tr>
<td></td>
<td>The following buttons are enabled: Retrieve, Release, Supervisor Assist, Emergency Assist, Statistics, Chat ctl, and Badline control.</td>
</tr>
<tr>
<td>Agent retrieves the call on hold</td>
<td>The agents is in the <strong>Talking</strong> state.</td>
</tr>
<tr>
<td></td>
<td>The following buttons are enabled: Hold, Release, Supervisor Assist, Emergency Assist, Conference, transfer, Statistics, Chat ctl and Badline control.</td>
</tr>
<tr>
<td>Agent releases the call</td>
<td>The agent continues to be in the same state he was in before talking the call.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>The agent is in the Wrap-up state, provided Wrap-up is configured.</td>
</tr>
<tr>
<td></td>
<td>If Wrap-up was not configured, then the <strong>Ready</strong> and <strong>Not Ready</strong> buttons are enabled.</td>
</tr>
<tr>
<td>Agent initiates a conference</td>
<td>The agent is in the <strong>Talking</strong> state.</td>
</tr>
<tr>
<td></td>
<td>The following buttons are enabled: Statistics, Chat ctl, Bad Line Ctrl, and Reconnect.</td>
</tr>
</tbody>
</table>
| Consult conference call is answered | **Conference Initiator Desktop**  
The agent is in the **Talking** state.                                                                 |
|                                 | The following buttons are enabled: Statistics, Chat ctl, BadLineCtrl, Alternate, Reconnect, and Conference complete.                                           |
| **Note**                        | The Ready, NR and Wrap-up buttons are enabled based on the agent desktop setting.                                                                          |
|                                 | **Conference Receiver Desktop**  
The agent is in the **Talking** state.                                                                 |
<p>|                                 | The following buttons are enabled: Hold, Release, Supervisor Assist, Emergency Assist, Conference, transfer, Statistics, Chat ctl, and Badline control.       |</p>
<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Buttons enabled and Description</th>
</tr>
</thead>
</table>
| Complete conference is done for the conference initiator | The agent continues to be in the same state he was in, before talking the call.  
**Note** The agent is in the Wrap-up state, provided Wrap-up is configured.  
If Wrap-up was not configured, then the Ready and Not Ready buttons are enabled.                                                                                                                                                                                                                     |
| Agent initiates a consult transfer            | The agent is in the **Talking** state.  
The following buttons enabled: Statistics, Chat ctrl, BadLine Ctrl, and Reconnect.                                                                                                                                                                                                                                                                       |
| Consult transfer call is answered             | **Consult transfer Initiator Desktop**  
The agent is in the **Talking** state.  
The following buttons are enabled: Statistics, Chat ctrl, Bad Line Ctrl, Alternate, Reconnect, Conference complete.  
**Note** The Ready, NR, and Wrap-up buttons are enabled based on the agent desktop setting.                                                                                                                                                                                                                                                              |
|                                              | **Consult transfer Receiver Desktop**  
The agents is in the **Talking** state.  
The following buttons are enabled: Hold, Release, Supervisor Assist, Emergency Assist, Conference, transfer, Statistics, Chat ctrl, and Badline control.  
**Note** The Ready, Not Ready, and wrap-up buttons are enabled based on the agent desktop settings configured in the Configuration Manager.                                                                                                                                                                                                                      |
| Complete conference is done for the Transfer Initiator | The agent continues to be in the same state he was in, before talking the call.  
**Note** The agent is in the Wrap-up state, provided Wrap-up is configured.  
If Wrap-up was not configured, then the Ready and Not Ready buttons are enabled. |
<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Buttons enabled and Description</th>
</tr>
</thead>
</table>
| Agent does a Single Step Transfer | The agent continues to be in the same state he was in, before talking the call.  
**Note**  
The agent is in the Wrap-up state, provided Wrap-up is configured.  
If Wrap-up was not configured, then the **Ready** and **Not Ready** buttons are enabled.  
After Initiating a Consult Transfer or a Consult Conference, there are two calls on the agent desktop of the initiator of Transfer and Conference until the Transfer/Conference is complete. |
| The held call is selected for Consult Conference when the call rings on the conference agent desktop | The following buttons are enabled: Statistics, Chat ctl, BadLine Ctrl, and Reconnect.  
**Note**  
The Ready, NR, and Wrap-up buttons are enabled based on the agent desktop setting. |
| The held call is selected for Consult Transfer when the call rings on the conference agent desktop | The following buttons are enabled: Statistics, Chat ctl, BadLine Ctrl, and Reconnect.  
**Note**  
The Ready, NR, and Wrap-up buttons are enabled based on the agent desktop setting.  
The button enablement on desktops based on the agent desk settings depends on:  
• Work Mode incoming  
• Work Mode outgoing  
The button enablement described are common, the only difference being they are enabled on an incoming call or an outgoing call based on the settings. |
| Agent desk setting configuration is set to **Not Allowed** | The agent is not allowed to go to the Wrap-up state.  
No state transition buttons are enabled as long as the call is on the agent desktop.  
Buttons Enabled - Default behavior. |
| Agent desk setting configuration is set to **Optional** | The Ready, NR, and Wrap-up buttons are enabled after a call is answered and the agent is in the **Talking** state. |
Buttons enabled and Description

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Buttons enabled and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent desk setting configuration is set to Required/Required with data</td>
<td>No state change buttons are enabled when the agent is talking on a call. After the call, the agent state changes to <strong>Wrap-up</strong>. In the Wrap-up dialog box, if you select “Apply”, the Ready and NR buttons are enabled and the agent state changes based on the selection (button click) done by the agent. <strong>Note</strong> Supervisor assist and Emergency assist are not present on a supervisor desktop. The rest of the button enablement are only applicable for different scenarios described above.</td>
</tr>
</tbody>
</table>

These are basic call scenarios and only to be used for reference. Customized desktops can have different enablement behaviors.

**Button Controls**

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

**Grid Controls**

Grid controls include the AgentSelectCtl, CallAppearanceCtl, AgentStatisticsCtl and SkillGroupStatisticsCtl. The Grid Controls display data, select calls (see CallAppearanceCtl) or Agents (AgentSelectCtl), and in some cases allow you to enter data (for example, Callvariables in the CallAppearanceCtl). You can use CTI OS to configure the following grid properties. See the *CTI OS System Manager Guide for Cisco Unified ICM*. 

- Columns to display
- Column header
- Column width
- Column alignment
Supervisor Status Bar

The Supervisor Softphone has a status bar that appears at the bottom of the window. The supervisor status bar information is configurable at design time using the property pages. You can also set it programmatically at run time.

CTI OS ActiveX Control Descriptions

This section describes the CTI OS ActiveX softphone controls listed in Table 21: CTI OS ActiveX Controls, on page 119.

AgentGreetingCtl

The Agent Greeting control enables the Agent Greeting feature to be turned on or off by toggling the Agent Greeting button to the on or off state. Agent Greeting is automatically in the on state at login and the icon appears like this

Click the button to turn Agent Greeting off and the icon changes to

Click the button again to turn Agent Greeting on.

RecordGreetingCtl

The Record Agent Greeting button is used to record and manage Agent Greeting messages.

Recording an Agent Greeting is very similar to recording a personal message for your voice mail. To record a greeting, you must be logged in to your desktop software and in the Not Ready state. After you click the Agent Greeting Record button you hear a brief ring tone, after which you receive voice instructions for recording a greeting. Options include selecting a greeting type (if your contact center uses more than one greeting per agent), recording, playing back, and confirming whether to use the new greeting. There is also an option for listening to your existing greetings.

The RecordGreeting control is basically just a MakeCall request to the appropriate DN. It places a Make Call request to a hard coded “RecordAgentGreeting” Dialed Number and sets the Placement type (CPT_RECORD_AGENT_GREETING) to 7.

AgentStateCtl

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

• **Login Button**. Click the login button to allow the agent to select a connection profile (see the *CTI OS System Manager Guide for Cisco Unified ICM*), agent ID and instrument or other switch specific fields.
You can configure the fields displayed. The dialog box shows a login dialog box configured for Unified CCE. An agent logging in can select a connection profile for the **Connect To:** drop down box, enter an agent ID, password and instrument and click **OK** to send a Login request.

- **Logout Button.** Click the logout button to log out the currently logged in agent. For some switches, including Unified CCE, the agent must be in the not ready state to enable this button. If Reason Codes are supported on the switch and configured on Unified ICM, a reason code dialog box pops up as shown below.

Use this dialog box to select a reason code to send with the logout request. You can configure reason codes at CTI OS server

- **Ready Button.** Click the ready button to put the agent in ready state (ready to accept calls).

- **Not Ready Button.** Click the not ready button to put the agent in not ready state (Unified ICM does not route calls to an agent in the not ready state). If Reason Codes are supported on the switch and configured on Unified ICM, a reason code dialog box pops up as shown below.
Use this dialog box to select a reason code to send with the not_ready request. You can configure reason codes at the CTI OS Server.

- **Work Ready Button.** Click this button to put the agent in the work ready or wrapup state. The behavior of this button depends on the wrapup mode support of the switch. On Unified CCE, the behavior is controlled by Unified ICM AgentDeskSettings (for more information, see Administration Guide for Cisco Unified Contact Center Enterprise).

- **Work Not Ready Button.** Click this button to put the agent in the work not ready or wrapup state. The behavior of this button depends on the wrapup mode support of the switch. On Unified CCE, the behavior is controlled by Unified ICM AgentDeskSettings (for more information, see the Administration Guide for Cisco Unified Contact Center Enterprise).

### Related Methods

The following methods may be of interest to users of the agent state ActiveX control.

**ReasonCodeState**

This version of the ReasonCodeState method returns the enumReasonCodeState value.

**Syntax**

**COM**

```csharp
HRESULT ReasonCodeState([out, retval] enumReasonCodeState *pVal)
```

**VB**

```vbnet
ReasonCodeState() As AgentStateCtlLib.enumReasonCodeState
```

**.NET**

```csharp
AgentStateCtlLib.enumReasonCodeState ReasonCodeState()
```
**Parameters**

None

**Return Value**

Return value is enumReasonCodeState (this returns an Integer type).

### ReasonCodeState

This version of the ReasonCodeState method sets the enumReasonCodeState value.

**Syntax**

**COM**

```c
```

**VB**

```vbnet
ReasonCodeState = AgentStateCtlLib.enumReasonCodeState
```

**.NET**

```csharp
ReasonCodeState = AgentStateCtlLib.enumReasonCodeState
```

**Parameters**

None

**Return Value**

None.

Following are the enumerated values for ReasonCodeState:

```c
typedef enum {
    eNotUsed,
    eRequested,
    eRequired,
} enumReasonCodeState;
```

### Related Events

The agent state control handles the following events.

**OnAgentStateChanged**

The OnAgentStateChanged message is generated when an agent state change event is fired.

**Syntax**

**COM**

```c
HRESULT OnAgentStateChanged([in] LPDISPATCH vEventParam)
```

**Parameters**

**vEventParam**

Event fired to change the Agent state.
Return Value
None.

OnCtlEnabledChanged
The OnCtlEnabledChanged message is generated when control enabled is changed.

Syntax
COM

HRESULT OnCtlEnabledChanged(BOOL enabled)

Parameters
enabled
This is a control enabled changed value and returns a Boolean value.

OnEnableControlReceived
The OnEnableControlReceived message is generated when button enablement is changed.

Syntax
COM

void OnEnableControlReceived(BOOL enabled)

Parameters
enabled
This is a control enabled changed value and returns a Boolean value.

Following are the Button enablement masks return by OnEnableControlReceived method.

AgentSelectCtl

Table 23: OnEnableControlReceived Button Enablement Masks

<table>
<thead>
<tr>
<th>m_eButtonType</th>
<th>nButtonMask</th>
</tr>
</thead>
<tbody>
<tr>
<td>eLoginAgentBtn</td>
<td>ENABLE_LOGIN</td>
</tr>
<tr>
<td>eLoginSupervisorBtn</td>
<td>ENABLE_LOGIN</td>
</tr>
<tr>
<td>eLogoutBtn</td>
<td>ENABLE_LOGOUT or ENABLE_LOGOUT_WITH_REASON</td>
</tr>
<tr>
<td>eReadyBtn</td>
<td>ENABLE_READY</td>
</tr>
<tr>
<td>eNotReadyBtn</td>
<td>ENABLE_NOTREADY or ENABLE_NOTREADY_WITH_REASON</td>
</tr>
<tr>
<td>eWorkReadyBtn</td>
<td>ENABLE_WORKREADY</td>
</tr>
</tbody>
</table>
The agent select control is used for supervising agents and becomes active if the currently logged in agent is a supervisor. When a supervisor logs in, this grid based control displays all agent team members of a supervisor (configured by Unified ICM), including agent name, AgentID, AgentState, TimeInState, and SkillGroups. The TimeInState column resets in real-time as the agents change state. If an agent remains in a state for more than 10 minutes, the TimeInState figure displays in red.

**Figure 22: Agent Select Grid Populated with Sample Data**

<table>
<thead>
<tr>
<th>Name</th>
<th>AgentID</th>
<th>State</th>
<th>Time In State</th>
<th>Skillgroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>kad 511</td>
<td>5111</td>
<td>Unknown</td>
<td>0.50</td>
<td>2</td>
</tr>
<tr>
<td>kad 512</td>
<td>5112</td>
<td>Unknown</td>
<td>0.50</td>
<td>2</td>
</tr>
<tr>
<td>CSE 301</td>
<td>301</td>
<td>T3/2</td>
<td>0.51</td>
<td>2/3/4</td>
</tr>
<tr>
<td>kad 5102</td>
<td>5102</td>
<td>NoReady</td>
<td>0.51</td>
<td>2</td>
</tr>
</tbody>
</table>

The agent select control handles the following events:

- **OnNewTeamMember.** Informs the supervisor of a new team member or a team member change. This updates a row in the agentselect grid (add/remove agent).
- **OnMonitoredAgentStateChange.** Informs the supervisor of an agent state change. The new agentstate displays in the State column and the TimeInState Column is set to zero.
- **OnAgentInfo Event.**

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

### Methods

**Table 24: Available Methods for AgentSelectCtl**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_UserDefinedCell</td>
<td>If the column type is user defined, gets the text from the requested cell.</td>
</tr>
<tr>
<td>GetCellText</td>
<td>Gets the text from the requested cell in requested row.</td>
</tr>
<tr>
<td>GetColumnInfo</td>
<td>Gets the information about the requested column.</td>
</tr>
<tr>
<td>GetSelectedRow</td>
<td>Gets the selected row index.</td>
</tr>
<tr>
<td>SelectRow</td>
<td>Sets the requested row as selected.</td>
</tr>
<tr>
<td>set_ColumnHeader</td>
<td>Sets the column header of requested column with given text.</td>
</tr>
</tbody>
</table>
### Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_ColumnType</td>
<td>Sets the column type of requested column with given value.</td>
</tr>
<tr>
<td>set_ColumnWidth</td>
<td>Sets the column width of requested column with given value.</td>
</tr>
<tr>
<td>set_UserDefinedCell</td>
<td>Sets the given text into the requested cell.</td>
</tr>
<tr>
<td>SetColumnInfo</td>
<td>Sets the given information for the requested column.</td>
</tr>
</tbody>
</table>

### get_UserDefinedCell

If the column type is user defined, gets the text from the requested cell.

#### Syntax

**COM**

```c
HRESULT UserDefinedCell(short nIndex, [out, retval] BSTR *pVal)
```

**VB**

```vb
get_UserDefinedCell(nIndex As Short) As String
```

**.NET**

```csharp
System.String get_UserDefinedCell(System.Int16 nIndex)
```

#### Parameters

**nIndex**

This is a cell index number and an input parameter as type Short.

#### Return Value

Return type is String.

If the requested cell is not user defined type, it throws an Invalid Argument error.

### GetCellText

Gets the text from the requested cell in requested row.

#### Syntax

**COM**

```c
HRESULT GetCellText([in] int nRow, [in] int nCol, [out,retval] BSTR* bstrContent)
```

**VB**

```vb
GetCellText(nRow As Integer, nCol As Integer) As String
```

**.NET**

```csharp
```
Parameters

nRow
This is a row index number and an input parameter as type Integer.

nCol
This is a column index number and an input parameter as type Integer.

Return Values
Return type is String.

GetColumnInfo

Gets the information about the requested column.

Syntax

COM
HRESULT GetColumnInfo([in] short nCol, [out] long *plColType, [out] int *iColWidth, [out] int *iColTextAlign, [out] BSTR *bstrColTitle)

VB
GetColumnInfo(nCol As Short, ByRef plcoltype As Integer, ByRef icolwidth As Integer, ByRef bstrcoltitle As String)

.NET

Parameters

nCol
This is a column index number and an input parameter as type Short.

plcoltype
This is a column type value and an output parameter as type Integer.

icolwidth
This is a column width value and an output parameter as Integer.

bstrcoltitle
This is a column title text and an output parameter as type String.

Return Values
None.

GetSelectedRow

 Gets the selected row index.
Syntax

COM
HRESULT GetSelectedRow([out, retval] int *nRow)

VB
GetSelectedRow() As Integer

.NET
System.Int32 GetSelectedRow()

Parameters
None

Return Values
Return type is Integer.

SelectRow

Sets the requested row as selected.

Syntax

COM
HRESULT SelectRow([in] int nRow, [out, retval] VARIANT_BOOL * bStatus)

VB
SelectRow(nRow As Integer) As Boolean

.NET
System.Boolean SelectRow(System.Int32 nRow)

Parameters
nRow
This is a row index number and an input parameter as type Integer.

Return Values
Return type is Boolean.

set_ColumnHeader

Sets the column header of requested column with given text.

Syntax

COM
HRESULT ColumnHeader(short nCol, [in] BSTR newVal)

VB
set_ColumnHeader(nCol As Short, newVal As String)

.NET
set_ColumnHeader(System.Int16 nCol, System.String newVal)
Parameters

nCol
This is a column index number and an input parameter as type Short.

newVal
This is a user passing header text and an input parameter as type String.

Return Values
None.

set_ColumnType

Sets the column type of requested column with given value.

Syntax

COM
HRESULT ColumnType(short nCol, [in] short newVal)

VB
set_ColumnType(nCol As Short, newVal As Short)

.NET
set_ColumnType(System.Int16 nCol, System.Int16 newVal)

Parameters

nCol
This is a column index number and an input parameter as type Short.

newVal
This is a user passing column type value and an input parameter as type Short.

Return Values
None.

set_ColumnWidth

Sets the column width of requested column with given value.

Syntax

COM
HRESULT ColumnWidth(short nCol, [in] short newVal)

VB
set_ColumnWidth(nCol As Short, newVal As Short)

.NET
set_ColumnWidth(System.Int16 nCol, System.Int16 newVal)
Parameters
nCol
This is a column index value and an input parameter as type Short.

newVal
This is a user passing column width value and an input parameter as type Short.

Return Values
None.

set_UserDefinedCell
Sets the given text into the requested cell.

Syntax
COM
HRESULT UserDefinedCell(short nIndex, [in] BSTR newVal);

VB
set_UserDefinedCell(nindex As Short, newVal As String)

.NET
set_UserDefinedCell(System.Int16 nindex, System.String newVal)

Parameters
nindex
This is a cell index number and an input parameter as type Short.

newVal
This is a user passing text and an input parameter as type String.

Return Values
None.

SetColumnInfo
Sets the given information for the requested column.

Syntax
COM

VB
SetColumnInfo(nCol As Short, iColType As Integer, iColWidth As Integer, iColTextAlign As Integer, bstrColTitle As String)
.NET
    SetColumnInfo(System.Int16 nCol, System.Int32 iColType, System.Int32 iColWidth,
    System.Int32 iColTextAlign, System.String bstrColTitle)

Parameters
nCol
    This is a column index number and an input parameter as type Short.
iColType
    This is a column type value and an input parameter as type Integer.
iColWidth
    This is a column width value and an input parameter as type Integer.
iColTextAlign
    This is a column text align value and an input parameter as type Integer.
bstrColTitle
    This is a column title text and an input parameter as type String.

Return Values
None.

AgentStatisticsCtl

The AgentStatistics control is a grid based control displaying Unified ICM agent real time statistics. You can configure the displayed columns at CTI OS server (for more information, see the CTI OS System Manager Guide for Cisco Unified ICM). Also, you can adjust the update interval, which defaults to 10 seconds.

Figure 23: Agent Statistics Grid

Methods

Table 25: Available methods for AgentStatisticsCtl

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_UserDefinedCell</td>
<td>If the column type is user defined, gets the text from the requested cell.</td>
</tr>
<tr>
<td>GetCellText</td>
<td>Gets the text from the requested cell in requested row.</td>
</tr>
<tr>
<td>GetColumnInfo</td>
<td>Gets the information about the requested column.</td>
</tr>
</tbody>
</table>
### Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_ColumnHeader</td>
<td>Sets the column header of requested column with given text.</td>
</tr>
<tr>
<td>setColumnType</td>
<td>Sets the column type of requested column with given value.</td>
</tr>
<tr>
<td>set_ColumnWidth</td>
<td>Sets the column width of requested column with given value.</td>
</tr>
<tr>
<td>set_UserDefinedCell</td>
<td>Sets the given text into the requested cell.</td>
</tr>
<tr>
<td>SetColumnInfo</td>
<td>Sets the given information for the requested column.</td>
</tr>
</tbody>
</table>

### get_UserDefinedCell

If the column type is user defined, gets the text from the requested cell.

**Syntax**

COM: HRESULT UserDefinedCell(short nIndex, [out, retval] BSTR *pVal)

VB: get_UserDefinedCell(nIndex As Short) As String

.NET: System.String get_UserDefinedCell(System.Int16 nIndex)

**Parameters**

nIndex

This is a cell index number and an input parameter as type Short.

**Return Value**

Return type is String.

If the requested cell is not user defined type, it throws an Invalid Argument error.

### GetCellText

Gets the text from the requested cell in requested row.

**Syntax**


VB: GetCellText(nRow As Integer, nCol As Integer) As String


**Parameters**

nRow

This is a row index number and an input parameter as type Integer.

nCol

This is a column index number and an input parameter as type Integer.

**Return Value**
Return type is String.

**GetColumnInfo**

Gets the information about the requested column.

**Syntax**

COM: HRESULT GetColumnInfo([in] short nCol, [out] long *plColType, [out] int *iColWidth, [out] int *iColTextAlign, [out] BSTR *bstrColTitle)

VB: GetColumnInfo(nCol As Short, ByRef plcoltype As Integer, ByRef icolwidth As Integer, ByRef bstrcoltitle As String)


**Parameters**

nCol
This is a column index number and an input parameter as type Short.

plcoltype
This is a column type value and an output parameter as type Integer.

icolwidth
This is a column width value and an output parameter as Integer.

bstrcoltitle
This is a column title text and an output parameter as type String.

**Return Value**

None.

**set_ColumnHeader**

Sets the column header of requested column with given text.

**Syntax**

COM: HRESULT ColumnHeader(short nCol, [in] BSTR newVal)

VB: set_ColumnHeader(nCol As Short, newVal As String)

.NET: set_ColumnHeader(System.Int16 nCol, System.String newVal)

**Parameters**

nCol
This is a column index number and an input parameter as type Short.

newVal
This is a user passing header text and an input parameter as type String.

**Return Value**

None.
**setColumnType**

Sets the column type of requested column with given value.

**Syntax**

COM: HRESULT ColumnType(short nCol, [in] short newVal)

VB: setColumnType(nCol As Short, newVal As Short)

.NET: setColumnType(System.Int16 nCol, System.Int16 newVal)

**Parameters**

- **nCol**
  This is a column index number and an input parameter as type Short.

- **newVal**
  This is a user passing column type value and an input parameter as type Short.

**Return Value**

None.

**setColumnWidth**

Sets the column width of requested column with given value.

**Syntax**

COM: HRESULT ColumnWidth(short nCol, [in] short newVal)

VB: setColumnWidth(nCol As Short, newVal As Short)


**Parameters**

- **nCol**
  This is a column index value and an input parameter as type Short.

- **newVal**
  This is a user passing column width value and an input parameter as type Short.

**Return Value**

None.

**setUserDefinedCell**

Sets the given text into the requested cell.

**Syntax**

COM: HRESULT UserDefinedCell(short nIndex, [in] BSTR newVal);

VB: setUserDefinedCell(nIndex As Short, newVal As String)

.NET: setUserDefinedCell(System.Int16 nIndex, System.String newVal)

**Parameters**

- **nIndex**
  This is an index number for the requested cell.
This is a cell index number and an input parameter as type Short.

newVal

This is a user passing text and an input parameter as type String.

**Return Value**

None.

### SetColumnInfo

Sets the given information for the requested column.

**Syntax**


VB: SetColumnInfo(nCol As Short, iColType As Integer, iColWidth As Integer, iColTextAlign As Integer, bstrColTitle As String)


**Parameters**

- **nCol**: This is a column index number and an input parameter as type Short.
- **iColType**: This is a column type value and an input parameter as type Integer.
- **iColWidth**: This is a column width value and an input parameter as type Integer.
- **iColTextAlign**: This is a column text align value and an input parameter as type Integer.
- **bstrColTitle**: This is a column title text and an input parameter as type String.

**Return Value**

None.

### AlternateCtl

**Figure 24: AlternateCtl**

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

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

*Figure 25: Answer Icon:*

![Answer Icon](image)

*Figure 26: Release Icon:*

![Release Icon](image)

**BadLineCtl**

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

**CallAppearanceCtl**

The CallAppearance Control is a grid based control displaying call information, including call status and call context data (for example, CallVariable1 through CallVariable10 and ECC variables).

*Figure 28: CallAppearance Control Displaying Two Calls*

<table>
<thead>
<tr>
<th>CallID</th>
<th>CallStatus</th>
<th>DNIS</th>
<th>ANI</th>
<th>CallType</th>
<th>DialedNu...</th>
<th>ModIf...</th>
<th>Wr</th>
</tr>
</thead>
<tbody>
<tr>
<td>16777884</td>
<td>Hold</td>
<td>5202</td>
<td>OTHER_IN</td>
<td>OTHER</td>
<td>5202</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16777886</td>
<td>Talking</td>
<td>5201</td>
<td>OTHER_IN</td>
<td>OTHER</td>
<td>5201</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each incoming or outgoing call appears in one row in the grid. When a call first arrives, it usually shows a status of “Ringing” until it is answered. You can answer a call by a double click in the grid, similar to a click on the Answer Button. You can edit some columns in the CallAppearance grid if so configured (for example, the Columns displaying CallVariables) by selecting the cell you want to edit.

The grid can display multiple calls (see above). If the grid is displaying multiple calls, a user can click and select a call anywhere on the row where the call appears. This highlights the whole row displaying this call (for example, in the above figure the call with ID 16777886 is currently selected). Any button controls (for example, Answer, Release, Hold) enable or disable themselves based on the state the newly selected call is in.

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

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

Related Methods

The following methods may be of interest to users of the call appearance control.

Answer

For more information, see Call Object, on page 389

GetValueInt

For more information, see CtiOs Object, on page 291

GetValueString

For more information, see CtiOs Object, on page 291

Related Events

The call appearance control handles the following events.

OnSetCurrentCallAppearance

The OnSetCurrentCallAppearance event is generated when the current call appearance object is changed.

Syntax

```c
void OnSetCurrentCallAppearance([in] IDispatch * pCall);
```

Parameters

pCall

A Pointer to ICall COM Call object (pCall is a pointer to ICall).

Return Value

None.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCellText</td>
<td>Gets the text from the requested cell in requested row.</td>
</tr>
<tr>
<td>GetSelectedRow</td>
<td>Gets the selected row index.</td>
</tr>
</tbody>
</table>
### Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SelectRow</td>
<td>Sets the requested row as selected.</td>
</tr>
<tr>
<td>set_ColumnECCName</td>
<td>Sets the column ECC name of requested column with given text.</td>
</tr>
<tr>
<td>set_ColumnECCOffset</td>
<td>Sets the column Offset value of requested column with given value.</td>
</tr>
<tr>
<td>set_ColumnHeader</td>
<td>Sets the column header of requested column with given text.</td>
</tr>
<tr>
<td>set_ColumnWidth</td>
<td>Sets the column width of requested column with given value.</td>
</tr>
<tr>
<td>SetCellText</td>
<td>Sets the given text to the requested cell in requested row.</td>
</tr>
</tbody>
</table>

### GetCellText

Gets the text from the requested cell in requested row.

**Syntax**

**COM:** HRESULT GetCellText([in] int nRow, [in] int nCol, [out,retval] BSTR* bstrContent)

**VB:** GetCellText(nRow As Integer, nCol As Integer) As String

**.NET:** System.String GetCellText(System.Int16 nRow, System.Int16 nCol)

**Parameters**

nRow
- This is a row index number and an input parameter as type Integer.

nCol
- This is a column index number and an input parameter as type Integer.

**Return Value**
- Return type is String.

### GetSelectedRow

Gets the selected row index.

**Syntax**

**COM:** HRESULT GetSelectedRow([out,retval] int *nRow)

**VB:** GetSelectedRow() As Integer

**.NET:** System.Int32 GetSelectedRow()

**Parameters**

None.

**Return Value**
**SelectRow**

Sets the requested row as selected.

**Syntax**

COM: HRESULT SelectRow([in] int nRow, [out,retval] VARIANT_BOOL * bStatus)

VB: SelectRow(nRow As Integer) As Boolean

.NET: System.Boolean SelectRow(System.Int32 nRow)

**Parameters**

nRow

This is a row index number and an input parameter as type Integer.

**Return Value**

Return type is Boolean.

---

**set_ColumnECCName**

Sets the column ECC name of requested column with given text.

**Syntax**

COM: HRESULT ColumnECCName(short nCol, [in] BSTR newVal)

VB: set_ColumnECCName(nCol As Short, newVal As String)

.NET: set_ColumnECCName(System.Int16 nCol, System.String newVal)

**Parameters**

nCol

This is a column index number and an input parameter as type Short.

newVal

This is a user passing ECC Name text and an input parameter as type String.

**Return Value**

None.

---

**set_ColumnECCOffset**

Sets the column Offset value of requested column with given value.

**Syntax**

COM: HRESULT ColumnECCOffset(short nCol, [in] short nNewValue)

VB: set_ColumnECCOffset(nCol As Short, nNewValue As Short)


**Parameters**

nCol
This is a column index number and an input parameter as type Short.
nNewVal
This is a user passing column width value and an input parameter as type Short.

**ReturnValue**
None.

### set_ColumnHeader

Sets the column header of requested column with given text.

**Syntax**

COM: HRESULT ColumnHeader(short nCol, [in] BSTR newVal)

VB: set_ColumnHeader(nCol As Short, newVal As String)

.NET: set_ColumnHeader(System.Int16 nCol, System.String newVal)

**Parameters**

nCol
This is a column index number and an input parameter as type Short.
nNewVal
This is a user passing header text and an input parameter as type String.

**ReturnValue**
None.

### set_ColumnWidth

Sets the column width of requested column with given value.

**Syntax**

COM: HRESULT ColumnWidth(short nCol, [in] short newVal)

VB: set_ColumnWidth(nCol As Short, newVal As Short)


**Parameters**

nCol
This is a column index value and an input parameter as type Short.

newVal
This is a user passing column width value and an input parameter as type Short.

**ReturnValue**
None.

### SetCellText

Sets the given text to the requested cell in requested row.
Syntax


VB: SetCellText(nRow As Integer, nCol As Integer, bstrContent As String) As Boolean


Parameters

nRow
This is a row index number and an input parameter as type Integer.
nCol
This is a column index number and an input parameter as type Integer.
bstrContent
This is a user passing cell text and an input parameter as type String.

Return Value

Return type is Boolean.

ChatCtl

The Chat Control provides a UI to formulate and send text messages to a supervisor or (if allowed) other agents. The chat privileges are configurable at CTI OS server (for more information, see CTI OS System Manager Guide for Cisco Unified ICM).

Figure 29: Chat Control

You can specify an AgentID in the **Send to AgentID** field and then enter a message in the **Edit Outgoing Message** box. Click the **Send** Button to send the message. Incoming messages appear in the **Message Display**. Click the **Clear** button to clear the display.

The ChatCtl does not implement a button directly, but you can link it to a button through Visual Basic so that a click on the button pops up the ChatCtl.
## Methods

**Table 27: Available methods for ChatCtl**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetAddressee</td>
<td>Gets the current Addresssee from the Send to Agent ID Combo box.</td>
</tr>
<tr>
<td>GetAllChatMessages</td>
<td>Gets the all chat messages from the Message Display Text Area.</td>
</tr>
<tr>
<td>GetChatMessageText</td>
<td>Gets the chat message from the Edit Outgoing Message Text Area.</td>
</tr>
<tr>
<td>OnMsgReceived</td>
<td>When message received from an Agent, appends the received message to the Message Display Text Area.</td>
</tr>
<tr>
<td>SendChatMessage</td>
<td>Sends the chat message to current Addresssee in the Send to Agent ID Combo box.</td>
</tr>
<tr>
<td>SetAddressee</td>
<td>Sets the current Addresssee to the Send to Agent ID Combo box.</td>
</tr>
<tr>
<td>SetChatMessageText</td>
<td>Sets the chat message to the Edit Outgoing Message Text Area.</td>
</tr>
</tbody>
</table>

### GetAddressee

Gets the current Addresssee from the Send to Agent ID Combo box.

**Syntax**

COM: HRESULT GetAddressee ([out,retval] BSTR* addresssee)

VB: GetAddressee()AsString

.NET: System.String GetAddressee()

**Parameters**

None.

**Return Value**

Return type is String.

### GetAllChatMessages

Gets the all chat messages from the Message Display Text Area.

**Syntax**

COM: HRESULT GetAllChatMessages ([out, retval] BSTR* Messages)

VB: GetAllChatMessages() As String

.NET: System.String GetAllChatMessages()

**Parameters**


GetChatMessageText

Gets the chat message from the Edit Outgoing Message Text Area.

Syntax


VB: GetChatMessageText() As String

.NET: System.String GetChatMessageText()

Parameters

None.

Return Value

Return type is String.

OnMsgReceived

When message is received from an Agent, appends the received message to the Message Display Text Area.

Syntax

COM: HRESULT OnMsgReceived ([in] BSTR from, [in] BSTR msg)

VB: OnMsgReceived(from As String, msg As String)

.NET: OnMsgReceived(System.String from, System.String msg)

Parameters

from

This is an Agent ID, who sends the message and is an input parameter as type String.

msg

This is a message text received from an Agent and is an input parameter as type String.

Return Value

None.

SendChatMessage

Sends the chat message to current Addressee in the Send to Agent ID Combo box.

Syntax


VB: SendChatMessage(addressee As String, msg As String)

.NET: SendChatMessage (System.String addressee, System.String msg)

Parameters
SetAddressee

Sets the current Addressee to the Send to Agent ID Combo box.

Syntax

COM: HRESULT SetAddressee ([in] BSTR address)
VB: SetAddressee(address As String)
.NET: SetAddressee(System.String address)

Parameters

address
This is as Agent ID, who receives the message and is an input parameter as type String.

Return Value
None.

SetChatMessageText

Sets the chat message to the Edit Outgoing Message Text Area.

Syntax

COM: HRESULT SetChatMessageText ([in] BSTR MessageText)
VB: SetChatMessageText(messageText As String)
.NET: SetChatMessageText(System.String messageText)

Parameters

messageText
This is an outgoing message text and is an input parameter as type String.

Return Value
None.

ConferenceCtl

You can use the conference control to create a conference call. You can do this in either single step or consultative mode.
Depending on the call status, selecting the Conference button once brings up the dialog box shown in the figure below (see also MakeCall dialog):

**Figure 32: Conference Init Dialog**

This dialog box is similar to the Make Call dialog box; you can initiate a consultative Conference (Conf Init) or place a Single Step Conference call.

Enter the number you wish to dial by either typing it into the Number to Dial text box or by clicks on the displayed keypad. After you enter the number you can click Conf Init to place a consultative conference call or Single Step to initiate a single step conference. This closes this dialog. If you choose to place a consultative call, the conference button changes to Conference Complete. You must click this button to complete the conference after talking to the consult agent.

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

The More button brings up an additional section of the dialog displaying all Call Variables with any values set in the original call. The agent can double click the appropriate line in the Value column to change or add values to send with the consult call (see the figure below).
EmergencyAssistCtl

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

HoldCtl

The HoldCtl is a button that provides a UI for sending hold and retrieve call requests. You can set the behavior (hold or retrieve) via the ButtonType property set from the Property Page, as described under AgentState controls.
You use the MakeCallCtl to place calls and to generate DTMF tones. When you click this button it brings up the dialing pad dialog box to enter data and place a makecall request.

Enter the number you wish to dial by either typing it into the **Number to Dial** text box or click the numbers on the displayed keypad. After you enter the number you can click **Make Call** to send the MakeCall request.

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

You can enter values for CallVariable1 through CallVariable10 and ECC Call Variables via the Dial Dialog. Click the **More** button on the dialog extends to display a grid listing all possible Call Variables. You can enter a value for each of these variables by double clicking the appropriate line in the Value column.
If the agent is on a call while selecting the **Make Call** button, the dialpad appears without the MakeCall feature. The agent can then use the dialpad to play DTMF tones.

**ReconnectCtl**

The **ReconnectCtl** is a button control allowing the agent to send a Reconnect Call request. Reconnect is a useful feature during a consult call. If an agent has Call A held and Call B active, reconnect hangs up Call B and makes Call A active. In a consult call scenario, reconnect hangs up the consult call and returns to the original call.

**SkillgroupStatisticsCtl**

The **SkillgroupStatisticsCtl** is a grid based control displaying Unified ICM real time SkillGroup statistics. You can configure the displayed columns at CTI OS server (for more information see **CTI OS System Manager Guide for Cisco Unified ICM**). You can configure the update interval, which defaults to 10 seconds.

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

![Figure 41: SkillgroupStatisticsCtl Displaying Sample Data for Three SkillGroups](image)

**Methods**

**Table 28: Available methods for SkillgroupStatisticsCtl**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_UserDefinedCell</td>
<td>If the column type is user defined, gets the text from the requested cell.</td>
</tr>
<tr>
<td>GetCellText</td>
<td>Gets the text from the requested cell in requested row.</td>
</tr>
<tr>
<td>GetColumnInfo</td>
<td>Gets the information about the requested column.</td>
</tr>
<tr>
<td>set_ColumnHeader</td>
<td>Sets the column header of requested column with given text.</td>
</tr>
<tr>
<td>set_ColumnType</td>
<td>Sets the column type of requested column with given value.</td>
</tr>
<tr>
<td>set_ColumnWidth</td>
<td>Sets the column width of requested column with given value.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>set_UserDefinedCell</td>
<td>Sets the given text into the requested cell.</td>
</tr>
<tr>
<td>SetColumnInfo</td>
<td>Sets the given information for the requested column.</td>
</tr>
</tbody>
</table>

**get_UserDefinedCell**

If the column type is user defined, gets the text from the requested cell.

**Syntax**

COM: HRESULT UserDefinedCell(short nIndex, [out, retval] BSTR *pVal)

VB: get_UserDefinedCell(nIndex As Short) As String

.NET: System.String get_UserDefinedCell(System.Int16 nIndex)

**Parameters**

nIndex

This is a cell index number and an input parameter as type Short.

**Return Value**

Return type is String.

If the requested cell is not user defined type, it throws an Invalid Argument error.

**GetCellText**

Gets the text from the requested cell in requested row.

**Syntax**


VB: GetCellText(nRow As Integer, nCol As Integer) As String


**Parameters**

nRow

This is a row index number and an input parameter as type Integer.

nCol

This is a column index number and an input parameter as type Integer.

**Return Value**

Return type is String.

**GetColumnInfo**

Gets the information about the requested column.

**Syntax**

COM: HRESULT GetColumnInfo([in] short nCol, [out] long *pColType, [out] int *iColWidth, [out] int *iColTextAlign, [out] BSTR *bstrColTitle)
VB: GetColumnInfo(nCol As Short, ByRef plcoltype As Integer, ByRef icolwidth As Integer, ByRef bstrcoltitle As String)

**Parameters**

nCol
This is a column index number and an input parameter as type Short.

plcoltype
This is a column type value and an output parameter as type Integer.

icolwidth
This is a column width value and an output parameter as Integer.

bstrcoltitle
This is a column title text and an output parameter as type String.

**Return Value**
None.

**set_ColumnHeader**
Sets the column header of requested column with given text.

**Syntax**
COM: HRESULT ColumnHeader(short nCol, [in] BSTR newVal)
VB: set_ColumnHeader(nCol As Short, newVal As String)
.NET: set_ColumnHeader(System.Int16 nCol, System.String newVal)

**Parameters**

nCol
This is a column index number and an input parameter as type Short.

newVal
This is a user passing header text and an input parameter as type String.

**Return Value**
None.

**setColumnType**
Sets the column type of requested column with given value.

**Syntax**
COM: HRESULT ColumnType(short nCol, [in] short newVal)
VB: setColumnType(nCol As Short, newVal As Short)
.NET: setColumnType(System.Int16 nCol, System.Int16 newVal)
Parameters

nCol
This is a column index number and an input parameter as type Short.

newVal
This is a user passing column type value and an input parameter as type Short.

Return Value
None.

set_ColumnWidth

Sets the column width of requested column with given value.

Syntax
COM: HRESULT ColumnWidth(short nCol, [in] short newVal)
VB: set_ColumnWidth(nCol As Short, newVal As Short)

Parameters

nCol
This is a column index value and an input parameter as type Short.

newVal
This is a user passing column width value and an input parameter as type Short.

Return Value
None.

set_UserDefinedCell

Sets the given text into the requested cell.

Syntax
COM: HRESULT UserDefinedCell(short nIndex, [in] BSTR newVal);
VB: set_UserDefinedCell(nindex As Short, newVal As String)

Parameters

nindex
This is a cell index number and an input parameter as type Short.

newVal
This is a user passing text and an input parameter as type String.

Return Value
None.
SetColumnInfo

Sets the given information for the requested column.

**Syntax**


VB: SetColumnInfo(nCol As Short, iColType As Integer, iColWidth As Integer, iColTextAlign As Integer, bstrColTitle As String)


**Parameters**

nCol

This is a column index number and an input parameter as type Short.

iColType

This is a column type value and an input parameter as type Integer.

iColWidth

This is a column width value and an input parameter as type Integer.

iColTextAlign

This is a column text align value and an input parameter as type Integer.

bstrColTitle

This is a column title text and an input parameter as type String.

**Return Value**

None.

StatusBarCtl

The CTI OS statusbar control displays information about the logged in agent as well as CTI OS specific details (see the following figure).

**Figure 42: StatusBar Control Displaying Sample Data**

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

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

**SupervisorOnlyCtl**

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

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

![Logout Icon]

**Set Monitored Agent Ready**: Forces the currently monitored agent from the “not ready” state into the ready state:

![Set Ready Icon]

**Barge-In**: Allows the supervisor to participate in the currently monitored call. The currently monitored call is selected via the CallAppearanceCtl (in monitor mode). Barge-in is really a conference on behalf of the monitored agent:

![Barge-In Icon]

**Intercept**: You can only apply Intercept on a previously barged in call. The monitored agent is dropped out of the call and the supervisor is left with the customer in a call:

![Intercept Icon]

Together with the AgentSelectCtl and the CallAppearanceCtl (in monitor mode), you use the SupervisorOnlyCtl in the CTI OS Supervisor Desktop application to build the Agent Real Time Status window, as shown in the following figure.
Figure 43: Supervisor Softphone Agent-RealTime Status Window

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

Start Silent Monitor: Initiates a silent monitor session with the currently monitored agent:

Stop Silent Monitor: Terminates the currently ongoing silent monitored session:

RecordCtl

The RecordCtl is a button that provides UI for Call Recording requests (start/stop recording). The requests are forwarded to CTI Server and are handled by a configured call recording service. To record a call you must select a current call (e.g. via the CallAppearanceCtl). After you click the record button it turns into record stop button.

Icon for Record Start:

Icon for Record Stop:

TransferCtl

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

![Icon for TransferInit](image)

Icon for Transfer Complete:

![Icon for Transfer Complete](image)

Depending on call status, selecting the Transfer button once brings up the dialog box shown in the following figure (see also MakeCall dialog box):

*Figure 44: Dial Dialog*

![Dial Dialog](image)

This dialog box is similar to the Make Call dialog box. You can initiate a consultative Transfer (Transfer Init) or place a Single Step Transfer call.

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

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

The **More** button brings up an additional section of the dialog box displaying all CallVariables and any values set in the original call. The agent can change or add values to send with the consult call by double clicking the appropriate line in the Value column (see the following figure).
The Silent Monitor StandAlone ActiveX Control

The Silent Monitor StandAlone ActiveX Control provides an interface for easy integration with the CTI OS Silent Monitor functionality. You can use the ComObject in Visual Basic 6.0 as well as other host containers. This section demonstrates the use of this control in Visual Basic 6.0.

Note

The Silent Monitor StandAlone ComObject is supported for use on Unified CCE only.

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

```idl
interface ISilentMonitor : IDispatch
{
    [id(1), helpstring("method Connect to CTIOS")]
    HRESULT Connect ([in] IArguments * args, [out] int* returnvalue);

    [id(2), helpstring("method Disconnect to CTIOS")]
    HRESULT Disconnect (/*[in]
    IArguments * args, [out] int* returnvalue*/);

    [id(3), helpstring("method StartMonitoring to CTIOS")]
    HRESULT StartMonitoring ([in] IArguments * args, [out] int* returnvalue);

    [id(4), helpstring("method StopMonitoring to CTIOS")]
    HRESULT StopMonitoring ([in] IArguments * args, [out] int* returnvalue);
};
```

Connect

The Connect method establishes a Monitor Mode Session with the specified CTI OS Server. The syntax and parameters are the same as the CTI OS session object Connect method (for more information, see Returns, on page 312 under AddEventListener method).
Disconnect

The Disconnect method disconnects an established session. This method has no required parameters. For more information about syntax and optional parameters, see CreateSilentMonitorManager, on page 315 in Session Object, on page 307.

StartMonitoring

The StartMonitoring method starts a Silent Monitor Session. The StartMonitoring Arguments array contains the following parameters:

Table 29: StartMonitoring Arguments Array Parameters

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

Note

If a pointer to the Agent object is available (for example, a m_MonitoredAgent), you can retrieve the PeripheralID via m_MonitoredAgent.GetValueInt("PeripheralID").

StopMonitoring

The StopMonitoring method stops a Silent Monitor Session. The StopMonitoring Arguments array contains the same parameters as the StartMonitoring method (StartMonitoring, on page 164).

SilentMonitor Com Object Events

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

dispinterface _ISilentMonitorCtlEvents
{
    properties:
    methods:
        [id(1)] void OnConnection([in] IArguments *pIArguments);
        [id(2)] void OnConnectionFailure([in] IArguments *pIArguments);
        [id(5)] void OnMonitorModeEstablished([in] IArguments *pIArguments);
        [id(39)] void OnConnectionClosed([in] IArguments *pIArguments);
        [id(41)] void OnControlFailureConf([in] IArguments *pIArguments);
        [id(304)] void OnCtiOsFailure([in] IArguments *pIArguments);
        [id(502)] void OnCallRTPStartedEvent([in] IArguments *pIArguments);
        [id(503)] void OnCallRTPStoppedEvent([in] IArguments *pIArguments);
        [id(802)] void OnSilentMonitorStatusReportEvent([in] IArguments *pIArguments);
        [id(803)] void OnStartSilentMonitorConf([in] IArguments *pIArguments);
        [id(804)] void OnStopSilentMonitorConf([in] IArguments *pIArguments);
        [id(805)] void OnSilentMonitorSessionDisconnected([in] IArguments *pIArguments);
Following is a brief description of each event. These events are described in detail in the Session Object and Silent Monitor Object sections of Event Interfaces and Events, on page 167

**Table 30: SilentMonitor Com Object Events**

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnConnection</td>
<td>Indicates that the connect method was successful in establishing a connection.</td>
</tr>
<tr>
<td>OnConnectionFailure</td>
<td>Indicates that an active connection has failed. Can also indicate a bad parameter in the Connect method.</td>
</tr>
<tr>
<td>OnMonitorModeEstablished</td>
<td>Signals a successful call to SetMsgFilter. The call to Setmsgfilter is hidden by the Standalone control.</td>
</tr>
<tr>
<td>OnConnectionClosed</td>
<td>Disconnect was called and the connection is now closed.</td>
</tr>
<tr>
<td>OnControlFailureConf</td>
<td>A ControlFailureConf was received and can be handled.</td>
</tr>
<tr>
<td>OnCtiOsFailure</td>
<td>A CtiosFailure event was received. This could be Silent Monitor specific error code.</td>
</tr>
<tr>
<td>OnCallRTPStartedEvent, OnCallRTPStoppedEvent</td>
<td>RTP events have been received signaling the start and stop of the RTP streams.</td>
</tr>
<tr>
<td>OnSilentMonitorStatusReport Event</td>
<td>Reports status from a monitored client to the monitoring application.</td>
</tr>
<tr>
<td>OnStartSilentMonitorConf, OnStopSilentMonitorConf</td>
<td>Acknowledge that CTI OS handled the StartMonitoring and StopMonitoring request, respectively.</td>
</tr>
<tr>
<td>OnSilentMonitorSessionDisconnected</td>
<td>Indicates that the Silent Monitor session has timed out on the monitoring side.</td>
</tr>
</tbody>
</table>

**Deployment**

The StandAlone Com Object is a COM dll that you must register on the client system via the Regsvr32 Silentmonitorctl.dll. You also require, ccsnmt.dll and the two standard CTI OS COM dlls (CTIOSClient.dll and Arguments.dll).

**Sample Usage in Visual Basic 6.0**

The following sample code assumes a VB 6.0 form with 4 buttons (Connect, Disconnect, StartMonitoring, and StopMonitoring). If the parameters are based on edit fields, the source code below is all that is needed to
silent monitor via CTIOs. It is important to note that this control does not require supervisor privileges or even any login. The only event handler shown below (OnSessionDisconnected) is the one for a timed out Silent Monitor session.

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

Private Sub btnConnect_Click()
    m_Args.clear
    m_Args.AddItem "CtiosA", "localhost"
    m_Args.AddItem "portA", "42028"
    Dim nRetVal As Long
    SilentMonitorCtl.Connect m_Args, nRetVal
    If nRetVal <> CIL_OK Then
        MsgBox "Connect returned error “ + Str(nRetVal)
    End If
End Sub

Private Sub btnDisconnect_Click()
    Dim nRetVal As Long
    SilentMonitorCtl.Disconnect
End Sub

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

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

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

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

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

The CIL event interfaces discussed in this section and the following sections apply only to the C++, COM, and VB interfaces. For more information about a discussion of Java CIL counterpart events and event handling in the Java CIL, see Events in Java CIL, on page 281. For more information about a discussion of .NET CIL event handling, see Events in .NET CIL, on page 282.

The Client Interface Library provides a publisher-subscriber model for notifying clients of events. Client applications using the CIL can subscribe to one or more of the CIL event interfaces. For more information and examples on how to subscribe and unsubscribe for events, see Building Your Custom CTI Application, on page 33

The published CIL event interfaces are organized around the different classes of CTI objects that the CIL provides. The event interfaces described in this chapter are:

• **ISessionEvents.** This interface publishes the events that relate to actions on the Session object.
• **ICallEvents.** This interface publishes the events that relate to actions on Call objects.
• **IAgentEvents.** This interface publishes the events that relate to actions on Agent objects.
• **ISkillGroupEvents.** This interface publishes the events that relate to actions on SkillGroup objects.
• **IButtonEnablementEvents.** This interface publishes the events that relate to changes in the enable-disable status of softphone buttons.
• **ISilentMonitorEvents.** This interface sends events to subscribers of the Silent Monitor interface.
• **IMonitoredAgentEventsInterface.** This interface fires Agent events to a supervisor for his team members.
• **IMonitoredCallEventsInterface.** This interface sends Call events to a supervisor for one of his agent team members.
• **LogEventsAdapter** (Java only). This class provides the default implementation for the message handlers in ILogEvents.
• **IGenericEvents.** This interface sends generic events to subscribers of the IGenericEvents interface.

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

**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 can use it to display connection status.

**Syntax**

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

**Parameters**

`args`  
Arguments array containing the following fields.

*Table 31: ISession Event Parameters*

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

OnConnectionClosed

The OnConnectionClosed message is generated when a connection is terminated by the client. This message has no fields. This event indicates successful completion of an action that the client (CIL or application using the CIL) initiated. By contrast, the OnConnectionFailure event is generated when the connection terminated for reasons other than that the client closed the connection.

OnConnectionFailure

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

**Syntax**

C++

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

COM

```csharp
void OnConnectionFailure (IArguments * args)
```

VB

```vbnet
session_OnConnectionFailure (ByVal args As CtiosCLIENTLib.IArguments)
```
Parameters

args

Arguments array containing the following fields.

Table 32: OnConnectionFailure Event Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventTime</td>
<td>INT</td>
<td>Integer value with time of day expressed in milliseconds.</td>
</tr>
<tr>
<td>FailedServer</td>
<td>STRING</td>
<td>Name or TCP/IP address of the server that failed to respond. See ReasonCode.</td>
</tr>
<tr>
<td>ReasonCode</td>
<td>SHORT</td>
<td>Reason code 0 : eProtocolMismatch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reason code 1 : eMissedHeartbeats</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reason code 2 : eTransportError</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reason code 3 : eConnectFail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reason code 4 : eOtherError</td>
</tr>
</tbody>
</table>

OnConnectionRejected

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

Syntax

C++: void OnConnectionRejected (Arguments& args)

COM: void OnConnectionRejected (IArguments * args)

VB: Session_OnConnectionRejected (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Not currently used, reserved for future use.

OnCTIOSFailure

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

Note

CTI OS CIL sends the disconnect request to CTI OS Server when the login attempt fails. Hence, CTI OS Server closes the client connection.
### Syntax

**C++**
```cpp
void OnCTIOSFailure (Arguments & args)
```

**COM**
```cpp
void OnCTIOSFailure (IArguments * args)
```

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

### Parameters

**arg**

Arguments array containing the following fields.

#### Table 33: OnCTIOSFailure Event Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FailureCode</td>
<td>INT</td>
<td>A value according to an enumerated value, as shown immediately following this table.</td>
</tr>
<tr>
<td>SystemEventID</td>
<td>INT</td>
<td>Present only if FailureCode equals ServerConnectionStatus. Contains a value according to an enumerated value, as shown immediately following this table.</td>
</tr>
<tr>
<td>SystemEventArg1</td>
<td>INT</td>
<td>Present only if SystemEventID equals SysPeripheralOnline or SysPeripheralOffline. Contains the peripheral ID of the affected peripheral.</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>STRING</td>
<td>An error message.</td>
</tr>
</tbody>
</table>

Following are the enumerated values for FailureCode:

```cpp
enum enumCTIOS_FailureCode
{
    eDriverOutOfService = 1,
    eServiceNotSupported = eDriverOutOfService + 1,
    eOperationNotSupported = eServiceNotSupported + 1,
    eInvalidPrivilege = eOperationNotSupported + 1,
    eUnknownRequestID = eInvalidPrivilege + 1,
    eUnknownEventID = eUnknownRequestID + 1,
    eUnknownObjectID = eUnknownEventID + 1,
    eInvalidObjectState = eUnknownObjectID + 1,
    eServerConnectionStatus = eInvalidObjectState + 1,
    eInconsistentAgentData = eServerConnectionStatus + 1,
    eAgentAlreadyLoggedIn = eInconsistentAgentData + 1,
    eForcedNotReadyForConfigError = eAgentAlreadyLoggedIn + 1,
    eMonitorModeConnectionDenied = eForcedNotReadyForConfigError + 1
};
```

Following are the enumerated values for SystemEventID:

```cpp
enum enumCTIOS_SystemEventID
{
    eSysCentralControllerOnline = 1,
    eSysCentralControllerOffline = 2,
};
```
OnCurrentAgentReset

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

Syntax

**C++**

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

**COM**

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

**VB**

```vb
session_OnCurrentAgentReset (ByVal args As CtiosCLIENTLib.IArguments)
```

Parameters

**args**

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Table 34: OnCurrentAgentReset Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keyword</strong></td>
</tr>
<tr>
<td>UniqueObjectID</td>
</tr>
</tbody>
</table>

OnCurrentCallChanged

The OnCurrentCallChanged message is generated when the current call changes to another call.
Syntax

C++
void OnCurrentCallChanged(Arguments & args)

COM
void OnCurrentCallChanged (IArguments * args)

VB
session_OnCurrentCallChanged (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args

Arguments array containing the following fields.

Table 35: OnCurrentCallChanged Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>Unique object ID (if any) of the new current call.</td>
</tr>
</tbody>
</table>

OnFailure Event

Not supported.

OnGlobalSettingsDownloadConf

You can configure the client after you are in the CTI OS Server and then download this configuration to each CTI OS client desktop. When an application executes the RequestDesktopSettings method call on the Session, an eGlobalSettingsDownloadRequest event is sent to the server.

In response, the server sends an OnGlobalSettingsDownloadConf event back to the calling application. The Arguments object passed as a parameter in this event contains the Desktop Settings configuration information. The Arguments object is an array that can contain up to seven elements, each of which has the value of a nested Arguments array in a hierarchy that closely matches that of the CTI OS server configuration in the Windows registry.

Each of these Arguments arrays is a packed version of the configuration contained in the CTI OS Server. For more information, see CTI OS System Manager Guide for Cisco Unified ICM.

This section describes the contents of the Arguments array returned in the OnGlobalSettingsDownloadConf event. Custom applications can add values at the lowest level under each key. Custom values added in this way are passed to the client in this event. This section also identifies which keys and values in the CTI OS registry are passed to the client in this event.

For more information about what is available and how to configure these items, see the following sections in the CTI OS System Manager Guide for Cisco Unified ICM:

- MainScreen
- Defining Connection Profiles
- Declaring ECC Variables
• Configuring the Call Appearance Grid
• Automatic Agent Statistics Grid Configuration
• Automatic Skill Group Statistics Grid Configuration

Syntax

C++
  void OnGlobalSettingsDownloadConf(Arguments & args)

COM
  void OnGlobalSettingsDownloadConf(IArguments * args)

VB
  session_OnGlobalSettingsDownloadConf(ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
  An Arguments array containing the Enterprise Desktop Settings configuration from a CTIOS server. For more information about the Enterprise Desktop Settings values listed below, see the CTIOS System Manager Guide for Cisco Unified ICM.

The following are the top level elements in the Enterprise Desktop Settings registry key. The CTIOS server passes configuration data for these elements to the client in the OnGlobalSettingsConf event:

• ECC (Expanded Call Context) variables
• Grid
• IPCCSilentMonitor
• Login
• ScreenPreferences
• SoundPreferences

Other keys or values that are added to the EnterpriseDesktopSettings/All Desktops key in the CTIOS server registry are passed to the client in the DesktopSettings Arguments array as empty Arguments arrays.

The following sections describe the contents of the args array:

• ECC – Arguments array that contains the Expanded Call Context (ECC) variables declared on the CTIOS server in the “ECC/Name” registry subtree (the following figure).

The CTIOS server does not send any registry information contained in the CTIOS registry keys representing the ECC scalar and array names. Thus the ECC Arguments arrays are empty in the OnGlobalSettingsDownloadConf event, regardless of the contents of those keys.
Each ECC scalar configured in the CTIOS server registry is represented as an empty Arguments array with keyword “user.<name>”, where <name> is the ECC name as configured on CTIOS server.

Each ECC array configured in the CTIOS server registry is represented as multiple empty Arguments arrays with keywords “user.<name>[0]” to “user.<name>[n-1]”, where <name> is the ECC name as configured on the CTIOS server and n is the size of the array as configured on the CTIOS server.

- **Grid** – Arguments array contains information from the CTIOS server registry's Grid subtree. The grid element contains an Arguments array of up to three Arguments arrays:
  - **AgentStatistics**
  - **CallAppearance**
  - **SkillGroupStatistics**

Each of these arrays contains the keyword “columns,” an Arguments array that contains multiple nested Arguments arrays with key=<column_number>, where <column_number> corresponds to the name of a key within the Columns/Number registry subtree.

The configuration information for any key or value added to the SkillGroupStatistics, AgentStatistics, or CallAppearance registry keys is not passed to the client in the OnGlobalSettingsDownloadConf event.

The value for each column number in the AgentStatistics and SkillGroupStatistics element is an Arguments array containing the following key-value pairs:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Header</td>
<td>string</td>
</tr>
<tr>
<td>Custom values</td>
<td>custom</td>
</tr>
</tbody>
</table>

1 Other registry values added to the <column_number> registry key are passed in the OnGlobalSettingsDownloadConf event. Subkeys added to the <column_number> registry key are not passed in this event.

The value for each column number in the CallAppearance element is an Arguments array containing the following key-value pairs:
Table 37: CallAppearance Column Number: Key Values

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Header</td>
<td>string</td>
</tr>
<tr>
<td>editable</td>
<td>boolean</td>
</tr>
<tr>
<td>maxchars</td>
<td>integer</td>
</tr>
<tr>
<td>Custom values</td>
<td>custom</td>
</tr>
</tbody>
</table>

2 Other registry values added to the <column_number> registry key are passed in the OnGlobalSettingsDownloadConf event. Subkeys added to the <column_number> registry key are not passed in this event.

You can add custom keys in the CTIOS Server registry's Grid subtree at the same level as the SkillGroupStatistics, AgentStatistics, and CallAppearance keys. The Grid Arguments array (see the following figure) within this event contain items corresponding to these custom keys. Any custom element that you add must follow the same hierarchy in the registry as that used by the existing top level elements.

The custom element hierarchy format is as follows:

Figure 47: Grid Arguments Array

- IPCCSilentMonitor – Arguments array that contains configuration information from the CTIOS server registry's IPCCSilentMonitor/ Name subtree.

The IPCCSilentMonitor Arguments array contains a nested Arguments array with key="settings." This array contains the following key-value pairs:

Table 38: IPCCSilentMonitor: Key Values

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaTerminationPort</td>
<td>integer</td>
</tr>
</tbody>
</table>
Configuration information for registry values added to the IPCCSilentMonitor/Settings registry key is passed to the client in the OnGlobalSettingsConf event. Configuration information for subkeys added to the Settings registry key is not passed in this event.

You can add custom keys to the CTI OS registry in the IPCCSilentMonitor subtree at the same level as the Settings key. The IPCCSilentMonitor Arguments array within this event contain items corresponding to these custom keys. Any custom element that you add must follow the same hierarchy in the registry as that used by the existing top level elements.

Two silent monitoring types are supported for Unified CCE:

- CTI OS based
- CCM based

You configure the silent Monitor type used by CTI OS using the CCMBasedSilentMonitor registry key. If CCMBasedSilentMonitor is present and set to true, CTI OS is using Call Manager's silent monitor implementation. When this is the case, supervisor applications must initiate silent monitor using the Agent.SuperviseCall() method. Agent applications do not need to do anything. If CCMBasedSilentMonitor is not present or set to 0, CTI OS implementation of silent monitor is in use. When this is the case, supervisor and agent applications must invoke silent monitor using the SilentMonitorManager object.

The format of the IPCCSilentMonitor Arguments array is shown in the following figure.

**Figure 48: IPCCSilentMonitor Arguments Array**

```
"IPCCSilentMonitor",<Arguments array>
  "Settings",<Arguments array>
    "HeartBeatInterval",<int>
    ...
    "CustomName",Custom Value
    "CustomName",Custom Value
  "CustomSettings",<Arguments array>
    "CustomName",Custom Value

"Login",<Arguments array>
```

- **Login** – Arguments array that contains the information from the CTI OS server registry's Login subtree. This array contains a nested Arguments array with key="ConnectionProfiles" and with an Arguments array value for each connection profile. The keyword of each array is the name for the Connection Profile listed in the CTI OS server's registry. The value is another Arguments array.

The following key-value pairs are contained in each connection profile Arguments array:
Table 39: Unified CCE Agent Statistics: Key Values

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtiOsA</td>
<td>string</td>
</tr>
<tr>
<td>CtiOsB</td>
<td>string</td>
</tr>
<tr>
<td>PortA</td>
<td>integer</td>
</tr>
<tr>
<td>PortB</td>
<td>integer</td>
</tr>
<tr>
<td>Heartbeat</td>
<td>integer</td>
</tr>
<tr>
<td>MaxHeartbeats</td>
<td>integer</td>
</tr>
<tr>
<td>AutoLogin</td>
<td>boolean</td>
</tr>
<tr>
<td>WarnIfAlreadyLoggedIn</td>
<td>boolean</td>
</tr>
<tr>
<td>ShowFieldBitMask</td>
<td>integer</td>
</tr>
<tr>
<td>RejectIfAlreadyLoggedIn</td>
<td>boolean</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>integer</td>
</tr>
<tr>
<td>IPCCSilentMonitorEnabled</td>
<td>boolean</td>
</tr>
<tr>
<td>TOS</td>
<td>boolean</td>
</tr>
<tr>
<td>SwitchCapabilityBitMask</td>
<td>integer</td>
</tr>
<tr>
<td>WarnIfSilentMonitored</td>
<td>boolean</td>
</tr>
<tr>
<td>RasCallMode&lt;sup&gt;3&lt;/sup&gt;</td>
<td>integer</td>
</tr>
</tbody>
</table>

<sup>3</sup> Applicable only to RAS enabled Unified CCE Profiles

Configuration information for keys or values that are added to the Login registry key in the CTI OS server's registry does not appear in the Login Arguments array.

The format of the Login Arguments array is shown in the following figure.
SilentMonitorService Subkey

The <profile_name>/SilentMonitorService subkey contains parameters that clients use to connect to one of a set of silent monitor services. It contains the following keys:

Table 40: SilentMonitorService Parameters

<table>
<thead>
<tr>
<th><strong>Keyword</strong></th>
<th><strong>Value</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ListenPort</td>
<td>integer</td>
<td>Port on which the silent monitor service is listening for incoming connections.</td>
</tr>
<tr>
<td>TOS</td>
<td>integer</td>
<td>QOS setting for the connection.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>integer</td>
<td>Amount of time in milliseconds between heartbeats.</td>
</tr>
<tr>
<td>HeartbeatRetries</td>
<td>integer</td>
<td>Number of missed heartbeats before the connection is abandoned.</td>
</tr>
</tbody>
</table>
A key that contains a list of silent monitor services to which the CIL tries to connect. The CIL randomly chooses one of the services in this list. This key contains two subkeys:

- 1 - index of the first silent monitor service
- N - index of the Nth silent monitor service

All subkeys contain the following keyword:

- SilentMonitorService - host name or IP address of the silent monitor service.

The following figure illustrates the hierarchy of the SilentMonitorService subkey.

**Figure 50: SilentMonitorService Subkey Hierarchy**

- **ScreenPreferences** – Arguments array that contains the information configured in the CTI OS server registry’s ScreenPreferences/Name subtree. The ScreenPreferences array contains an element MainScreen, which is an Arguments array that contains the following key-value pairs:

  - AgentStatisticsIntervalSec
  - BringToFrontOnCall
  - FlashOnCall

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentStatisticsIntervalSec</td>
<td>integer</td>
</tr>
<tr>
<td>BringToFrontOnCall</td>
<td>boolean</td>
</tr>
<tr>
<td>FlashOnCall</td>
<td>boolean</td>
</tr>
</tbody>
</table>
You can add custom keys to the CTI OS registry in the ScreenPreferences subtree at the same level as the “MainScreen” key. The ScreenPreferences Arguments array within this event contains items corresponding to these custom keys. Any custom key that you add must follow the same hierarchy in the registry as that used by the existing top level keys.

Registry values added to the MainScreen registry key on the CTI OS server are passed to the client in the OnGlobalSettingsDownloadConf event. Subkeys added to the MainScreen registry key are not passed in this event.

The format of the ScreenPreferences Arguments array is shown in the following figure.

**Figure 51: ScreenPreferences Arguments Array**

- **SoundPreferences** – Arguments array that contains information configured on the CTI OS server in the SoundPreferences/Name subtree. This array includes a nested Arguments array that includes a setting for each sound, including .wav files to be played, and whether or not each one is mute. It can also include custom name/value pairs for a custom application.

The SoundPreferences array contains the following key-value pairs:

**Table 42: SoundPreferences: Key Values**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value</th>
<th>Subtree</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTMF*</td>
<td>Arguments array</td>
<td>SoundPreferences/Name/DTMF</td>
</tr>
<tr>
<td>DialTone*</td>
<td>Arguments array</td>
<td>SoundPreferences/Name/DialTone</td>
</tr>
<tr>
<td>OriginatingTone*</td>
<td>Arguments array</td>
<td>SoundPreferences/Name/OriginatingTone</td>
</tr>
<tr>
<td>RingInTone*</td>
<td>Arguments array</td>
<td>SoundPreferences/Name/RingInTone</td>
</tr>
<tr>
<td>All*</td>
<td>Arguments array</td>
<td>SoundPreferences/Name/All</td>
</tr>
</tbody>
</table>

* Registry values added to this registry key in the CTI OS server registry are included in the Arguments array. Subkeys added to this registry key are not present.
The DTMF, DialTone, OriginatingTone, RingInTone, and All arrays each contain the keyword Mute, which has a boolean value. Custom registry values added to the DialTone DTMF, DialTone, OriginatingTone, RingInTone, and All registry keys are present in the array. Subkeys added to the these registry keys are not present in the array.

You can add custom keys in the SoundPreferences subtree at the same level as the All, DTMF, DialTone, OriginatingTone, and RingInTone keys. The SoundPreferences array contains items corresponding to these custom keys. Any custom element that you add must follow the same hierarchy in the registry as that used by the existing top level elements.

The format of the SoundPreferences Arguments array is shown in the following figure.

Figure 52: SoundPreferences Arguments Array

This configuration is stored in the Windows System Registry database and many of the values are set when the CTI OS Server Setup is run. You can set custom configuration at a later time by using the Windows Registry Editor.

OnHeartbeat

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

Syntax

C++
void OnHeartbeat(Arguments& args)

COM
void Onheartbeat (IArguments * args)

VB
session_OnHeartbeat (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.

Table 43:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventTime</td>
<td>INT</td>
<td>Integer value with time of day expressed in milliseconds.</td>
</tr>
</tbody>
</table>
OnMissingHeartbeat

The `OnMissingHeartbeat` event is generated when an expected heartbeat is not received. It returns the number of consecutive heartbeats missed and time of day. When the number of heartbeats missed equals or exceeds the maximum number of heartbeats allowed (set in the `MaxHeartbeats` property), an `OnConnectionFailure` event is generated instead of an `OnMissingHeartbeat` event, and the CIL automatically attempts to reconnect to the CTI OS server, alternating between the CtiosA and CtiosB servers passed as parameters in the Connect method.

**Syntax**

**C++**

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

**COM**

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

**VB**

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

**Parameters**

`args`

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventTime</td>
<td>INT</td>
<td>Integer value with time of day expressed in milliseconds.</td>
</tr>
<tr>
<td>Consecutive MissedHeartbeats</td>
<td>INT</td>
<td>Integer value with the number of heartbeats missed.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Integer value with the heartbeat interval, in milliseconds.</td>
</tr>
</tbody>
</table>

OnMonitorModeEstablished

The `OnMonitorModeEstablished` event is generated when Monitor Mode is established.

**Syntax**

**C++**

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

**COM**

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

**VB**

```vb
session_OnMonitorModeEstablished (ByVal args As CtiosCLIENTLib.IArguments)
```
### Parameters

**args**

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIL ConnectionID</td>
<td>STRING</td>
<td>ID of the client connection on the server.</td>
</tr>
<tr>
<td>StatusSystem</td>
<td>ARGUMENTS</td>
<td>Arguments array containing the following elements:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• StatusCTIServer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• StatusCtiServerDriver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• StatusCentralController</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• StatusPeripherals (Arguments array with a peripheral ID for each key and a boolean true/false value indicating if that peripheral is online.)</td>
</tr>
</tbody>
</table>

#### OnSnapshotDeviceConf

The OnSnapshotDeviceConf confirmation message is fired to the client as part of a snapshot operation. For AgentMode clients, the OnSnapshotDeviceConf arrives at startup time, after the OnQueryAgentStateConf message. The OnSnapshotDeviceConf indicates the number of calls present at the device, and their UniqueObjectIDs.

**Syntax**

**C++**

```cpp
void OnSnapshotDeviceConf (Arguments & args);
```

**COM**

```cpp
HRESULT OnSnapshotDeviceConf ([in] IArguments * args);
```

**VB**

```vb
OnSnapshotDeviceConf (ByVal args as CTIOSCLIENTLIB.IArguments)
```

### Parameters

**args**

Arguments array containing the following fields.
Table 46: OnSnapshotDeviceConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>Unique ID of the device object on the server. There are no device objects in the CIL, so this keyword cannot be used to retrieve a device object at this point.</td>
<td>STRING</td>
</tr>
<tr>
<td>NumCalls</td>
<td>The number of active calls associated with this device, up to a maximum of 16.</td>
<td>SHORT</td>
</tr>
<tr>
<td>ValidCalls</td>
<td>An Arguments array containing the list of calls on the device. The Unique ObjectID of each call is a key in the Arguments object. The value is a boolean indicating if the call is valid. Calls not listed are not valid calls on the device.</td>
<td>ARGUMENTS</td>
</tr>
</tbody>
</table>

Remarks
The CIL uses this event to rectify the list of calls on a device when logging in after a failover, in case the status of calls on the device changes during the failure period. An example of such a scenario is an agent talking on a call on a hardphone and a CTI failure occurs. The agent hangs up the call before CTI is recovered. After CTI and the CIL recover, they use the snapshot to discover that the call it currently has is no longer on the device. CTI then fires an EndCall event to remove the call from its call list.

OnSnapshotSkillGroupList
Not supported.

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

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

C++

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

COM

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

VB

```vb
session_OnTranslationRoute(ByVal args As CtiosCLIENTLib.IArguments)
```

Parameters

args

Arguments array containing the following fields.

**Table 47: OnTranslationRoute Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMEnterpriseUniqueID</td>
<td>STRING</td>
<td>This string is a globally unique key for this contact, which corresponds to the Unified ICM 64 bit key. You can use this parameter to match this contact to a follow-on call event.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the Unified ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>INT</td>
<td>The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>RouterCallKeySequenceNumber</td>
<td>INT</td>
<td>Together with RouterCallKeyDay and RouterCallKeyCallID fields forms the TaskID.</td>
</tr>
<tr>
<td>NumNamedVariables</td>
<td>SHORT</td>
<td>Number of Named variables.</td>
</tr>
<tr>
<td>NumNamedArrays</td>
<td>SHORT</td>
<td>Number of Named Arrays.</td>
</tr>
<tr>
<td>ANI</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>UserToUserInfo</td>
<td>STRING</td>
<td>The ISDN user-to-user information element.</td>
</tr>
<tr>
<td>DNIS</td>
<td>STRING</td>
<td>The DNIS number to which this call will arrive on the ACD/PBX.</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
</tbody>
</table>
**ICallEvents Interface**

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

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallerEnteredDigits</td>
<td>STRING</td>
<td>The digits entered by the caller in response to VRU prompting.</td>
</tr>
<tr>
<td>CallVariable1</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable10</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>ECC</td>
<td>ARGUMENTS</td>
<td>A nested Arguments structure of key-value pairs for all of the ECC data arriving with this call.</td>
</tr>
</tbody>
</table>

Many of the parameters that CTI OS receives from the CTI Server are inconsequential to most customer applications. The most important parameters for doing a screenpop are included with the events described in this section. The more inconsequential parameters are suppressed at the CTI OS Server, to minimize network traffic to the clients. However, you can enable the complete set of available event arguments by setting the following registry setting:

```plaintext
[HKLM\Cisco Systems\CTIOS\Server\CallObject\MinimizeEventArgs = 0].
```

### OnAgentPrecallEvent

The OnAgentPrecallEvent event is applicable to Unified CCE only. The equivalent on all other TDM events is TranslationRouteEvent.

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

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

C++
void OnAgentPrecallEvent(Arguments& args)

COM
void OnAgentPrecallEvent (IArguments * args)

VB
session_OnAgentPrecallEvent (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.

Table 48: OnAgentPrecallEvent Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMEnterpriseUniqueID</td>
<td>STRING</td>
<td>This string is a globally unique key for this contact, which corresponds to the Unified ICM 64 bit key. You can use this parameter to match this contact to a follow-on call event.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the Unified ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>RouterCalKeyCallID</td>
<td>INT</td>
<td>The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>The agent instrument that the call is routed to.</td>
</tr>
<tr>
<td>NumNamedVariables</td>
<td>SHORT</td>
<td>Number of Named variables.</td>
</tr>
<tr>
<td>NumNamedArrays</td>
<td>SHORT</td>
<td>Number of Named Arrays.</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The Unified ICM ServiceID of the service that the call is attributed to.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>An optional, user-defined number of the agent SkillGroup the call is attributed to, as known to the peripheral.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup the call is attributed to.</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>ANI</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>UserToUserInfo</td>
<td>STRING</td>
<td>The ISDN user-to-user information element.</td>
</tr>
<tr>
<td>DNIS</td>
<td>STRING</td>
<td>The DNIS number to which this call will arrive on the ACD/PBX.</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits</td>
<td>STRING</td>
<td>The digits entered by the caller in response to VRU prompting.</td>
</tr>
<tr>
<td>CallVariable1</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>...</td>
<td>STRING</td>
<td>...</td>
</tr>
<tr>
<td>CallVariable10</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>ECC</td>
<td>ARGUMENTS</td>
<td>A nested Arguments structure of key-value pairs for all of the ECC data arriving with this call.</td>
</tr>
<tr>
<td>CallTypeIDTag</td>
<td>INT</td>
<td>Specifies CallType of the call and indicates that the agent is reserved via LegacyPreCall.</td>
</tr>
<tr>
<td>PreCallInvokeIDTag</td>
<td>INT</td>
<td>Specifies the invoking of the LegacyPreCall.</td>
</tr>
</tbody>
</table>

**OnAgentPrecallAbortEvent**

**Note**

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

**Syntax**

**C++**

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

**COM**

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

**VB**

```vb
session_OnAgentPrecallAbortEvent (ByVal args As CtiosCLIENTLib.IArguments)
```

**Parameters**

`args`

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMEnterpriseUniqueID</td>
<td>STRING</td>
<td>This string is a globally unique key for this contact, which corresponds to the Unified ICM 64 bit key. You can use this parameter to match this contact to a follow-on call event.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the Unified ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>INT</td>
<td>The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>The agent instrument that the call will be routed to.</td>
</tr>
</tbody>
</table>

**OnAlternateCallConf**

The OnAlternateCallConf event is fired to the client to indicate that an Alternate request was received by the CTI Server.
Syntax

**C++**

void OnAlternateCallConf (Arguments & args);

**COM**

HRESULT OnAlternateCallConf ([in] IArguments * args);

**VB**

Session_ OnAlternateCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

**Parameters**

args

Arguments array containing the following field.

**Table 50: OnAlternateCallConf Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
</tbody>
</table>

**OnAnswerCallConf**

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

Syntax

**C++**

void OnAnswerCallConf (Arguments & args);

**COM**

HRESULT OnAnswerCallConf ([in] IArguments * args);

**VB**

Session_ OnAnswerCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

**Parameters**

args

Arguments array containing the following field.

**Table 51: OnAnswerCallConf Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
</tbody>
</table>
OnCallBegin

The OnCallBegin event is generated at the first association between a call and the CTI Client. The event passes the call identifier and the initial call context data. The ConnectionCallID identifies the call. This message always precedes any other event messages for that call.

Subsequent changes to the call context data (if any) are signaled by an OnCallDataUpdate event containing the changed call data.

---

**Note**

There can be multiple calls with the same ConnectionCallID value.

---

**Syntax**

**C++**

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

**COM**

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

**VB**

```cpp
session_OnCallBegin (ByVal args As CtiosCLIENTLib.IArguments)
```

**Parameters**

`args`

Arguments array containing the following fields.

**Table 52: OnCallBegin Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call’s records in the Unified ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>INT</td>
<td>The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>RouterCallKeySequenceNumber</td>
<td>INT</td>
<td>Together with RouterCallKeyDay and RouterCallKeyCallID fields forms the TaskID.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>UINT</td>
<td>The Call ID value assigned to this call by the peripheral or the Unified ICM.</td>
</tr>
<tr>
<td>ANI (optional)</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>DNIS (optional)</td>
<td>STRING</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>UserToUserInfo (Optional)</td>
<td>STRING</td>
<td>The ISDN user-to-user information element, unspecified, up to 131 bytes.</td>
</tr>
<tr>
<td>DialedNumber (Optional)</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits (Optional)</td>
<td>STRING</td>
<td>The digits entered by the caller in response to VRU prompting.</td>
</tr>
<tr>
<td>ServiceNumber (Optional)</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID (Optional)</td>
<td>INT</td>
<td>The Unified ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>An optional, user-defined number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID (Optional)</td>
<td>INT</td>
<td>The system-generated identifier of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
</tbody>
</table>
## OnCallCleared

An OnCallCleared event is generated when the voice portion of all parties on a call is terminated, normally when the last device disconnects from a call. With this event the connection status becomes LCS_NULL.

### Syntax

**C++**

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

**COM**

```cpp
generic
void OnCallCleared (IArguments * args)
```
VB

session_OnCallCleared (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.

Table 53: OnCallCleared Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ICMEnterprise UniqueID</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
<tr>
<td>(Optional)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OnCallConnectionCleared

An OnCallConnectionCleared event is generated when a party drops from a call. With this event the connection status becomes LCS_NULL.

Note

If the CallConnectionCleared event is received after having received a CallFailed event, the event does not include a CallStatus because it is important to preserve the fact that the call failed (maintaining the CallStatus of LSC_Fail). Because of this exception, the CallStatus of the CallConnectionCleared event is optional.

Syntax

C++

void OnCallConnectionCleared(Arguments& args)

COM

void OnCallConnectionCleared (IArguments * args)

VB

session_OnCallConnectionCleared (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.
### Table 54: OnCallConnectionCleared Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ICMEnterprise UniqueID</td>
<td>(Optional)STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>

### OnCallConferenced

The joining of calls into a conference call or the adding of a new call joining a conference can generate an OnCallConferenced event. With this event, the connections at the controller's device merge to become one connection with a status of LCS_CONNECT, and the status of the connections at the original caller's device and at the consulted device remain unchanged.

#### Syntax

**C++**

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

**COM**

```com
void OnCallConferenced (IArguments * args)
```

**VB**

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

#### Parameters

**args**

Arguments array containing the following fields.

#### Table 55: OnCallConferenced Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the Unified ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>INT</td>
<td>The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>UINT</td>
<td>The Call ID value assigned to this call by the peripheral or the Unified ICM.</td>
</tr>
<tr>
<td>ANI (optional)</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>DNIS (optional)</td>
<td>STRING</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>UserToUserInfo (Optional)</td>
<td>STRING</td>
<td>The ISDN user-to-user information element. unspecified, up to 131 bytes.</td>
</tr>
<tr>
<td>DialedNumber (Optional)</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits (Optional)</td>
<td>STRING</td>
<td>The digits entered by the caller in response to VRU prompting.</td>
</tr>
<tr>
<td>ServiceNumber (Optional)</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID (Optional)</td>
<td>INT</td>
<td>The Unified ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>An optional, user-defined number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID (Optional)</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority (Optional)</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>CallWrapupData (Optional)</td>
<td>STRING</td>
<td>Call-related wrap-up data.</td>
</tr>
<tr>
<td>CallVariable1 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>...</td>
<td>STRING</td>
<td>...</td>
</tr>
<tr>
<td>CallVariable10 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallStatus (optional)</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ECC (optional)</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains all of the Expanded Call Context variables in use; for example: user.ArrayVariable[0]user.ArrayVariable[1]...user.ArrayVariable[n]user.ScalarVariable</td>
</tr>
<tr>
<td>CTIClients (Optional)</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains the information about the number of clients that are using the Call object; for example: CTIClient[1]CTIClientSignatureCTIClientTimestamp</td>
</tr>
<tr>
<td>ICMEnterpriseUnique ID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>
OnCallDataUpdate

Changes to the call context data generate an OnCallDataUpdate event. Only the changed items are in the event argument array. The initial call context is provided in the OnCallBegin event.

Syntax

**C++**

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

**COM**

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

**VB**

```cpp
session_OnCallDataUpdate (ByVal args As CtiosCLIENTLib.IArguments)
```

Parameters

**args**

Arguments array containing the following fields.

*Table 56: OnCallUpdate Parameters*

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the Unified ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>INT</td>
<td>The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>RouterCallKeySequenceNumber</td>
<td>INT</td>
<td>Together with RouterCallKeyDay and RouterCallKeyCallID fields forms the TaskID.</td>
</tr>
<tr>
<td><strong>Keyword</strong></td>
<td><strong>Type</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>UINT</td>
<td>The Call ID value assigned to this call by the peripheral or the Unified ICM.</td>
</tr>
<tr>
<td>ANI (optional)</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>DNIS (optional)</td>
<td>STRING</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>UserToUserInfo (Optional)</td>
<td>STRING</td>
<td>The ISDN user-to-user information element. unspecified, up to 131 bytes.</td>
</tr>
<tr>
<td>DialedNumber (Optional)</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits (Optional)</td>
<td>STRING</td>
<td>The digits entered by the caller in response to VRU prompting.</td>
</tr>
<tr>
<td>ServiceNumber (Optional)</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID (Optional)</td>
<td>INT</td>
<td>The Unified ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>An optional, user-defined number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID (Optional)</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority (Optional)</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>CallWrapupData (Optional)</td>
<td>STRING</td>
<td>Call-related wrap-up data.</td>
</tr>
<tr>
<td>CallVariable1 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
</tbody>
</table>
### OnCallDelivered

The OnCallDelivered event may be generated when the call arrives at the agent's teleset. Both parties (call connections) receive this event. With this event, the called party's connection status becomes LCS_ALERTING but the calling party's connection status remains LCS_INITIATE.

With certain switches, when a call is made outside of the ACD, this event may not be received. For more information, see OnCallReachedNetwork.

**Syntax**

**C++**

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

**COM**

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

**VB**

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

**Parameters**

`args`

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>STRING</td>
<td>...</td>
</tr>
<tr>
<td>CallVariable10 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallStatus (optional)</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ECC (optional)</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains all of the Expanded Call Context variables in use; for example: user.ArrayVariable[0], user.ArrayVariable[1], user.ArrayVariable[n], user.ScalarVariable</td>
</tr>
<tr>
<td>CTIClients (Optional)</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains the information about the number of clients that are using the Call object; for example: CTIClient[1], CTIClientSignature, CTIClientTimestamp</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>
### Table 57: OnCallDelivered Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The Unified ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>An optional, user-defined number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID (Optional)</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority (Optional)</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call. See Table 6: CIL Error Codes, on page 21.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
<tr>
<td>TrunkNumber (optional)</td>
<td>INT</td>
<td>The number representing a trunk.</td>
</tr>
</tbody>
</table>
OnCallDequeuedEvent

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

Syntax

C++
void OnCallDequeuedEvent(Arguments& args)

COM
void OnCallDequeuedEvent (IArguments * args)

VB
session_OnCallDequeuedEvent (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.

Table 58: OnCallDequeuedEvent Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrunkGroup Number</td>
<td>INT</td>
<td>The number representing a trunk group.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>INT</td>
<td>The identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of the connection identifier supplied in the ConnectionDeviceID.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>SHORT</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>SHORT</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>SHORT</td>
<td>Identifies the teleset line being used.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The Unified ICM ServiceID of the service that the call is attributed to.</td>
</tr>
</tbody>
</table>
### OnCallDiverted

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

**Syntax**

**C++**

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

**COM**

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

**VB**

```vb
session_OnCallDiverted (ByVal args As CtiosCLIENTLib.IArguments)
```

**Parameters**

args

Arguments array containing the following fields.

**Table 59: OnCallDiverted Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>Unique reference generated for a call at client.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>ConnectionDevice IDType</td>
<td>SHORT</td>
<td>Indicates the type of ConnectionDeviceID value.</td>
</tr>
<tr>
<td>Connection DeviceID</td>
<td>INT</td>
<td>The device identifier of the connection between the call and the device.</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.

#### Syntax

**C++**

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

**COM**

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

**VB**

```cpp
session_OnCallEnd (ByVal args As CtiosCLIENTLib.IArguments)
```

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionCallID</td>
<td>UINT</td>
<td>The Call ID value assigned to this call by the peripheral or the Unified ICM.</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The Unified ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>DivertingDeviceType</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the DivertingDeviceID field.</td>
</tr>
<tr>
<td>CalledDeviceType</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID field.</td>
</tr>
<tr>
<td>LocalConnectionState</td>
<td>SHORT</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>SHORT</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>DivertingDeviceID (Optional)</td>
<td>STRING</td>
<td>The device identifier of the device from which the call was diverted.</td>
</tr>
<tr>
<td>CalledDeviceID (Optional)</td>
<td>STRING</td>
<td>The device identifier of the device to which the call was diverted.</td>
</tr>
</tbody>
</table>
OnCallEstablished

The OnCallEstablished event is generated when the call is answered at the agent's teleset. Both parties (call connections) receive this event when the call is answered. With this event, the call status of both parties becomes LCS_CONNECT.

Note

With certain switches, when a call is made outside of the ACD, this event may not be received. See OnCallReachedNetwork for more detail.

Syntax

C++

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

COM

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

VB

```vb
session_OnCallEstablished (ByVal args As CtiosCLIENTLib.IArguments)
```

Parameters

args

Arguments array containing the following fields.

Table 60: OnCallEnd Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>CallStatus (optional)</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ICMEnterprise UniqueID (optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The Unified ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>An optional, user-defined number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID (Optional)</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority (Optional)</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call. See Table 6: CIL Error Codes, on page 21.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
<tr>
<td>TrunkNumber (optional)</td>
<td>INT</td>
<td>The number representing a trunk.</td>
</tr>
</tbody>
</table>
OnCallFailed

The `OnCallFailed` event is generated when a call is not completed. With this event the connection status becomes LCS_FAIL. This usually happens as a result of a MakeCall or a MakeConsultCall request, but can occur at any other point in the call's lifetime if the call fails on an ACD. In this case, you should perform any required cleanup prior to arrival of an EndCall event.

**Note**

The events (CallConnectionCleared and CallCleared) received after the CallFailed event does not include a CallStatus because, until the call has ended, it is important to preserve the fact that this is a failed call.

**Syntax**

**C++**

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

**COM**

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

**VB**

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

**Parameters**

*args*

Arguments array containing the following fields.

**Table 62: OnCallFailed Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrunkGroup Number (optional)</td>
<td>INT</td>
<td>The number representing a trunk group.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
</tbody>
</table>
OnCallHeld

Placing a call on hold at the agent's teleset can generate an OnCallHeld event. With this event the connection status becomes LCS_HELD.

Syntax

C++

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

COM

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

VB

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

Parameters

args

Arguments array containing the following fields.

Table 63: OnCallHeld Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID (Optional)</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
</tbody>
</table>

OnCallOriginated

The initiation of a call from the peripheral can generate an OnCallOriginated event. Only the connection making the call receives this event. With this event the connection status becomes LCS_INITIATE.

Syntax

C++

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

COM

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

VB

```vbnet
session_OnCallOriginated (ByVal args As CtiosCLIENTLib.IArguments)
```
Parameters

args

Arguments array containing the following fields.

Table 64: OnCallOriginated Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The Unified ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>The user-defined number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID (Optional)</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority (Optional)</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
</tbody>
</table>
OnCallQueuedEvent

The placing of a call in a queue pending the availability of some resource can generate an OnCallQueuedEvent message to the CTI Client. Clients with Client Events Service can receive this message when an outbound call is queued waiting for a trunk or other resource. Clients with All Events Service can also receive this message when inbound calls are queued.

Syntax

C++

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

COM

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

VB

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

Parameters

`args`

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection Device ID</td>
<td>INT</td>
<td>The identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionDevice ID Type</td>
<td>SHORT</td>
<td>Indicates the type of the connection identifier supplied in the ConnectionDeviceID.</td>
</tr>
<tr>
<td>Queued Device ID</td>
<td>STRING</td>
<td>The device identifier of the queuing device.</td>
</tr>
<tr>
<td>QueuedDevice ID Type</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the QueuedDeviceID.</td>
</tr>
<tr>
<td>Calling Device ID</td>
<td>STRING</td>
<td>The device identifier of the calling device.</td>
</tr>
<tr>
<td>CallingDevice ID Type</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID.</td>
</tr>
<tr>
<td>Called Device ID</td>
<td>STRING</td>
<td>The device identifier of the called device.</td>
</tr>
<tr>
<td>CalledDevice ID Type</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID.</td>
</tr>
</tbody>
</table>
### OnCallReachedNetworkEvent

The connection of an outbound call to another network can generate an `OnCallReachedNetworkEvent`. With some switches outside the ACD, this can be the last event the outbound connection receives. For these switches, you cannot assume that when the called party receives and answers the call that the `OnCallDelivered` and `OnCallEstablished` events is received.

**Syntax**

**C++**

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

**COM**

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

**VB**

```vbnet
session_OnCallReachedNetworkEvent (ByVal args As CtiosCLIENTLib.IArguments)
```
Parameters

args

Arguments array containing the following fields.

Table 66: OnCallReachedNetworkEvent Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection DeviceID</td>
<td>STRING</td>
<td>The identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ConnectionDevice IDType</td>
<td>SHORT</td>
<td>Indicates the type of the connection identifier supplied in the ConnectionDeviceID.</td>
</tr>
<tr>
<td>TrunkUsedDeviceID</td>
<td>STRING</td>
<td>The device identifier of the selected trunk.</td>
</tr>
<tr>
<td>TrunkUsedDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the TrunkUsedDeviceID.</td>
</tr>
<tr>
<td>CalledDeviceID</td>
<td>STRING</td>
<td>The device identifier of the called device.</td>
</tr>
<tr>
<td>CalledDeviceIDType</td>
<td>SHORT</td>
<td>Indicates the type of the device identifier supplied in the CalledDeviceID.</td>
</tr>
<tr>
<td>LocalConnection State</td>
<td>SHORT</td>
<td>The state of the local end of the connection.</td>
</tr>
<tr>
<td>EventCause</td>
<td>SHORT</td>
<td>Indicates a reason or explanation for the occurrence of the event.</td>
</tr>
<tr>
<td>LineHandle</td>
<td>SHORT</td>
<td>Identifies the teleset line being used.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>TrunkNumber (optional)</td>
<td>INT</td>
<td>The number representing a trunk.</td>
</tr>
<tr>
<td>TrunkGroup Number (optional)</td>
<td>INT</td>
<td>The number representing a trunk group.</td>
</tr>
</tbody>
</table>

OnCallRetrieved

Resuming a call previously placed on hold at the agent’s teleset can generate an OnCallRetrieved event. With this event the connection status becomes LCS_CONNECT.
**Syntax**

C++

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

COM

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

VB

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

**Parameters**

args

Arguments array containing the following fields.

**Table 67: OnCallRetrieved Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
</tbody>
</table>

**OnCallServiceInitiatedEvent**

The initiation of telecommunications service ("dial tone") at the agent's teleset can generate an OnCallServiceInitiatedEvent to the CTI Client. However, when the call is made through the software, there is no way to detect the equivalent of the phone off hook. Therefore, after a call is made this event is received in sequence along with the OnCallOriginated and OnCallDelivered events. With this event the connection status becomes LCS_INITIATE.

**Syntax**

C++

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

COM

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

VB

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

**Parameters**

args

Arguments array containing the following fields.
### Table 68: OnCallServiceInitiatedEvent Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The Unified ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>The optional, user-defined number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID (Optional)</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority (Optional)</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
</tbody>
</table>

#### OnCallStartRecordingConf

The OnCallStartRecordingConf event is fired to the client to indicate that the CTI server received a StartRecord request.
Syntax

C++

    void OnCallStartRecordingConf (Arguments & args);

COM

    HRESULT OnCallStartRecordingConf ([in] IArguments * args);

VB

    Session_ OnCallStartRecordingConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following field.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
</tbody>
</table>

OnCallStopRecordingConf

The OnCallStopRecordingConf event is fired to the client to indicate that the CTI server received a StopRecord request.

Syntax

C++

    void OnCallStopRecordingConf (Arguments & args);

COM

    HRESULT OnCallStopRecordingConf ([in] IArguments * args);

VB

    Session_ OnCallStopRecordingConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following field.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
</tbody>
</table>
OnCallTransferred

The transfer of a call to another destination can generate an OnCallTransferred event. With this event the two connections at the controller’s device end and the status of the connections at the original caller's device and the consulted device are unchanged.

Syntax

C++

```cpp
doctoratOnCallTransferred(Arguments & args)
```  

COM

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

VB

```cpp
session_OnCallTransferred (ByVal args As CtiosCLIENTLib.IArguments)
```  

Parameters

args

Arguments array containing the following fields.

Table 71: OnCallTransferred Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the Unified ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>RouterCalKeyCallID</td>
<td>INT</td>
<td>The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>UINT</td>
<td>The Call ID value assigned to this call by the peripheral or the Unified ICM.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ANI (optional)</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>DNIS (optional)</td>
<td>STRING</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>UserToUserInfo (Optional)</td>
<td>STRING</td>
<td>The ISDN user-to-user information element. unspecified, up to 131 bytes.</td>
</tr>
<tr>
<td>DialedNumber (Optional)</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits (Optional)</td>
<td>STRING</td>
<td>The digits entered by the caller in response to VRU prompting.</td>
</tr>
<tr>
<td>ServiceNumber (Optional)</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral.</td>
</tr>
<tr>
<td>ServiceID (Optional)</td>
<td>INT</td>
<td>The Unified ICM ServiceID of the service that the call is attributed to.</td>
</tr>
<tr>
<td>SkillGroupNumber (Optional)</td>
<td>INT</td>
<td>The optional, user-defined number of the agent SkillGroup the call is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attributed to, as known to the peripheral.</td>
</tr>
<tr>
<td>SkillGroupID (Optional)</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup the call is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attributed to.</td>
</tr>
<tr>
<td>SkillGroupPriority (Optional)</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>applicable or not available.</td>
</tr>
<tr>
<td>CallWrapupData (Optional)</td>
<td>STRING</td>
<td>Call-related wrap-up data.</td>
</tr>
<tr>
<td>CallVariable1 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td></td>
<td>STRING</td>
<td>...</td>
</tr>
<tr>
<td>CallVariable10 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
</tbody>
</table>
### OnClearCallConf

The OnClearCallConf event is fired to the client to indicate that the CTI server received a Clear request.

#### Syntax

**C++**
```cpp
void OnClearCallConf (Arguments & args);
```

**COM**
```csharp
HRESULT OnClearCallConf ([in] IArguments * args);
```

**VB**
```vbnet
OnClearCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)
```

#### Parameters

**args**
Arguments array containing the following field.

#### Table 72: OnClearCallConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
</tbody>
</table>
OnClearConnectionConf

The OnClearConnectionConf event is fired to the client to indicate that the CTI server received a ClearConnection request.

Syntax

C++
void OnClearConnectionConf (Arguments & args);

COM
HRESULT OnClearConnectionConf ([in] IArguments * args);

VB
Session_OnClearConnectionConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args
Arguments array containing the following field.

Table 73: OnClearConnectionConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
</tbody>
</table>

OnConferenceCallConf

The OnConferenceCallConf event is fired to the client to indicate that the CTI server received a ConferenceCall or SingleStepConferenceCall request.

Syntax

C++
void OnConferenceCallConf (Arguments & args);

COM
HRESULT OnConferenceCallConf ([in] IArguments * args);

VB
Session_OnConferenceCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args
Arguments array containing the following field.
### OnConsultationCallConf

The `OnConsultationCallConf` event is fired to the client to indicate that the CTI server received a `MakeConsultCall` request.

**Syntax**

**C++**

```cpp
void OnConsultationCallConf (Arguments & args);
```

**COM**

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

**VB**

```vbnet
OnConsultationCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)
```

**Parameters**

**args**

Arguments array containing the following field.

**Table 75: OnConsultationCallConf Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
</tbody>
</table>

### OnControlFailureConf

The `OnControlFailureConf` event is generated when a request to the peripheral (the ACD) fails.

**Syntax**

**C++**

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

**COM**

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

**VB**

```vbnet
session_OnControlFailureConf (ByVal args As CtiosCLIENTLib.IArguments)
```
Parameters

args

Arguments array containing the following fields.

Table 76: OnControlFailureConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>Peripheral ID.</td>
</tr>
<tr>
<td>FailureCode</td>
<td>SHORT</td>
<td>Code ID.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID that represents a specific client.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>MessageType</td>
<td>INT</td>
<td>Contains the CTI OS Command Request ID that failed to execute. The message types included in this parameter are those to used to control Call, Agent State and Supervisor actions. For more information, see CTI OS Keywords and Enumerated Types, on page 499.</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>STRING</td>
<td>String text containing the description of the failure.</td>
</tr>
</tbody>
</table>

OnHoldCallConf

The OnHoldCallConf event is fired to the client to indicate that the CTI server received a Hold request.

Syntax

**C++**

```c++
void OnHoldCallConf (Arguments & args);
```

**COM**

```c
HRESULT OnHoldCallConf ([in] IArguments * args);
```

**VB**

```vb
Session_OnHoldCallConf (ByVal args as IArguments)
```
Parameters

args

Arguments array containing the following field.

Table 77: OnHoldCallConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
</tbody>
</table>

OnMakePredictiveCallConf

Not supported.

OnReconnectCallConf

The OnReconnectCallConf event is fired to the client to indicate that the CTI server received a Reconnect request.

Syntax

C++

    void OnReconnectCallConf (Arguments & args);

COM

    HRESULT OnReconnectCallConf ([in] IArguments * args);

VB

    OnReconnectCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following field.

Table 78: OnMakePredictiveCallConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
</tbody>
</table>

OnReleaseCallConf

Not supported.
OnRetrieveCallConf

The OnRetrieveCallConf event is fired to the client to indicate that the CTI server received a RetrieveCall request.

Syntax

C++
void OnRetrieveCallConf (Arguments & args);

COM
HRESULT OnRetrieveCallConf ([in] IArguments * args);

VB
Session_ OnRetrieveCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following field.

Table 79: OnReleaseCallConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
</tbody>
</table>

OnSendDTMFConf

The OnSendDTMFConf event is fired to the client to indicate that the CTI server received a SendDTMF request.

Syntax

C++
void OnSendDTMFConf (Arguments & args);

COM
HRESULT OnSendDTMFConf ([in] IArguments * args);

VB
Session_ OnSendDTMFConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Not used; reserved for future use.
**OnSetCallDataConf**

The OnSetCallDataConf event is fired to the client to indicate that the CTI server received a SetCallData request.

**Syntax**

C++

```cpp
void OnSetCallConf (Arguments & args);
```

COM

```cpp
HRESULT OnClearCallConf ([in] IArguments * args);
```

VB

```vb
OnClearCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)
```

**Parameters**

*args*

Arguments array containing the following fields.

<p>| <strong>Table 80: OnReleaseCallConf Parameters</strong> |</p>
<table>
<thead>
<tr>
<th><strong>Keyword</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
</tbody>
</table>

**OnSnapshotCallConf**

The OnSnapshotCallConf event is generated when a SnapshotCall request for a specific call is successful. It contains all the information known about the specific connection at that point in time.

**Syntax**

C++

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

COM

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

VB

```vb
session_OnSnapshotCallConf (ByVal args As CtiosCLIENTLib.IArguments)
```

**Parameters**

*args*

Arguments array containing the following fields.
### Table 81: OnSnapShotCallConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the call object.</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>CallerEnteredDigits</td>
<td>STRING</td>
<td>The digits entered by the caller in response to VRU prompting.</td>
</tr>
<tr>
<td>CallWrapupData</td>
<td>STRING</td>
<td>Call-related wrap-up data.</td>
</tr>
<tr>
<td>CallVariable1 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>...</td>
<td>STRING</td>
<td>...</td>
</tr>
<tr>
<td>CallVariable10 (Optional)</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CustomerPhoneNumber</td>
<td>STRING</td>
<td>The customer phone number associated with the call.</td>
</tr>
<tr>
<td>CustomerAccountNumber</td>
<td>STRING</td>
<td>The customer account number associated with the call.</td>
</tr>
<tr>
<td>ECC</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains all of the Expanded Call Context variables in use; for example: user.ArrayVariable[0].user.ArrayVariable[1].user.ArrayVariable[n].user.ScalarVariable</td>
</tr>
<tr>
<td>CTIClients (Optional)</td>
<td>ARGUMENTS</td>
<td>Arguments array that contains the information about the number of clients that are using the Call object; for example: CTIClient[1].CTIClientSignature.CTIClientTimestamp</td>
</tr>
<tr>
<td>RouterCallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the Unified ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
</tbody>
</table>
### Table 82: SnapshotCallConfl Additional Fields

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
The identifier of the call connection. This field always immediately follows the corresponding CallConnectionDeviceID field.

The active state of the call. This field always immediately follows the corresponding CallConnectionDeviceID field.

Indicates the type of the device identifier supplied in the CallDeviceID field.

### OnTransferCallConf

The OnTransferCallConf event is fired to the client to indicate that the CTI server received a TransferCall or SingleStepTransferCall request.

#### Syntax

**C++**

```cpp
void OnTransferCallConf (Arguments & args);
```

**COM**

```csharp
HRESULT OnTransferCallConf ([in] IArguments * args);
```

**VB**

```vbnet
Session_ OnTransferCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)
```

#### Parameters

**args**

Arguments array containing the following field.

#### Table 83: OnTransferCallConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
</tbody>
</table>

## IAgentEvents Interface

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

The OnAgentDeskSettingsConf event confirms successful completion of the request and provides the query response.

Syntax

C++
void OnAgentDeskSettings(Arguments& args)

COM
void OnAgentDeskSettings (IArguments * args)

VB
session_OnAgentDeskSettings (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvokeID</td>
<td>UINT</td>
<td>Set to the same value as the InvokeID from the corresponding request message.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>UINT</td>
<td>The Unified ICM PeripheralID of the ACD where the device is located.</td>
</tr>
<tr>
<td>DeskSettingsMask</td>
<td>UINT</td>
<td>A bitwise combination of the Boolean desk setting Masks listed in the table below.</td>
</tr>
<tr>
<td>WrapupData IncomingMode</td>
<td>UINT</td>
<td>Indicates whether the agent is allowed or required to enter wrap-up data after an inbound call: 0 = Required, 1 = Optional, 2 = Not allowed, 3 = Required With WrapupData.</td>
</tr>
<tr>
<td>WrapupData OutgoingMode</td>
<td>UINT</td>
<td>Indicates whether the agent is allowed or required to enter wrap-up data after an outbound call: 0 = Required, 1 = Optional, 2 = Not allowed.</td>
</tr>
<tr>
<td>LogoutNonActivityTime</td>
<td>UINT</td>
<td>Number of seconds of non-activity at the desktop after which the Unified ICM automatically logs out the agent.</td>
</tr>
<tr>
<td>QualityRecordingRate</td>
<td>UINT</td>
<td>Indicates how frequently calls to the agent are recorded.</td>
</tr>
<tr>
<td>RingNoAnswerTime</td>
<td>UINT</td>
<td>Number of seconds a call can ring at the agent's station before being redirected.</td>
</tr>
</tbody>
</table>
Set for a warning message box to prompt on agent desktop when silent monitor starts.

Set for an audio click at beginning of the silent monitor.

Set for PIM to create a blind conference call for supervisor assist request; otherwise creates consultative call.

Set for PIM to create a blind conference call for emergency call request; otherwise creates a consultative call.

Set for automatically record when emergency call request.

Set for the recording request to go through Call Manager/PIM.

Auto Wrap-up time out.

The dialed number identifier for new re-route destination in the case of ring no answer.

<table>
<thead>
<tr>
<th>Mask Name</th>
<th>Description</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESK_AVAIL_AFTER_INCOMING_MASK</td>
<td>Set for automatically consider the agent available after handling an incoming call.</td>
<td>0x00000001</td>
</tr>
<tr>
<td>DESK_AVAIL_AFTER_OUTGOING_MASK</td>
<td>Set for automatically consider the agent available after handling an outbound call.</td>
<td>0x00000002</td>
</tr>
<tr>
<td>DESK_AUTO_ANSWER_ENABLED_MASK</td>
<td>Set when calls to the agent are automatically answered.</td>
<td>0x00000004</td>
</tr>
<tr>
<td>DESK_IDLE_REASON_REQUIRED_MASK</td>
<td>Set when the agent must enter a reason before entering the Idle state.</td>
<td>0x00000008</td>
</tr>
<tr>
<td>DESK_LOGOUT_REASON_REQUIRED_MASK</td>
<td>Set when the agent must enter a reason before logging out.</td>
<td>0x00000010</td>
</tr>
<tr>
<td>DESK_SUPERVISOR_CALLS_ALLOWED_MASK</td>
<td>Set when the agent can initiate supervisor assisted calls.</td>
<td>0x00000020</td>
</tr>
<tr>
<td>DESK_AGENT_TO_AGENT_CALLS_ALLOWED</td>
<td>Set when calls to other agents are allowed.</td>
<td>0x00000040</td>
</tr>
<tr>
<td>Mask Name</td>
<td>Description</td>
<td>Numeric Value</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>DESK_OUTBOUND_ACCESS_INTERNATIONAL_MASK</td>
<td>Set when the agent can initiate international calls.</td>
<td>0x00000080</td>
</tr>
<tr>
<td>DESK_OUTBOUND_ACCESS_PUBLIC_NET_MASK</td>
<td>Set when the agent can initiate calls through the public network.</td>
<td>0x00000100</td>
</tr>
<tr>
<td>DESK_OUTBOUND_ACCESS_PRIVATE_NET_MASK</td>
<td>Set when the agent can initiate calls through the private network.</td>
<td>0x00000200</td>
</tr>
<tr>
<td>DESK_OUTBOUND_ACCESS_OPERATOR_ASSISTED_MASK</td>
<td>Set when the agent can initiate operator assisted calls.</td>
<td>0x00000400</td>
</tr>
<tr>
<td>DESK_OUTBOUND_ACCESS_PBX_MASK</td>
<td>Set when the agent can initiate outbound PBX calls.</td>
<td>0x00000800</td>
</tr>
<tr>
<td>DESK_NON_ACD_CALLS_ALLOWED_MASK</td>
<td>Set when the agent can place or handle non-ACD calls.</td>
<td>0x00001000</td>
</tr>
<tr>
<td>DESK_AGENT_CAN_SELECT_GROUP_MASK</td>
<td>Set when the agent can select which groups they are logged into.</td>
<td>0x00002000</td>
</tr>
</tbody>
</table>

**OnAgentGreetingControlConf**

The `OnAgentGreetingControlConf` event confirms the successful completion of the `SetAgentGreetingAction` request.

**Syntax**

**C++**

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

**COM**

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

**VB**

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

**Parameters**

**args**

Arguments array containing the following fields.

**Table 86: OnAgentGreetingControlConf Event Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageHeader</td>
<td>MHDR</td>
<td>Standard Message Header.</td>
</tr>
<tr>
<td>InvokeID</td>
<td>UINT</td>
<td>Set to the same value as the InvokeID from the corresponding request message.</td>
</tr>
</tbody>
</table>
### OnAgentInfoEvent

The OnAgentInfoEvent event is generated as a response to a query to the Agent Name Lookup Service and carries the agent's name. The CTI OS server generates this query when it is configured to do agent name lookup. The OnAgentInfoEvent event is sent to the client if the server obtained the information.

**Syntax**

- **C++**
  ```cpp
cpp
  void OnAgentInfoEvent(Arguments& args)
  ```
- **COM**
  ```cpp
  void OnAgentInfoEvent (IArguments * args)
  ```
- **VB**
  ```vbnet
  session_OnAgentInfoEvent (ByVal args As CtiosCLIENTLib.IArguments)
  ```

**Parameters**

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

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>A unique object ID for the Agent object.</td>
</tr>
<tr>
<td>AgentLastName</td>
<td>STRING</td>
<td>Agent's last name.</td>
</tr>
<tr>
<td>AgentFirstName</td>
<td>STRING</td>
<td>Agent's first name.</td>
</tr>
</tbody>
</table>

### OnAgentStateChange

The OnAgentStateChange event is generated when the agent state at the ACD changes. This can be as a response to a Login, Logout, or SetAgentState request.

**Syntax**

- **C++**
  ```cpp
  void OnAgentStateChange(Arguments& args)
  ```
- **COM**
  ```cpp
  void OnAgentStateChange (IArguments * args)
  ```
- **VB**
  ```vbnet
  session_OnAgentStateChange (ByVal args As CtiosCLIENTLib.IArguments)
  ```

**Parameters**

- **args**
  Arguments array containing the following fields.
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the agent state change occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>AgentState</td>
<td>SHORT</td>
<td>One of the values in Table 89: AgentState values, on page 234 representing the current overall state of the associated agent.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The optional, user-defined number of the agent SkillGroup affected by the state change, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup affected by the state change. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>StateDuration</td>
<td>INT</td>
<td>The number of seconds since the agent entered this state (typically 0).</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>EventReasonCode</td>
<td>SHORT</td>
<td>A peripheral-specific code indicating the reason for the state change.</td>
</tr>
<tr>
<td>SkillGroupState</td>
<td>SHORT</td>
<td>Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>The agent's ACD login ID.</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING</td>
<td>The agent's ACD teleset extension.</td>
</tr>
<tr>
<td>CTIClientSignature (Optional)</td>
<td>STRING</td>
<td>The Client Signature of the CTI Client that is associated with this agent.</td>
</tr>
</tbody>
</table>
Enablement Mask

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when the agent is on this state.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

UniqueObjectID

| STRING          | A unique object ID for the Agent object. |

AgentInstrument

| STRING          | The agent's ACD instrument number. |

The following table provides the AgentState values.

**Table 89: AgentState values**

<table>
<thead>
<tr>
<th>enum Value</th>
<th>Description</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eLogin</td>
<td>The agent has logged on to the ACD. It does not necessarily indicate that the agent is ready to accept calls.</td>
<td>0</td>
</tr>
<tr>
<td>eLogout</td>
<td>The agent has logged out of the ACD and cannot accept any additional calls.</td>
<td>1</td>
</tr>
<tr>
<td>eNotReady</td>
<td>The agent is unavailable for any call work.</td>
<td>2</td>
</tr>
<tr>
<td>eAvailable</td>
<td>The agent is ready to accept a call.</td>
<td>3</td>
</tr>
<tr>
<td>eTalking</td>
<td>The agent is currently talking on a call (inbound, outbound, or inside).</td>
<td>4</td>
</tr>
<tr>
<td>eWorkNotReady</td>
<td>The agent is performing after call work, but will not be ready to receive a call when completed.</td>
<td>5</td>
</tr>
<tr>
<td>eWorkReady</td>
<td>The agent is performing after call work, and will be ready to receive a call when completed.</td>
<td>6</td>
</tr>
<tr>
<td>eBusyOther</td>
<td>The agent is busy performing a task associated with another active SkillGroup.</td>
<td>7</td>
</tr>
<tr>
<td>eReserved</td>
<td>The agent is reserved for a call that will arrive at the ACD shortly.</td>
<td>8</td>
</tr>
<tr>
<td>eUnknown</td>
<td>The agent state is currently unknown.</td>
<td>9</td>
</tr>
<tr>
<td>enum Value</td>
<td>Description</td>
<td>Numeric Value</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>eHold</td>
<td>The agent currently has all calls on hold.</td>
<td>10</td>
</tr>
</tbody>
</table>

**Note**

Not all switches support all the states listed in the above table, and you should not make any assumptions about which states are supported on a particular switch without verification.

## OnAgentStatistics

The OnAgentStatistics event is fired to the client to indicate that the CTI server received a request to enable agent statistics (via the EnableAgentStatistics method). The arrival of events event is determined by the configuration on the server.

The table under Parameters details all the agent statistics that can be received. To optimize bandwidth, the default configuration on the server is set to minimize the agent statistics sent. Only the statistics that the Agent Statistics grid is configured for are sent to the client. For more information about on how to configure the agent statistics grid and minimize agent statistics, see *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted*.

### Syntax

**C++**

```cpp
void OnAgentStatistics (Arguments & args);
```

**COM**

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

**VB**

```vb
Session_ OnAgentStatistics (ByVal args as CTIOSCLIENTLIB.IArguments)
```

### Parameters

**args**

Arguments array containing the following fields.

**Table 90: OnAgentStatistics Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>The Unified ICM PeripheralID of the ACD where the agent is located.</td>
<td>INT</td>
</tr>
<tr>
<td>AgentExtension (required)</td>
<td>The agent's ACD teleset extension.</td>
<td>STRING</td>
</tr>
<tr>
<td>AgentID (required)</td>
<td>The agent's ACD login ID.</td>
<td>STRING</td>
</tr>
<tr>
<td>AgentInstrument (required)</td>
<td>The agent's ACD instrument number.</td>
<td>STRING</td>
</tr>
</tbody>
</table>
The OnAgentStatistics event contains all the agent statistics fields necessary to display the statistics configured on the CTI OS server.

**OnChatMessage**

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

**Syntax**

**C++**

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

**COM**

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

**VB**

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

**Parameters**

**args**

Arguments array containing the following fields.

**Table 91: OnChatMessage Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution</td>
<td>STRING</td>
<td>Currently the only supported value is “agent”.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>The AgentID of the message target.</td>
</tr>
<tr>
<td>Target</td>
<td>STRING</td>
<td>The AgentID of the message target.</td>
</tr>
<tr>
<td>Message</td>
<td>STRING</td>
<td>The text message provided by the sender.</td>
</tr>
<tr>
<td>Source</td>
<td>STRING</td>
<td>The AgentID of the message sender.</td>
</tr>
</tbody>
</table>

**OnControlFailureConf**

The OnControlFailureConf event is generated when the previously issued request, identified by the InvokeID field failed. It is sent in place of the corresponding confirmation message for that request.

**Syntax**

**C++**

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

**COM**

```cpp
void OnControlFailureConf (IArguments * args)
```
VB

session_OnControlFailureConf (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.

Table 92: OnControlFailureConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvokeID</td>
<td>INT</td>
<td>InvokeID of the request that failed</td>
</tr>
<tr>
<td>FailureCode</td>
<td>SHORT</td>
<td>A value specifying the reason that the request failed. For a list of the Control Failure Code see the table below.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID that represents a specific client.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>An object ID that uniquely identifies the Call object.</td>
</tr>
<tr>
<td>MessageType</td>
<td>INT</td>
<td>Contains the CTI OS Command Request ID that failed to execute. The message types included in this parameter are those to used to control Call, Agent State and Supervisor actions. For more information, see CTIOS Keywords andEnumerated Types, on page 499.</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>STRING</td>
<td>String text containing the description of the failure.</td>
</tr>
</tbody>
</table>

Table 93: Control Failure Codes

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTI_NO_ERROR</td>
<td>No error occurred.</td>
<td>0</td>
</tr>
<tr>
<td>E_CTI_INVALID_VERSION</td>
<td>The CTI Server does not support the protocol version number requested by the CTI client.</td>
<td>1</td>
</tr>
<tr>
<td>E_CTI_INVALID_MESSAGE_TYPE</td>
<td>A message with an invalid message type field was received.</td>
<td>2</td>
</tr>
<tr>
<td>Status Code</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>E_CTI_INVALID_FIELD_TAG</td>
<td>A message with an invalid floating field tag was received.</td>
<td>3</td>
</tr>
<tr>
<td>E_CTI_SESSION_NOT_OPEN</td>
<td>No session is currently open on the connection.</td>
<td>4</td>
</tr>
<tr>
<td>E_CTI_SESSION_ALREADY_OPEN</td>
<td>A session is already open on the connection.</td>
<td>5</td>
</tr>
<tr>
<td>E_CTI_REQUIRED_DATA_MISSING</td>
<td>The request did not include one or more floating items that are required.</td>
<td>6</td>
</tr>
<tr>
<td>E_CTI_INVALID_PERIPHERAL_ID</td>
<td>A message with an invalid PeripheralID value was received.</td>
<td>7</td>
</tr>
<tr>
<td>E_CTI_INVALID_AGENT_DATA</td>
<td>The provided agent data items are invalid.</td>
<td>8</td>
</tr>
<tr>
<td>E_CTI_AGENT_NOT_LOGGED_ON</td>
<td>The indicated agent is not currently logged in.</td>
<td>9</td>
</tr>
<tr>
<td>E_CTI_DEVICE_IN_USE</td>
<td>The indicated agent teleset is already associated with a different CTI client.</td>
<td>10</td>
</tr>
<tr>
<td>E_CTI_NEW_SESSION_OPENED</td>
<td>This session is being terminated due to a new session open request from the client.</td>
<td>11</td>
</tr>
<tr>
<td>E_CTI_FUNCTION_NOT_AVAILABLE</td>
<td>A request message was received for a function or service that was not granted to the client.</td>
<td>12</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALLID</td>
<td>A request message was received with an invalid CallID value.</td>
<td>13</td>
</tr>
<tr>
<td>E_CTI_PROTECTED_VARIABLE</td>
<td>The CTI client cannot update the requested variable.</td>
<td>14</td>
</tr>
<tr>
<td>E_CTI_CTI_SERVER_OFFLINE</td>
<td>The CTI Server cannot function normally. The CTI client closes the session upon receipt of this error.</td>
<td>15</td>
</tr>
<tr>
<td>E_CTI_TIMEOUT</td>
<td>The CTI Server failed to respond to a request message within the time-out period, or no messages were received from the CTI client within the IdleTimeout period.</td>
<td>16</td>
</tr>
<tr>
<td>E_CTI_UNSPECIFIED_FAILURE</td>
<td>An unspecified error occurred.</td>
<td>17</td>
</tr>
<tr>
<td>E_CTI_INVALID_TIMEOUT</td>
<td>The IdleTimeout field contains a value that is less than 20 seconds (4 times the minimum heartbeat interval of 5 seconds).</td>
<td>18</td>
</tr>
<tr>
<td>E_CTI_INVALID_SERVICE_MASK</td>
<td>The ServicesRequested field has unused bits set. All unused bit positions must be zero.</td>
<td>19</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_MSG_MASK</td>
<td>The CallMsgMask field has unused bits set. All unused bit positions must be zero.</td>
<td>20</td>
</tr>
<tr>
<td>E_CTI_INVALID_AGENT_STATE_MASK</td>
<td>The AgentStateMask field has unused bits set. All unused bit positions must be zero.</td>
<td>21</td>
</tr>
<tr>
<td>Status Code</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>E_CTI_INVALID_RESERVED_FIELD</td>
<td>A Reserved field has a non-zero value.</td>
<td>22</td>
</tr>
<tr>
<td>E_CTI_INVALID_FIELD_LENGTH</td>
<td>A floating field exceeds the allowable length for that field type.</td>
<td>23</td>
</tr>
<tr>
<td>E_CTI_INVALID_DIGITS</td>
<td>A STRING field contains characters that are not digits (“0” through “9”).</td>
<td>24</td>
</tr>
<tr>
<td>E_CTI_BAD_MESSAGE_FORMAT</td>
<td>The message is improperly constructed. This can be caused by omitted or incorrectly sized fixed message fields.</td>
<td>25</td>
</tr>
<tr>
<td>E_CTI_INVALID_TAG_FOR_MSG_TYPE</td>
<td>A floating field tag is present that specifies a field that does not belong in this message type.</td>
<td>26</td>
</tr>
<tr>
<td>E_CTI_INVALID_DEVICE_ID_TYPE</td>
<td>A DeviceIDType field contains an invalid value.</td>
<td>27</td>
</tr>
<tr>
<td>E_CTI_INVALID_LCL_CONN_STATE</td>
<td>A LocalConnectionState field contains an invalid value.</td>
<td>28</td>
</tr>
<tr>
<td>E_CTI_INVALID_EVENT_CAUSE</td>
<td>An EventCause field contains an invalid value.</td>
<td>29</td>
</tr>
<tr>
<td>E_CTI_INVALID_NUM_PARTIES</td>
<td>The NumParties field contains a value that exceeds the maximum (16).</td>
<td>30</td>
</tr>
<tr>
<td>E_CTI_INVALID_SYS_EVENT_ID</td>
<td>The SystemEventID field contains an invalid value.</td>
<td>31</td>
</tr>
<tr>
<td>E_CTI_INCONSISTENT_AGENT_DATA</td>
<td>The provided agent extension, agent ID, and/or agent instrument values are inconsistent with each other.</td>
<td>32</td>
</tr>
<tr>
<td>E_CTI_INVALID_CONNECTION_ID_TYPE</td>
<td>A ConnectionDeviceIDType field contains an invalid value.</td>
<td>33</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_TYPE</td>
<td>The CallType field contains an invalid value.</td>
<td>34</td>
</tr>
<tr>
<td>E_CTI_NOT_CALL_PARTY</td>
<td>A CallDataUpdate or Release Call request specified a call that the client is not a party to.</td>
<td>35</td>
</tr>
<tr>
<td>E_CTI_INVALID_PASSWORD</td>
<td>The ClientID and Client Password provided in an OPEN_REQ message is incorrect.</td>
<td>36</td>
</tr>
<tr>
<td>E_CTI_CLIENT_DISCONNECTED</td>
<td>The client TCP/IP connection was disconnected without a CLOSE_REQ.</td>
<td>37</td>
</tr>
<tr>
<td>E_CTI_INVALID_OBJECT_STATE</td>
<td>An invalid object state value was provided.</td>
<td>38</td>
</tr>
<tr>
<td>E_CTI_INVALID_NUM_SKILL_GROUPS</td>
<td>An invalid NumSkillGroups value was provided.</td>
<td>39</td>
</tr>
<tr>
<td>E_CTI_INVALID_NUM_LINES</td>
<td>An invalid NumLines value was provided.</td>
<td>40</td>
</tr>
<tr>
<td>Status Code</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>E_CTI_INVALID_LINE_TYPE</td>
<td>An invalid LineType value was provided.</td>
<td>41</td>
</tr>
<tr>
<td>E_CTI_INVALID_ALLOCATION_STATE</td>
<td>An invalid AllocationState value was provided.</td>
<td>42</td>
</tr>
<tr>
<td>E_CTI_INVALID_ANSWERING_MACHINE</td>
<td>An invalid AnsweringMachine value was provided.</td>
<td>43</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_MANNER_TYPE</td>
<td>An invalid CallMannerType value was provided.</td>
<td>44</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_PLACEMENT_TYPE</td>
<td>An invalid CallPlacementType value was provided.</td>
<td>45</td>
</tr>
<tr>
<td>E_CTI_INVALIDCONSULT_TYPE</td>
<td>An invalid ConsultType value was provided.</td>
<td>46</td>
</tr>
<tr>
<td>E_CTI_INVALID_FACILITY_TYPE</td>
<td>An invalid FacilityType value was provided.</td>
<td>47</td>
</tr>
<tr>
<td>E_CTI_INVALID_MSG_TYPE_FOR_VERSION</td>
<td>The provided MessageType is invalid for the opened protocol version.</td>
<td>48</td>
</tr>
<tr>
<td>E_CTI_INVALID_TAG_FOR_VERSION</td>
<td>A floating field tag value is invalid for the opened protocol version.</td>
<td>49</td>
</tr>
<tr>
<td>E_CTI_INVALID_AGENT_WORK_MODE</td>
<td>An invalid AgentWorkMode value was provided.</td>
<td>50</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_OPTION</td>
<td>An invalid call option value was provided.</td>
<td>51</td>
</tr>
<tr>
<td>E_CTI_INVALID_DESTINATION_COUNTRY</td>
<td>An invalid destination country value was provided.</td>
<td>52</td>
</tr>
<tr>
<td>E_CTI_INVALID_ANSWER_DETECT_MODE</td>
<td>An invalid answer detect mode value was provided.</td>
<td>53</td>
</tr>
<tr>
<td>E_CTI_MUTUALLY_EXCLUS_DEVICEID_TYPES</td>
<td>A peripheral monitor request cannot specify both a call and a device.</td>
<td>54</td>
</tr>
<tr>
<td>E_CTI_INVALID_MONITORID</td>
<td>An invalid monitorID value was provided.</td>
<td>55</td>
</tr>
<tr>
<td>E_CTI_SESSION_MONITOR_ALREADY_EXISTS</td>
<td>A requested session monitor was already created.</td>
<td>56</td>
</tr>
<tr>
<td>E_CTI_SESSION_MONITOR_IS_CLIENTS</td>
<td>A client may not monitor its own session.</td>
<td>57</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_CONTROL_MASK</td>
<td>An invalid call control mask value was provided.</td>
<td>58</td>
</tr>
<tr>
<td>E_CTI_INVALID_FEATURE_MASK</td>
<td>An invalid feature mask value was provided.</td>
<td>59</td>
</tr>
<tr>
<td>E_CTI_INVALID_TRANSFER_CONFERENCE_SETUP_MASK</td>
<td>An invalid transfer conference setup mask value was provided.</td>
<td>60</td>
</tr>
<tr>
<td>Status Code</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>E_CTI_INVALID_ARRAY_INDEX</td>
<td>An invalid named array index value was provided.</td>
<td>61</td>
</tr>
<tr>
<td>E_CTI_INVALID_CHARACTER</td>
<td>An invalid character value was provided.</td>
<td>62</td>
</tr>
<tr>
<td>E_CTI_CLIENT_NOT_FOUND</td>
<td>There is no open session with a matching ClientID.</td>
<td>63</td>
</tr>
<tr>
<td>E_CTI_SUPERVISOR_NOT_FOUND</td>
<td>The agent's supervisor is unknown or does not have an open CTI session.</td>
<td>64</td>
</tr>
<tr>
<td>E_CTI_TEAM_NOT_FOUND</td>
<td>The agent is not a member of an agent team.</td>
<td>65</td>
</tr>
<tr>
<td>E_CTI_NO_CALL_ACTIVE</td>
<td>The specified agent does not have an active call.</td>
<td>66</td>
</tr>
<tr>
<td>E_CTI_NAMED_VARIABLE_NOT_CONFIGURED</td>
<td>The specified named variable is not configured in the Unified ICM database.</td>
<td>67</td>
</tr>
<tr>
<td>E_CTI_NAMED_ARRAY_NOT_CONFIGURED</td>
<td>The specified named array is not configured in the Unified ICM database.</td>
<td>68</td>
</tr>
<tr>
<td>E_CTI_INVALID_CALL_VARIABLE_MASK</td>
<td>The specified call variable mask in not valid.</td>
<td>69</td>
</tr>
<tr>
<td>E_CTI_ELEMENT_NOT_FOUND</td>
<td>An internal error occurred manipulating a named variable or named array element.</td>
<td>70</td>
</tr>
<tr>
<td>E_CTI_INVALID_DISTRIBUTION_TYPE</td>
<td>The specified distribution type is invalid.</td>
<td>71</td>
</tr>
<tr>
<td>E_CTI_INVALID_SKILL_GROUP</td>
<td>The specified skill group is invalid.</td>
<td>72</td>
</tr>
<tr>
<td>E_CTI_TOO_MUCH_DATA</td>
<td>The total combined size of named variables and named arrays cannot exceed the limit of 2000 bytes.</td>
<td>73</td>
</tr>
<tr>
<td>E_CTI_VALUE_TOO_LONG</td>
<td>The value of the specified named variable or named array element exceeds the maximum permissible length.</td>
<td>74</td>
</tr>
<tr>
<td>E_CTI_SCALAR_FUNCTION_ON_ARRAY</td>
<td>A NamedArray was specified with a NamedVariable tag.</td>
<td>75</td>
</tr>
<tr>
<td>E_CTI_ARRAY_FUNCTION_ON_SCALAR</td>
<td>A NamedVariable was specified with a NamedArray tag.</td>
<td>76</td>
</tr>
<tr>
<td>E_CTI_INVALID_NUM_NAMED_VARIABLES</td>
<td>The value in the NumNamedVariables field is different than the number of NamedVariable floating fields in the message.</td>
<td>77</td>
</tr>
<tr>
<td>E_CTI_INVALID_NUM_NAMEDARRAYS</td>
<td>The value in the NumNamedArrays field is different than the number of NamedArray floating fields in the message.</td>
<td>78</td>
</tr>
</tbody>
</table>
OnEmergencyCall

The OnEmergencyCall event indicates that a CTI client (with Supervisory capabilities) is handling the indicated call as an emergency call. This event only applies to ACDs with Supervisor capabilities.

**Syntax**

C++

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

COM

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

VB

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

**Parameters**

`args`

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the call is located.</td>
</tr>
<tr>
<td>Connection CallID</td>
<td>INT</td>
<td>The Call ID value assigned to the call by the peripheral or the Unified ICM.</td>
</tr>
<tr>
<td>ConnectionDevice IDType</td>
<td>SHORT</td>
<td>Indicates the type of the connection identifier supplied in the ConnectionDeviceID floating field.</td>
</tr>
<tr>
<td>SessionID</td>
<td>INT</td>
<td>The CTI client SessionID of the CTI client making the notification.</td>
</tr>
<tr>
<td>Connection DeviceID</td>
<td>INT</td>
<td>The identifier of the connection between the call and the agent's device.</td>
</tr>
<tr>
<td>ClientID (required)</td>
<td>STRING</td>
<td>The ClientID of the client making the notification.</td>
</tr>
<tr>
<td>ClientAddress (Required)</td>
<td>STRING</td>
<td>The IP address of the client making the notification.</td>
</tr>
<tr>
<td>AgentExtension (Required)</td>
<td>STRING</td>
<td>The agent's teleset extension.</td>
</tr>
<tr>
<td>AgentID (required)</td>
<td>STRING</td>
<td>The agent's ACD login ID.</td>
</tr>
</tbody>
</table>
### OnLogoutFailed

The `OnLogoutFailed` is always generated before (or with) an `OnControlFailureConf` event and is identical to it but is generated only when a Logout request fails.

#### Syntax

- **C++:** `void OnLogoutFailed (Arguments& args)`
- **COM:** `void OnLogoutFailed (IArguments * args)`
- **VB:** `session_OnLogoutFailed (ByVal args As CtiosCLIENTLib.IArguments)`

#### Parameters

- **args**

  Arguments array containing the following fields.

  **Table 95: OnLogoutFailed Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvokeID</td>
<td>INT</td>
<td>InvokeID of the request that failed.</td>
</tr>
<tr>
<td>FailureCode</td>
<td>SHORT</td>
<td>A value specifying the reason that the request failed. For a list of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Control Failure Codes see Table 92: <code>OnControlFailureConf</code> Parameters, on page 237.</td>
</tr>
</tbody>
</table>

### OnMakeCallConf

The `OnMakeCallConf` event confirms the successful completion of the MakeCall request. It conveys the information detailed in the table under Parameters.

#### Syntax

- **C++**

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

VB
Session_OnMakeCallConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Java
void OnMakeCallConf (Arguments args);

Parameters
args
Arguments array containing the following fields.

Table 96: OnMakeCallConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewConnectionCallID</td>
<td>The Call ID value assigned to the call by the peripheral or the Unified ICM.</td>
<td>UINT</td>
</tr>
<tr>
<td>NewConnectionDevice IDType</td>
<td>Indicates the type of the connection identifier supplied in the New ConnectionDeviceID floating field.</td>
<td>SHORT</td>
</tr>
<tr>
<td>LineHandle</td>
<td>Identifies the teleset line used, if known. Otherwise this field is set to 0xffff.</td>
<td>SHORT</td>
</tr>
<tr>
<td>LineType</td>
<td>Indicates the type of the teleset line given in the LineHandle field.</td>
<td>SHORT</td>
</tr>
<tr>
<td>NewConnectionDeviceID</td>
<td>The identifier of the connection between the call and the device.</td>
<td>STRING</td>
</tr>
</tbody>
</table>

OnNewAgentTeamMember

The OnNewAgentTeamMember event informs the supervisor about a new agent team member. The event is typically received in response to a RequestAgentTeamList request from the supervisor object. It is also received when CTI OS Server receives an AGENT_TEAM_CONFIG_EVENT indicating a change in agent team configuration (add/remove).

Syntax

C++
void OnNewAgentTeamMember (Arguments& args)

COM
void OnNewAgentTeamMember (IArguments * args)

VB
session_OnNewAgentTeamMember (ByVal args As CtiosCLIENTLib.IArguments)
Parameters

args

Arguments array that can contain the following fields. Not all fields are always returned. Skillgroup and AgentInstrument are not returned if the agent is not logged in.

Table 97: OnNewAgentTeamMember Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>STRING</td>
<td>The Unified ICM PeripheralID of the agent's ACD.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>Unique object ID of the Agent object for this agent.</td>
</tr>
<tr>
<td>AgentState</td>
<td>SHORT</td>
<td>One of the values in Table 89: AgentState values, on page 234 representing the current state of the associated agent.</td>
</tr>
<tr>
<td>NumSkillGroups</td>
<td>INT</td>
<td>The number of skill groups that the agent is currently associated with, up to a maximum of 99.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent's ACD login.</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING</td>
<td>Agent's ACD teleset extension.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>Agent's ACD instrument number.</td>
</tr>
<tr>
<td>AgentLastName</td>
<td>STRING</td>
<td>Agent's last name.</td>
</tr>
<tr>
<td>AgentFirstName</td>
<td>STRING</td>
<td>Agent's first name.</td>
</tr>
<tr>
<td>AgentName</td>
<td>STRING</td>
<td>Agent's full name.</td>
</tr>
<tr>
<td>AgentAvailabilityStatus</td>
<td>SHORT</td>
<td>The current status of the agent's availability to take a call.</td>
</tr>
<tr>
<td>EventReasonCode</td>
<td>SHORT</td>
<td>A peripheral-specific code indicating the reason for the change in agent state to NotReady.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when the agent is on the state specified in the AgentState field.</td>
</tr>
<tr>
<td>SupervisorID</td>
<td>STRING</td>
<td>The ID of the agent's supervisor.</td>
</tr>
</tbody>
</table>
OnPostLogout

The OnPostLogout event is generated after the agent has logged out. Arrival of this event guarantees that the agent state event signalling the agent's transition to logout state was received and handled by all interested event listeners.

Syntax

C++

    void OnPostLogout (Arguments& args)

COM

    void OnPostLogout (IArguments * args)

VB

    session_OnPostLogout (ByVal args As CtiosCLIENTLib.IArguments)
### Parameters

**args**

Arguments array containing the following fields.

#### Table 98: OnPostLogout Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the agent state change occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>AgentState</td>
<td>SHORT</td>
<td>One of the values in Table 89: AgentState values, on page 234 representing the current overall state of the associated agent.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The optional, user-defined number of the agent SkillGroup affected by the state change, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup affected by the state change. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>StateDuration</td>
<td>INT</td>
<td>The number of seconds since the agent entered this state (typically 0).</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>EventReasonCode</td>
<td>SHORT</td>
<td>A peripheral-specific code indicating the reason for the state change.</td>
</tr>
<tr>
<td>SkillGroupState</td>
<td>SHORT</td>
<td>Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>The agent's ACD login ID.</td>
</tr>
</tbody>
</table>
### OnPreLogout

The OnPreLogout event just before the agent is logged out. It allows for any cleanup or logic that needs to be done before logout is completed.

**Syntax**

**C++**

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

**COM**

```com
void OnPreLogout (IArguments * args)
```

**VB**

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

**Parameters**

- **args**

  Arguments array containing the following fields.

### Remarks

When PG failover occurs, the client application can receive an OnPostLogout event with an EventReasonCode of CTIOS_FORCED_LOGOUT_REASON_CODE. For example, this can happen on an Unified CCE system after reconnecting to a different server during a failover, because there is a race condition of the PG logging the agent out and the client reconnecting to the other server before it happens. If this happens, the client application should not disconnect from CTIOSServer.

---

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentExtension</td>
<td>STRING</td>
<td>The agent's ACD teleset extension.</td>
</tr>
<tr>
<td>CTIClientSignature (Optional)</td>
<td>STRING</td>
<td>The Client Signature of the CTI Client that is associated with this agent.</td>
</tr>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when the agent is on this state.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>A unique object ID for the Agent object.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>The agent's ACD instrument number.</td>
</tr>
</tbody>
</table>
### Table 99: OnPreLogout Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the agent state change occurred.</td>
</tr>
<tr>
<td>PeripheralType</td>
<td>SHORT</td>
<td>The type of the peripheral.</td>
</tr>
<tr>
<td>AgentState</td>
<td>SHORT</td>
<td>One of the values in Table 89: AgentState values, on page 234 representing the current overall state of the associated agent.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The optional, user-defined number of the agent SkillGroup affected by the state change, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup affected by the state change. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>StateDuration</td>
<td>INT</td>
<td>The number of seconds since the agent entered this state (typically 0).</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>EventReasonCode</td>
<td>SHORT</td>
<td>A peripheral-specific code indicating the reason for the state change.</td>
</tr>
<tr>
<td>SkillGroupState</td>
<td>SHORT</td>
<td>Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>The agent's ACD login ID.</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING</td>
<td>The agent's ACD teleset extension.</td>
</tr>
</tbody>
</table>
| CTIClientSignature(Optional) | STRING | The Client Signature of the CTI Client that is associated with this agent. }}
### OnQueryAgentStateConf

The `OnQueryAgentStateConf` event is generated and returned by the server at login as a response to the `QueryAgentState()` request. A user cannot issue this request.

**Syntax**

**C++**

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

**COM**

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

**VB**

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

**Parameters**

`args`

Arguments array containing the following fields.

**Table 100: OnQueryAgentStateConf Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent's ACD login.</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING</td>
<td>Agent's ACD teleset extension.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>Agent's ACD instrument number.</td>
</tr>
<tr>
<td>AgentState</td>
<td>SHORT</td>
<td>One of the values in Table 89: AgentState values, on page 234 representing the current state of the associated agent.</td>
</tr>
<tr>
<td>NumSkillGroups</td>
<td>INT</td>
<td>The number of skill groups that the agent is currently associated with, up to a maximum of 20.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SkillGroup[j]</td>
<td>ARGUMENTS</td>
<td>Argument array that contains Skill Group information for the j-th element less than NumSkillGroups. The message contains NumSkillGroups elements of this type.</td>
</tr>
<tr>
<td>MRDID</td>
<td>INT</td>
<td>Media Routing Domain ID as configured in Unified ICM and the ARM client.</td>
</tr>
<tr>
<td>NumTasks</td>
<td>INT</td>
<td>The number of tasks currently assigned to the agent—this is the number that Unified ICM compares to the MaxTaskLimit to decide if the agent is available to be assigned additional tasks. This includes active tasks as well as those that are offered, paused, and in wrapup.</td>
</tr>
<tr>
<td>AgentMode</td>
<td>SHORT</td>
<td>The mode that the agent is not in when the login completes. ROUTABLE = 0, NOT ROUTABLE = 1</td>
</tr>
<tr>
<td>MaxTaskLimit</td>
<td>INT</td>
<td>The maximum number of tasks that the agent can simultaneously work on.</td>
</tr>
<tr>
<td>ICAgentID</td>
<td>INT</td>
<td>The Unified ICM Skill Target ID, a unique agent identifier for Unified ICM.</td>
</tr>
</tbody>
</table>
An agent is available to work on a task in this Media Routing Domain if the agent meets all of these conditions:

- The agent is routable for this Media Routing Domain.
- The agent is not in Not Ready state for skill groups in other Media Routing Domain.
- The agent is temp routable, meaning that the agent is not in Reserved, Active, Work-Ready, or Work-Not Ready state on a non-interruptible task in another Media Routing Domain.
- The agent has not reached the maximum task limit for this Media Routing Domain.

An available agent is eligible to be assigned a task. Who can assign a task to the agent is determined by whether or not the agent is Routable.

An agent is *ICMAvailable* in MRD X if he is available in X and Routable with respect to X. An agent is *ApplicationAvailable* in MRD X if he is available in X and not Routable with respect to X. Otherwise an agent is *NotAvailable* in MRD X.

- NOT AVAILABLE = 0,
- ICM AVAILABLE = 1,
- APPLICATION AVAILABLE = 2

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent Availability Status</td>
<td>INT</td>
<td>An agent is available to work on a task in this Media Routing Domain if the agent meets all of these conditions:</td>
</tr>
</tbody>
</table>

- • The agent is routable for this Media Routing Domain.
- • The agent is not in Not Ready state for skill groups in other Media Routing Domain.
- • The agent is temp routable, meaning that the agent is not in Reserved, Active, Work-Ready, or Work-Not Ready state on a non-interruptible task in another Media Routing Domain.
- • The agent has not reached the maximum task limit for this Media Routing Domain.

An available agent is eligible to be assigned a task. Who can assign a task to the agent is determined by whether or not the agent is Routable.

An agent is *ICMAvailable* in MRD X if he is available in X and Routable with respect to X. An agent is *ApplicationAvailable* in MRD X if he is available in X and not Routable with respect to X. Otherwise an agent is *NotAvailable* in MRD X.

- NOT AVAILABLE = 0,
- ICM AVAILABLE = 1,
- APPLICATION AVAILABLE = 2

Each SkillGroup field in the message contains the following information.

Each SkillGroup field in the message contains the following information.
Table 101: SkillGroup Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The optional, user-defined number of an agent SkillGroup queue that the call was added to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupPriority</td>
<td>SHORT</td>
<td>The priority of the skill group, or 0 when the skill group priority is not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupState</td>
<td>SHORT</td>
<td>One of the values representing the current state associated agent with respect to the skill group.</td>
</tr>
</tbody>
</table>

OnSetAgentModeEvent

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

Syntax

C++
void OnSetAgentModeEvent (Arguments& args)

COM
void OnSetAgentModeEvent (IArguments * args)

VB
Session_OnSetAgentModeEvent (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.

Table 102: OnSetAgentModeEven Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>STRING</td>
<td>ID of the Unified ICM Peripheral ACD associated with the agent.</td>
</tr>
</tbody>
</table>
OnSetAgentStateConf

The OnSetAgentStateConf confirmation message is fired to the client to indicate that the CTI server received the SetAgentState request. This confirmation message does not indicate that the agent has changed to the desired state; rather, the programmer should expect one or more OnAgentStateChange events to indicate the change of state.

Syntax

C++

    int OnSetAgentStateConf (Arguments & args);

COM

    HRESULT OnSetAgentStateConf ([out] IArguments * args);

VB

    Session_OnSetAgentStateConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Java

    void OnSetAgentStateConf (Arguments args);

Parameters

args

Not used; reserved for future use.
OnStartMonitoringAgent

The OnStartMonitoringAgent event is generated when a new agent is selected to be monitored in response to a StartMonitoringAgent() request.

Syntax

C++
void OnStartMonitoringAgent (Arguments& args)

COM
void OnStartMonitoringAgent (IArguments * args)

VB
session_OnStartMonitoringAgent (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.

Table 103: OnStartMonitoringAgent Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>Unique object ID for the supervisor object.</td>
</tr>
<tr>
<td>AgentReference</td>
<td>STRING</td>
<td>String containing the Agent ID for the agent to be monitored.</td>
</tr>
<tr>
<td>SupervisorID</td>
<td>STRING</td>
<td>String containing the supervisor's AgentID.</td>
</tr>
<tr>
<td>SupervisorKey</td>
<td>STRING</td>
<td>Supervisor's unique object ID.</td>
</tr>
<tr>
<td>BargedInCallID</td>
<td>STRING</td>
<td>If the supervisor has barged in on the agent's call, the unique object ID of that call.</td>
</tr>
<tr>
<td>Supervisor AgentState</td>
<td>STRING</td>
<td>The supervisor's agent state.</td>
</tr>
</tbody>
</table>

Remarks
This is a Supervisor specific event. It is supported for use with Unified CCE only.

OnStopMonitoringAgent

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

C++
void OnStopMonitoringAgent (Arguments & args)

COM
void OnStopMonitoringAgent (IArguments * args)

VB
session_OnStopMonitoringAgent (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.

Table 104: OnStopMonitoringAgent Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>Unique object ID for the supervisor object.</td>
</tr>
<tr>
<td>AgentReference</td>
<td>STRING</td>
<td>String containing the Agent ID for the agent to be monitored.</td>
</tr>
<tr>
<td>SupervisorID</td>
<td>STRING</td>
<td>String containing the supervisor's AgentID.</td>
</tr>
<tr>
<td>SupervisorKey</td>
<td>STRING</td>
<td>Supervisor's unique object ID.</td>
</tr>
<tr>
<td>BargedInCallID</td>
<td>STRING</td>
<td>If the supervisor has barged in on the agent's call, the unique object ID of that call.</td>
</tr>
<tr>
<td>Supervisor AgentState</td>
<td>STRING</td>
<td>The supervisor's agent state.</td>
</tr>
</tbody>
</table>

Remarks
This is a Supervisor specific event. It is supported for use with Unified CCE only.

OnUserMessageConf
Not supported.

ISkillGroupEvents 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. You can configure the update frequency of OnSkillGroupStatisticsUpdated on the CTI OS server (for more information, see CTI OS System Manager’s Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted).

Syntax

C++
void OnSkillGroupStatisticsUpdated (Arguments & args)

COM
void OnSkillGroupStatisticsUpdated (IArguments * args)

VB
skillgroup_ OnSkillGroupStatisticsUpdated (ByVal args As CtiosCLIENTLib.IArguments)

Parameters

args
Arguments array containing the following fields.

Table 105: OnSkillGroupStatisticsUpdated Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralsID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD on which the agent resides.</td>
</tr>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The optional, user-defined 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>INT</td>
<td>The system-assigned identifier of the skill group. May contain the special value NULL_SKILL_GROUP when not available.</td>
</tr>
</tbody>
</table>

The statistics event also contains all the statistics fields listed in Table 157: SkillGroup Statistics, on page 424 in a nested Arguments array named STATISTICS.

OnSkillInfoEvent

Provides information about a particular skill group. This event is sent to any client that has enabled skill group statistics.
### Syntax

**C++**

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

**COM**

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

**VB**

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

#### Parameters

**args**

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>Skill group number.</td>
</tr>
<tr>
<td>SkillGroupName</td>
<td>STRING</td>
<td>Skill group name associated with the skill group number above.</td>
</tr>
</tbody>
</table>

### IBUTTONENABLEMENTEVENTS

This interface allows a client application to receive events that indicate what buttons you can enable on the user interface, given the current agent and current call states.

### OnButtonEnablementChange

The OnButtonEnablementChange event is received by a client in agent mode whenever CIL receives an agent or call event that carries the EnablementMask field in its parameters. This event allows the client application to enable or disable elements on the user interface. The fields in the event are the same as in OnButtonEnablementChange.

**C++**

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

**COM**

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

**VB**

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

#### Parameters

**args**

Arguments array containing the following fields.
### Table 107: `OnButtonEnablementChange` Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call. For more information, see the table below.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>ID of the object (for example, agent, call) that the event is meant for.</td>
</tr>
<tr>
<td>MessageID</td>
<td>INT</td>
<td>The event that triggered the button enablement change.</td>
</tr>
</tbody>
</table>

Note: The following table represents the C++/COM/VB enumerations. Enumerations for Java are in the description of CtiOs_Enums.ButtonEnablement in the Javadoc. Reference bits by the enumeration rather than the actual number in the bit mask.

### Table 108: Table of Enablement Bits

<table>
<thead>
<tr>
<th>Button</th>
<th>Bit Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE_ALL</td>
<td>0x00400000</td>
</tr>
<tr>
<td>ENABLE_ANSWER</td>
<td>0x00000001</td>
</tr>
<tr>
<td>ENABLE_RELEASE</td>
<td>0x00000002</td>
</tr>
<tr>
<td>ENABLE_HOLD</td>
<td>0x00000004</td>
</tr>
<tr>
<td>ENABLE_RETRIEVE</td>
<td>0x00000008</td>
</tr>
<tr>
<td>ENABLE_MAKECALL</td>
<td>0x00000010</td>
</tr>
<tr>
<td>ENABLE_TRANSFER_INIT</td>
<td>0x00000020</td>
</tr>
<tr>
<td>ENABLE_TRANSFER_COMPLETE</td>
<td>0x00000040</td>
</tr>
<tr>
<td>ENABLE_SINGLE_STEP_TRANSFER</td>
<td>0x00000080</td>
</tr>
<tr>
<td>ENABLE_CONFERENCE_INIT</td>
<td>0x00000100</td>
</tr>
<tr>
<td>ENABLE_CONFERENCE_COMPLETE</td>
<td>0x00000200</td>
</tr>
<tr>
<td>ENABLE_SINGLE_STEP_CONFERENCE</td>
<td>0x00000400</td>
</tr>
<tr>
<td>ENABLE_ALTERNATE</td>
<td>0x00000800</td>
</tr>
<tr>
<td>ENABLE_RECONNECT</td>
<td>0x00001000</td>
</tr>
<tr>
<td>Button</td>
<td>Bit Mask</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>ENABLE_WRAPUP</td>
<td>0x00002000</td>
</tr>
<tr>
<td>ENABLE_INSIDE_MAKECALL</td>
<td>0x00004000</td>
</tr>
<tr>
<td>ENABLE_OUTSIDE_MAKECALL</td>
<td>0x00008000</td>
</tr>
<tr>
<td>ENABLE_SUPERVISOR_ASSIST</td>
<td>0x00010000</td>
</tr>
<tr>
<td>ENABLE_EMERGENCY_CALL</td>
<td>0x00020000</td>
</tr>
<tr>
<td>ENABLE_BAD_LINE_CALL</td>
<td>0x00040000</td>
</tr>
<tr>
<td>ENABLE_STATISTICS</td>
<td>0x00080000</td>
</tr>
<tr>
<td>ENABLE_CHAT</td>
<td>0x00100000</td>
</tr>
<tr>
<td>ENABLE_RECORD</td>
<td>0x00200000</td>
</tr>
<tr>
<td>ENABLE_LOGIN</td>
<td>0x01000000</td>
</tr>
<tr>
<td>ENABLE_LOGOUT</td>
<td>0x02000000</td>
</tr>
<tr>
<td>ENABLE_LOGOUT_WITH_REASON</td>
<td>0x04000000</td>
</tr>
<tr>
<td>ENABLE_READY</td>
<td>0x08000000</td>
</tr>
<tr>
<td>ENABLE_NOTREADY</td>
<td>0x10000000</td>
</tr>
<tr>
<td>ENABLE_NOTREADY_WITH_REASON</td>
<td>0x20000000</td>
</tr>
<tr>
<td>ENABLE_WORKREADY</td>
<td>0x40000000</td>
</tr>
<tr>
<td>ENABLE_WORKNOTREADY</td>
<td>0x80000000</td>
</tr>
<tr>
<td>DISABLE_READY</td>
<td>0xF7FFFFFF</td>
</tr>
<tr>
<td>DISABLE_NOTREADY</td>
<td>0xCFFFFFFF</td>
</tr>
<tr>
<td>DISABLE_WORKREADY</td>
<td>0xBFFFFFFF</td>
</tr>
</tbody>
</table>

**Supervisor Button Enablement Masks**

<table>
<thead>
<tr>
<th>Button</th>
<th>Bit Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE_SET_AGENT_LOGOUT</td>
<td>0x00000001</td>
</tr>
<tr>
<td>ENABLE_SET_AGENT_READY</td>
<td>0x00000002</td>
</tr>
<tr>
<td>ENABLE_SILENTMONITOR</td>
<td>0x00000004</td>
</tr>
<tr>
<td>ENABLE_BARGE_IN</td>
<td>0x00000004</td>
</tr>
<tr>
<td>ENABLE_INTERCEPT</td>
<td>0x00000008</td>
</tr>
<tr>
<td>ENABLE_CLEAR</td>
<td>0x00000010</td>
</tr>
<tr>
<td>ENABLE_START_SILENTMONITOR</td>
<td>0x00000020</td>
</tr>
</tbody>
</table>
### OnSupervisorButtonChange

A client in agent mode working as supervisor receives the `OnSupervisorButtonChange` event whenever CIL receives a Monitored Agent, Monitored call event that carries the SupervisorBtnEnablementMask field in its parameters. This event allows the client application to enable or disable elements on the user interface. The fields in the event are the same as in `OnButtonEnablementChange`.

#### C++

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

#### COM

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

#### VB

```vbnet
session_ OnSupervisorButtonChange (ByVal args As CtiosCLIENTLib.IArguments)
```

---

<table>
<thead>
<tr>
<th>Button</th>
<th>Bit Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000040 ENABLE_STOP_SILENTMONITOR</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFEDisable_SET_AGENT_LOGOUT</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFDDisable_SET_AGENT_READY</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFBDisable_SILENTMONITOR</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFBDisable_BARGE_IN</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFF7DISABLE_INTERCEPT</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFEFDisable_CLEAR</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFDFDisable_START_SILENTMONITOR</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFBFDisable_STOP_SILENTMONITOR</td>
<td></td>
</tr>
<tr>
<td>DISABLE_SUPERVISE_CALL</td>
<td>DISABLE_BARGE_IN &amp; DISABLE_INTERCEPT &amp; DISABLE_CLEAR &amp; DISABLE_SILENTMONITOR &amp; DISABLE_START_SILENTMONITOR &amp; DISABLE_STOP_SILENTMONITOR</td>
</tr>
<tr>
<td>DISABLE_SET_AGENT_STATE</td>
<td>DISABLE_SET_AGENT_LOGOUT, DISABLE_SET_AGENT_READY</td>
</tr>
<tr>
<td>DISABLE_ALL_AGENT_SELECT</td>
<td>DISABLE_BARGE_IN &amp; DISABLE_INTERCEPT &amp; DISABLE_CLEAR &amp; DISABLE_SILENTMONITOR &amp; DISABLE_START_SILENTMONITOR &amp; DISABLE_STOP_SILENTMONITOR</td>
</tr>
</tbody>
</table>
Parameters

args

Arguments array containing the following fields.

Table 109: OnSupervisorButtonChange Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupervisorBtn EnablementMask</td>
<td>INT</td>
<td>Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call. For more information, see Table 108: Table of Enablement Bits, on page 259.</td>
</tr>
</tbody>
</table>

Remarks

Supported for use with Unified CCE only.

IMonitoredAgentEvents Interface

Note

The events in this section are supported for use with Unified CCE only.

This interface fires Agent events to a supervisor for his team members. IMonitoredAgentEvents are triggered by the supervisor sending a StartMonitoringAllAgentTeams request (for more information, see Agent Object, on page 343). For more information about the event parameters, see the IAgentEvents interface.

The most common event handled is the OnMonitoredAgentStateChange event, which informs a supervisor of agent state changes of agents in the supervisor's team. All the parameters are the same as the regular OnAgentStateChange events, except for an additional keyword called CTIOS_MONITORED, which indicates that this event is for a monitored agent.

List of Monitored Agent events:

OnMonitoredAgentStateChange([in] IArguments *pIArguments);
OnMonitoredAgentInfoEvent([in] IArguments *pIArguments);

IMonitoredCallEvents Interface

Note

The events in this section are supported with Unified CCE only.

This interface fires Call events to a supervisor for one of his agent team members. When the supervisor sends a StartMonitoringAgent request (for more information, see Agent Object, on page 343), the supervisor starts
receiving MonitoredCallEvents for this “currently” monitored agent. Monitored call events are received until the supervisor sends a StopMonitoringAgent request for this agent.

The IMonitoredCallEvents interface includes OnMonitoredCallBegin, OnMonitoredCallEnd, and OnMonitoredCallDataUpdate as well as other call events (see list below). These events are described in detail for the ICallEventsInterface. The only difference is that the Arguments array contains an additional keyword call CTIOS_MONITORED, indicating that this event is for a monitored call.

List of Monitored Call events:

OnMonitoredCallBegin([in] IArguments *pIArguments);
OnMonitoredCallEnd([in] IArguments *pIArguments);
OnMonitoredCallDataUpdate([in] IArguments *pIArguments);
OnMonitoredCallDelivered([in] IArguments *pIArguments);
OnMonitoredCallEstablished([in] IArguments *pIArguments);
OnMonitoredCallHeld([in] IArguments *pIArguments);
OnMonitoredCallRetrieved([in] IArguments *pIArguments);
OnMonitoredCallCleared([in] IArguments *pIArguments);
OnMonitoredCallConnectionCleared([in] IArguments *pIArguments);
MonitoredCallReachedNetworkEvent([in] IArguments *pIArguments);
OnMonitoredCallOriginated([in] IArguments *pIArguments);
OnMonitoredCallFailed([in] IArguments *pIArguments);
OnMonitoredCallTransferred([in] IArguments *pIArguments);
OnMonitoredCallConferenced([in] IArguments *pIArguments);
OnMonitoredCallDiverted([in] IArguments *pIArguments);
OnMonitoredTranslationRoute([in] IArguments *pIArguments);
OnMonitoredCallAgentPrecallEvent([in] IArguments *pIArguments);
OnMonitoredCallAgentPrecallAbortEvent([in] IArguments *pIArguments);
MonitoredCallServiceInitiatedEvent([in] IArguments *pIArguments);
MonitoredCallQueuedEvent([in] IArguments *pIArguments);
MonitoredCallDequeuedEvent([in] IArguments *pIArguments);

ISilentMonitorEvents

The silent monitor manager object fires events on the ISilentMonitorEvents interface. The following events are published to subscribers of the ISilentMonitorEvents interface.

Note

The events in this section are supported with Unified CCE only.
The following events apply only to CTI OS based silent monitor unless noted otherwise.

**OnCallRTPStarted**

The OnCallRTPStarted event indicates that an RTP media stream has started. This event accompanies the Call object in an Unified CCE environment.

**Syntax**

**C++**

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

**COM**

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

**VB**

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

**Parameters**

**args**

Arguments array containing the following fields.

**Table 110: OnCallRTPStarted Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitorID</td>
<td>UINT</td>
<td>The Monitor ID of the device or call monitor that caused this message to be sent to the client, or zero if there is no monitor associated with the event (All Events Service).</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>UINT</td>
<td>The Unified ICM PeripheralID of the ACD where the device is located.</td>
</tr>
<tr>
<td>ClientPort</td>
<td>UINT</td>
<td>The TCP/IP port number of the CTI Client connection.</td>
</tr>
<tr>
<td>Direction</td>
<td>USHORT</td>
<td>The direction of the event. One of the following values: 0: Input; 1: Output; 2: Bi-directional.</td>
</tr>
</tbody>
</table>
### OnCallRTPStopped

The OnCallRTPStopped event indicates that an RTP media has stopped. This event accompanies the Call object in a Unified CCE environment.

#### Syntax

**C++**

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

**COM**

```cpp
void OnCallRTPStopped (IAArguments * args)
```

**VB**

```vb
session_OnCallRTPStopped (ByVal args As CtiosCLIENTLib.IArguments)
```

---

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTPType</td>
<td>USHORT</td>
<td>The type of the event. One of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Audio;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Video;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: Data.</td>
</tr>
<tr>
<td>BitRate</td>
<td>UINT</td>
<td>The media bit rate, used for g.723 payload only.</td>
</tr>
<tr>
<td>EchoCancellation</td>
<td>USHORT</td>
<td>on/off.</td>
</tr>
<tr>
<td>PacketSize</td>
<td>UINT</td>
<td>In milliseconds.</td>
</tr>
<tr>
<td>PayloadType</td>
<td>USHORT</td>
<td>The audio codec type.</td>
</tr>
<tr>
<td>ConnectionDeviceIDType</td>
<td>USHORT</td>
<td>Indicates the type of the connection identifier supplied in the ConnectionDeviceID floating field.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>UINT</td>
<td>The Call ID value assigned to this call by the peripheral or Unified ICM.</td>
</tr>
<tr>
<td>ConnectionDeviceID</td>
<td>STRING</td>
<td>The identifier of the connection between the call and the device.</td>
</tr>
<tr>
<td>ClientAddress</td>
<td>STRING</td>
<td>The IP address of the phone.</td>
</tr>
<tr>
<td>AgentID (optional)</td>
<td>STRING</td>
<td>The agent's ACD login ID.</td>
</tr>
<tr>
<td>AgentExtension (optional)</td>
<td>STRING</td>
<td>The agent's ACD teleset extension.</td>
</tr>
<tr>
<td>AgentInstrument (optional)</td>
<td>STRING</td>
<td>The agent's ACD instrument number.</td>
</tr>
</tbody>
</table>
## Parameters

args

Arguments array containing the following fields.

### Table 111: OnCallRTPStopped Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitorID</td>
<td>UINT</td>
<td>The Monitor ID of the device or call monitor that caused this message to be sent to the client, or zero if there is no monitor associated with the event (All Events Service).</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>UINT</td>
<td>The Unified ICM PeripheralID of the ACD where the device is located.</td>
</tr>
<tr>
<td>ClientPort</td>
<td>UINT</td>
<td>The TCP/IP port number of the CTI Client connection.</td>
</tr>
<tr>
<td>Direction</td>
<td>USHOR</td>
<td>The direction of the event. One of the following values: 0: Input; 1: Output; 2: Bi-directional.</td>
</tr>
<tr>
<td>RTPType</td>
<td>USHOR</td>
<td>The type of the event. One of the following values: 0: Audio; 1: Video; 2: Data.</td>
</tr>
<tr>
<td>BitRate</td>
<td>UINT</td>
<td>The media bit rate, used for g.723 payload only.</td>
</tr>
<tr>
<td>EchoCancellation</td>
<td>USHOR</td>
<td>on/off.</td>
</tr>
<tr>
<td>PacketSize</td>
<td>UINT</td>
<td>In milliseconds.</td>
</tr>
<tr>
<td>PayloadType</td>
<td>USHOR</td>
<td>The audio codec type.</td>
</tr>
<tr>
<td>ConnectionDevice IDType</td>
<td>USHOR</td>
<td>Indicates the type of the connection identifier supplied in the ConnectionDeviceID floating field.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>UINT</td>
<td>The Call ID value assigned to this call by the peripheral or Unified ICM.</td>
</tr>
</tbody>
</table>
The identifier of the connection between the call and the device.

<table>
<thead>
<tr>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Object ID of the object being monitored.</td>
<td>STRING</td>
</tr>
<tr>
<td>Agent ID of the agent to be monitored. This message contains either AgentID or DeviceID, but not both.</td>
<td>STRING</td>
</tr>
<tr>
<td>Device ID of the agent to be monitored. This message contains either AgentID or DeviceID, but not both.</td>
<td>STRING</td>
</tr>
</tbody>
</table>

### OnStartSilentMonitorConf

The OnStartSilentMonitorConf event is sent to the monitoring application to indicate that the CTI OS server has processed a StartSilentMonitorRequest.

#### Syntax

**C++**

```cpp
void OnStartSilentMonitorConf (Arguments & args);
```

**COM**

```cpp
HRESULT OnStartSilentMonitorConf ([in] Arguments* args);
```

**VB**

```vb
Session_ OnStartSilentMonitorConf (ByVal args as CTIOSCLIENTLIB.IArguments)
```

### Parameters

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

#### Table 112: OnStartSilentMonitorConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID of the agent to be monitored. This message contains either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>DeviceID</td>
<td>STRING</td>
<td>Device ID of the agent to be monitored. This message contains either AgentID or DeviceID, but not both.</td>
</tr>
</tbody>
</table>
### OnSilentMonitorStartedEvent

#### For CTI OS Based Silent Monitor

The OnSilentMonitorStartedEvent event is fired to the subscriber to indicate that a silent monitor session has started on its behalf and that audio transmission to the monitoring client has started.

#### Syntax

**C++**

```cpp
void OnSilentMonitorStartedEvent(Arguments & args);
```

**COM**

```cpp
HRESULT OnSilentMonitorStartedEvent([in] Arguments* args);
```

**VB**

```vb
Session_OnSilentMonitorStartedEvent(ByVal args As CTIOSCLIENTLIB.IArguments)
```

#### Parameters

*args*

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the silent monitor start was requested.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the monitoring application.</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port of the monitoring application.</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Heartbeat interval for the silent monitor session.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Timeout for no activity.</td>
</tr>
<tr>
<td>OriginatingServerID</td>
<td>STRING</td>
<td>TCP/IP Address:Port of the CTI OS server from which the request originated.</td>
</tr>
<tr>
<td>OriginatingClientID</td>
<td>STRING</td>
<td>Client Identification of the monitoring application.</td>
</tr>
</tbody>
</table>
### Table 113: OnSilentMonitorStartedEvent Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID of the agent to be monitored. This message contains either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>DeviceID</td>
<td>STRING</td>
<td>Device ID of the agent to be monitored. This message contains either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where silent monitoring started.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the monitoring application.</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port of the monitoring application.</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Heartbeat interval for the silent monitor session.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Timeout for no activity.</td>
</tr>
<tr>
<td>OriginatingServerID</td>
<td>STRING</td>
<td>TCP/IP Address: Port of the CTI OS server from which the request originated.</td>
</tr>
<tr>
<td>OriginatingClientID</td>
<td>STRING</td>
<td>Client Identification of the monitoring application.</td>
</tr>
</tbody>
</table>

### For CCM-Based Silent Monitor

When you configure CCM based silent monitor, this event tells the monitored application, for example an agent desktop, that it is being monitored. This event, in addition to call events for the silent monitor call, tells the monitoring application, for example a supervisor desktop, that silent monitor of the agent has begun.

**Note**

At failover, the desktop can receive multiple OnSilentMonitorStartedEvents.
OnSilentMonitorStartRequestedEvent

The OnSilentMonitorStartRequestedEvent event is fired to the subscriber to indicate that a silent monitor session request has arrived and that it will be established on its behalf if the DoDefaultMessageHandling parameter is set to True. The default behavior is to start sending audio and establish the session automatically. If the subscriber wishes to process the event by itself, they must set the DoDefaultMessageHandling parameter to False and invoke AcceptSilentMonitoring when it is ready to start the session and call ReportSMSessionStatus to the monitoring client.

CTIOS server generates this event whenever a remote application calls the StartSilentMonitorRequest method.

Syntax

C++

    void OnSilentMonitorStartRequestedEvent(Arguments & args);

COM

    HRESULT OnSilentMonitorStartRequestedEvent([in] Arguments* args);

Parameters

args

Arguments array containing the following fields.

Table 114: OnSilentMonitorStartRequestedEvent

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SilentMonitorInitiatingAgentUID</td>
<td>STRING</td>
<td>Unique object ID of the agent that initiated silent monitor.</td>
</tr>
<tr>
<td>SilentMonitorInitiatingDeviceID</td>
<td>STRING</td>
<td>ID of the device that initiated silent monitor.</td>
</tr>
<tr>
<td>SilentMonitorTargetAgentUID</td>
<td>STRING</td>
<td>Unique object ID of the silently monitored agent.</td>
</tr>
<tr>
<td>SilentMonitorTargetDeviceID</td>
<td>STRING</td>
<td>ID of the silently monitored device.</td>
</tr>
<tr>
<td>SilentMonitorCallUID</td>
<td>STRING</td>
<td>Unique object ID of the silent monitor call.</td>
</tr>
</tbody>
</table>
VB

Session_OnSilentMonitorStartRequestedEvent(ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args
Arguments array containing the following fields.

Table 115: OnSilentMonitorStartRequestedEvent Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID of the agent to be monitored. This message contains either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>DeviceID</td>
<td>STRING</td>
<td>Device ID of the agent to be monitored. This message contains either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the silent monitor start was requested.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the monitoring application.</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port of the monitoring application.</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Heartbeat interval for the silent monitor session.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Timeout for no activity.</td>
</tr>
<tr>
<td>OriginatingServerID</td>
<td>STRING</td>
<td>TCP/IP Address: Port of the CTI OS server from which the request originated.</td>
</tr>
<tr>
<td>OriginatingClientId</td>
<td>STRING</td>
<td>Client Identification of the monitoring application.</td>
</tr>
</tbody>
</table>
When this parameter is set to True, it instructs the SilentMonitorManager to immediately start sending audio and establish the silent monitor session. If this parameter is set to False, it instructs the SilentMonitorManager to not send voice and not establish the silent monitor session. In this case, it is the responsibility of the subscriber to report this status accordingly.

---

### OnSilentMonitorSessionDisconnected

The OnSilentMonitorSessionDisconnected event is sent to the application to report errors if the connection fails between the monitoring and monitored clients.

**Syntax**

**C++**

```cpp
void OnSilentMonitorSessionDisconnected (Arguments & args);
```

**COM**

```cpp
HRESULT OnSilentMonitorSessionDisconnected ([in] Arguments* args);
```

**VB**

```vb
Session_OnSilentMonitorSessionDisconnected (ByVal args as CTIOSCLIENTLIB.IArguments)
```

**Parameters**

`args`

Arguments array containing the following fields.

**Table 116: OnSilentMonitorSessionDisconnected Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored.</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
<tr>
<td>StatusCode</td>
<td>SHORT</td>
<td>One of the ISilentMonitorEvent status codes in Table 120: ISilentMonitorEvent Status Codes, on page 276.</td>
</tr>
</tbody>
</table>
OnSilentMonitorStopRequestedEvent

For CTI OS Based Silent Monitor

The OnSilentMonitorStopRequestedEvent event is fired to the subscriber to indicate that a silent monitor session was stopped on their behalf. CTI OS server generates this event whenever a remote application calls the StopSilentMonitorRequest method.

Syntax

C++

```cpp
void OnSilentMonitorStopRequestedEvent(Arguments & args);
```

COM

```cpp
HRESULT OnSilentMonitorStopRequestedEvent([in] Arguments* args);
```

VB

```cpp
Session_ OnSilentMonitorStopRequestedEvent(ByVal args as CTIOSCLIENTLIB.IArguments)
```

Parameters

*args*

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID of the agent to be monitored. This message contains either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>DeviceID</td>
<td>STRING</td>
<td>Device ID of the agent to be monitored. This message contains either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where silent monitoring has stopped.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the monitoring application.</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
</tbody>
</table>
For CCM-Based Silent Monitor

When CCM based silent monitor is configured this event tells the monitored application, for example an agent desktop, that it is no longer being monitored. This event in addition to call events for the silent monitor call tells the monitoring application, for example a supervisor desktop, that silent monitor of the agent has ended.

If an error occurs, the Disposition field is set to the error returned in OnControlFailure.

Syntax

C++

```cpp
void OnSilentMonitorStopRequestedEvent(Arguments & args);
```

COM

```com
HRESULT OnSilentMonitorStopRequestedEvent([in] Arguments* args);
```

VB

```vbnet
Session_OnSilentMonitorStopRequestedEvent(ByVal args as CTIOSCLIENTLIB.IArguments)
```

Parameters

args

Arguments array containing the following fields.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SilentMonitorInitiatingAgentUID</td>
<td>STRING</td>
<td>Unique object ID of the agent that initiated silent monitor.</td>
</tr>
<tr>
<td>SilentMonitorInitiatingDeviceID</td>
<td>STRING</td>
<td>ID of the device that initiated silent monitor.</td>
</tr>
<tr>
<td>SilentMonitorTargetAgentUID</td>
<td>STRING</td>
<td>Unique object ID of the silently monitored agent.</td>
</tr>
<tr>
<td>SilentMonitorTargetDeviceID</td>
<td>STRING</td>
<td>ID of the silently monitored device.</td>
</tr>
<tr>
<td>SilentMonitorCallUID</td>
<td>STRING</td>
<td>Unique object ID of the silent monitor call.</td>
</tr>
</tbody>
</table>
If the silent monitor session failed, the event cause carried by the call failed event is stored here. If the silent monitor session was either terminated by the supervisor or the agent's call ended, this field is set to 0.

### OnSilentMonitorStatusReportEvent

The OnSilentMonitorStatusReportEvent event indicates a change in status of a silent monitor session. This event is sent only to the monitoring application.

#### Syntax

**C++**

```cpp
void OnSilentMonitorStatusReportEvent (Arguments & args);
```

**COM**

```cpp
HRESULT OnSilentMonitorStatusReportEvent ([in] Arguments* args);
```

**VB**

```vb
Session_OnSilentMonitorStatusReportEvent (ByVal args as CTIOSCLIENTLIB.IArguments)
```

#### Parameters

arg
Arguments array containing the following fields.

#### Table 119: OnSilentMonitorStatusReportEvent Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored.</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
<tr>
<td>StatusCode</td>
<td>SHORT</td>
<td>One of the ISilentMonitorEvent status codes in Table 120: ISilentMonitorEvent Status Codes, on page 276.</td>
</tr>
<tr>
<td>OriginatingServerID</td>
<td>STRING</td>
<td>TCP/IP Address:Port of the CTI OS server from which the request originated.</td>
</tr>
<tr>
<td>OriginatingClientID</td>
<td>STRING</td>
<td>Client Identification of the monitoring application.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TargetCILClientID</td>
<td>STRING</td>
<td>CIL Client ID of the monitoring application.</td>
</tr>
</tbody>
</table>

**Table 120: ISilentMonitorEvent Status Codes**

<table>
<thead>
<tr>
<th>enum Value</th>
<th>Numeric Value (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Codes</td>
<td></td>
</tr>
<tr>
<td>eSMStatusUnknown</td>
<td>-1</td>
</tr>
<tr>
<td>eSMStatusOK</td>
<td>0</td>
</tr>
<tr>
<td>eSMStatusFailed</td>
<td>0x00000001</td>
</tr>
<tr>
<td>eSMStatusComError</td>
<td>0x00000002</td>
</tr>
<tr>
<td>eSMStatusMonitorStarted</td>
<td>0x00000003</td>
</tr>
<tr>
<td>eSMStatusMonitorStopped</td>
<td>0x00000004</td>
</tr>
<tr>
<td>eSMStatusHeartbeatTimeout</td>
<td>0x00000005</td>
</tr>
<tr>
<td>eSMStatusOutOfMemory</td>
<td>0x00000006</td>
</tr>
<tr>
<td>eSMStatusPortUnavailable</td>
<td>0x00000007</td>
</tr>
<tr>
<td>eSMStatusIncorrectStateForThisAction</td>
<td>0x00000008</td>
</tr>
<tr>
<td>eSMStatusResourceError</td>
<td>0x00000009</td>
</tr>
<tr>
<td>eSMStatusRejectedBadParameter</td>
<td>0x0000000A</td>
</tr>
<tr>
<td>eSMStatusWinsockError</td>
<td>0x0000000B</td>
</tr>
<tr>
<td>eSMStatusMediaTerminationNotPresent</td>
<td>0x0000000C</td>
</tr>
<tr>
<td>eSMStatusIPPhoneInformationNotAvailable</td>
<td>0x0000000D</td>
</tr>
<tr>
<td>eSMStatusMissingParameter</td>
<td>0x0000000E</td>
</tr>
<tr>
<td>eSMStatusSessionNotFound</td>
<td>0x0000000F</td>
</tr>
<tr>
<td>eSMStatusSessionAlreadyExists</td>
<td>0x00000010</td>
</tr>
<tr>
<td>eSMStatusDisconnected</td>
<td>0x00000011</td>
</tr>
<tr>
<td>eSMStatusInvalidStateForAction</td>
<td>0x00000012</td>
</tr>
<tr>
<td>eSMStatusInProgress</td>
<td>0x00000013</td>
</tr>
<tr>
<td>eSMStatusMaxSessionsExceeded</td>
<td>0x00000014</td>
</tr>
<tr>
<td>eSMStatusCCMSilentMonitor</td>
<td>0x00000015</td>
</tr>
</tbody>
</table>
### enum Value  
**Silent Monitor Session Codes**

<table>
<thead>
<tr>
<th>Numeric Value (Hex)</th>
<th>enum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10000000e</td>
<td>eSMStatusSessionTerminatedAbnormally</td>
</tr>
<tr>
<td>0x10000001e</td>
<td>eSMStatusRejectedAlreadyInSession</td>
</tr>
<tr>
<td>0x10000002e</td>
<td>eSMStatusRejectedWinPcapNotPresent</td>
</tr>
<tr>
<td>0x10000003e</td>
<td>eSMStatusWinPcapError</td>
</tr>
<tr>
<td>0x10000004e</td>
<td>eSMStatusMediaUnknownCodec</td>
</tr>
<tr>
<td>0x10000005e</td>
<td>eSMStatusIncorrectSessionMode</td>
</tr>
<tr>
<td>0x10000006e</td>
<td>eSMStatusPeerSilentMonitorNotEnabled</td>
</tr>
<tr>
<td>0x10000007e</td>
<td>eSMStatusSilentMonitorNotEnabled</td>
</tr>
<tr>
<td>0x10000008e</td>
<td>eSMStatusNoResponseFromPeer</td>
</tr>
<tr>
<td>0x10000009e</td>
<td>eSMStatusPeerLoggedOut</td>
</tr>
<tr>
<td>0x1000000Ae</td>
<td>eSMStatusSessionTerminatedByMonitoredClient</td>
</tr>
<tr>
<td>0x1000000Be</td>
<td>eSMStatusSessionTerminatedByMonitoringClient</td>
</tr>
<tr>
<td>0x1000000Ce</td>
<td>eSMStatusNoRTPPacketsReceivedFormIPPhone</td>
</tr>
<tr>
<td>0x1000000De</td>
<td>eSMStatusSessionConnectionToDelegateLost</td>
</tr>
<tr>
<td>0x20000000e</td>
<td>eSMStatusMTError</td>
</tr>
</tbody>
</table>

### Voice Capture-Specific Codes

<table>
<thead>
<tr>
<th>Numeric Value (Hex)</th>
<th>enum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x30000000e</td>
<td>eSMStatusWPNoPacketsReceived</td>
</tr>
<tr>
<td>0x30000001e</td>
<td>eSMStatusWPFailedToOpenDevice</td>
</tr>
<tr>
<td>0x30000002e</td>
<td>eSMStatusWPFailedToSetFilterExp</td>
</tr>
<tr>
<td>0x30000003e</td>
<td>eSMStatusWPErrortInFilterExp</td>
</tr>
</tbody>
</table>

---

**OnStopSilentMonitorConf**

This `OnStopSilentMonitorConf` event is sent to the monitoring application to indicate that the CTI OS server has processed a `StopSilentMonitorRequest`.

**Syntax**

**C++**

```cpp
void OnStopSilentMonitorConf (Arguments & args);
```
COM

HRESULT OnStopSilentMonitorConf ([in] Arguments* args);

VB

Session_OnStopSilentMonitorConf (ByVal args as CTIOSCLIENTLIB.IArguments)

Parameters

args

Arguments array containing the following fields.

Table 121: OnStopSilentMonitorConf Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored.</td>
</tr>
<tr>
<td>AgentID</td>
<td>STRING</td>
<td>Agent ID of the agent to be monitored. This message contains either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>DeviceID</td>
<td>STRING</td>
<td>Device ID of the agent to be monitored. This message contains either AgentID or DeviceID, but not both.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the silent monitor start was requested.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the monitoring application.</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port of the monitoring application.</td>
</tr>
<tr>
<td>SMSSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Heartbeat interval for the silent monitor session.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Timeout for no activity.</td>
</tr>
<tr>
<td>OriginatingServerID</td>
<td>STRING</td>
<td>TCP/IP Address:Port of the CTIOS server from which the request originated.</td>
</tr>
<tr>
<td>OriginatingClientID</td>
<td>STRING</td>
<td>Client Identification of the monitoring application.</td>
</tr>
</tbody>
</table>
OnRTPStreamTimedoutEvent

The OnRTPStreamTimedoutEvent event is sent to the monitored application to report that no RTP voice packets were received from the monitored IP Phone.

Syntax

**C++**
```cpp
void OnRTPStreamTimedoutEvent (Arguments & args);
```

**COM**
```cpp
HRESULT OnRTPStreamTimedoutEvent ([in] Arguments* args);
```

**VB**
```vb
Session_ OnRTPStreamTimedoutEvent (ByVal args as CTIOSCLIENTLIB.IArguments)
```

Parameters

args
Arguments array containing the following fields.

**Table 122: OnRTPStreamTimedoutEvent Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored.</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
</tbody>
</table>
IGenericEvents Interface

The IGenericEvents interface receives Generic events. Unlike other interfaces that have a callback method for each event, the IGenericEvents interface has one method that passes the CtiOs_Enums.EventID code and the Arguments for the event.

OnEvent

Passes the eventID code and arguments for generic events received by the IGenericEvents interface.

Syntax

Java

```java
void OnEvent(int iEventID, Arguments rArgs)
```

.NET

```csharp
void OnEvent(int iEventID, Cisco.CtiOs.Cil.EventPublisher.EventPublisherEventArgs args)
```

Java Adapter Classes

The CTI OS Java CIL contains the same adapter classes as the C++ CIL plus the LogEventsAdapter class. This class provides the default implementation for the message handlers in ILogEvents.

This section lists the methods available in the CTI OS Java CIL for event subscription and unsubscription.

IAllInOne

The following methods subscribe and unsubscribe the CTI OS Session Object for the IAllInOne interface:

Methods

- `int addAllInOneEventListener(IAllInOne allInOneEvents)`
- `int removeAllInOneEventListener(IAllInOne allInOneEvents)`

IAgentEvents

The following methods subscribe and unsubscribe the CTI OS Session Object for the IAgentEvents interface:

Methods

- `int addAgentEventListener(IAgentEvents agentEvents)`

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StatusCode</td>
<td>SHORT</td>
<td>One of the ISilentMonitorEvent status codes in Table 120: ISilentMonitorEvent Status Codes, on page 276.</td>
</tr>
</tbody>
</table>
• int removeAgentEventListener (IAgentEvents agentEvents)

**IButtonEnablementEvents**

The following methods subscribe and unsubscribe the CTI OS Session Object for the IButtonEnablementEvents interface:

**Methods**

• int addButtonEnablementEventListener (IButtonEnablementEvents buttonEvents)
• int removeButtonEnablementEventListener (IButtonEnablementEvents buttonEvents)

**ICallEvents**

The following methods subscribe and unsubscribe the CTI OS Session Object for the ICallEvents interface:

**Methods**

• int addCallEventListener (ICallEvents callEvents)
• int removeCallEventListener (ICallEvents callEvents)

**ISkillGroupEvents**

The following methods subscribe and unsubscribe the CTI OS Session Object for the ISkillGroupEvents interface:

**Methods**

• int addSkillGroupEventListener (ISkillGroupEvents skillGroupEvents)
• int removeSkillGroupEventListener (ISkillGroupEvents skillGroupEvents)

**Events in Java CIL**

To subscribe for events in the Java CIL, use the AddEventListener method. This method has the following syntax:

```java
int AddEventListener (IGenericEvents Listener, int iListID)
```

where Listener is the IGenericEvents object that subscribes for events and iListID is the ID of the subscriber list to add this listener to. Java subscriber list IDs are part of the CtIo杉 Enums.SubscriberList interface; each C++/COM/VB event interface has a corresponding Java subscriber list (for example, C++/COM/VB ISessionEvents corresponds to Java eSessionList). For more information about the CtIo杉 Enums.SubscriberList interface, see the Javadoc file.

The IGenericEvents interface, though it contains the C++/COM/VB events documented in this chapter, does not have a callback method for each event. Instead, the OnEvent method passes the event ID code and arguments for each event. The OnEvent method has the following syntax:
void OnEvent(int iEventID, Arguments rArgs)

where iEventID is the event ID code for the event and rArgs is the arguments for the event. The arguments for each Java event are the same as for the corresponding C++/COM/VB event. For more information about the IGenericEvents interface, see the Javadoc file.

To unsubscribe for events in the Java CIL, use the RemoveEventListener method. This method has the following syntax:

int RemoveEventListener(IGenericEvents Listener, int iListID)

where Listener is the IGenericEvents object that is unsubscribing for events and iListID is the ID of the subscriber list to remove this listener from.

Events in .NET CIL

To subscribe for events in the .NET CIL, use the AddEventListener method. This method has the following syntax:

CilError AddEventListener(IGenericEvents Listener, int iListID)

where Listener is the IGenericEvents object that subscribes for events and iListID is the ID of the subscriber list to add this listener to. Subscriber list IDs for .NET are part of the CtiOs_Enums.SubscriberList interface; each C++/COM/VB event interface has a corresponding .NET subscriber list (for example, C++/COM/VB ISessionEvents corresponds to .NET eSessionList).

The IGenericEvents interface, though it contains the C++/COM/VB events documented in this chapter, does not have a callback method for each event. Instead, the OnEvent method passes the event ID code and arguments for each event. The OnEvent method has the following syntax:

void OnEvent(Object sender, Cisco.CtiOs.Cil.EventPublisher.EventPublisherEventArgs eventArgs)

where, sender is a null object and eventArgs contains the eventID and arguments for the event. The arguments for each .NET event are the same as for the corresponding C++/COM/VB event.

The EventPublisherEventArgs class is a data type that defines the information passed to receivers of the event. This information includes the event ID and an Arguments array containing the arguments for the event. Therefore, event handling code must extract the event arguments from the EventPublisherEventArgs object as shown in the following sample code snippet, which uses the .NET CIL:

Arguments args = eventArgs.rArgs; EventID receivedEvent = (EventID)eventArgs.iEventID;
switch(receivedEvent)
{
    case EventID.eQueryAgentStatisticsConf:
        ProcessQueryConf(args);
        break;
    ...
}

To unsubscribe for events in the .NET CIL, use the RemoveEventListener method.
This method has the following syntax:

```
CilError RemoveEventListener(IGenericEvents Listener, int iListID)
```

where Listener is the IGenericEvents object that is unsubscribing for events and iListID is the ID of the subscriber list from which to remove this listener.

## Event Parameters

### Amount of Nonessential Call Object Parameters

The MinimizeEventArgs registry value controls the amount of nonessential Call object parameters that are sent to the client. When MinimizeEventArgs is set to 1, a minimal set of nonessential Call object parameters are sent to the CTI OS Client. When the MinimizeEventArgs registry value is set to 0, the CTI OS server sends to CTI OS Clients the event parameters listed in Table 6-90.

The MinimizeEventArgs value is located under the following registry key:

`HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems, Inc.\Ctios\<Customer-Instancename>\CTIOS1\Server\CallObject`

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>eCallRetrievedEvent</td>
<td>CTIOS_RETRIEVINGDEVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_RETRIEVINGDEVICEIDFULL</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ENABLEMENTMASK</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ICMENTERPRISEUNIQUEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_UNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_DEVICEUNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLSTATUS*</td>
</tr>
<tr>
<td></td>
<td>CTIOS_FILTERTARGET**</td>
</tr>
<tr>
<td>eCallHeldEvent</td>
<td>CTIOS_HOLDINGDEVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_HOLDINGDEVICEIDFULL</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ENABLEMENTMASK</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ICMENTERPRISEUNIQUEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_UNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_DEVICEUNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_FILTERTARGET**</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLSTATUS*</td>
</tr>
<tr>
<td>Event Name</td>
<td>Parameters</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>eCallConnectionClearedEvent</td>
<td>CTIOS_RELEASINGDEVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_RELEASINGDEVICEIDFULL</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ENABLEMENTMASK</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ICMENTERPRISEUNIQUEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_UNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_DEVICEUNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_FILTERTARGET**</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLSTATUS*</td>
</tr>
<tr>
<td>eCallTransferredEvent</td>
<td>CTIOS_PRIMARYCALLID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SECONDARYCALLID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_TRANSFERRINGDEVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_TRANSFERRINGDEVICEIDFULL</td>
</tr>
<tr>
<td></td>
<td>CTIOS_TRANSFERREDDEVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_TRANSFERREDDEVICEIDFULL</td>
</tr>
<tr>
<td></td>
<td>CTIOS_NUMPARTIES</td>
</tr>
<tr>
<td></td>
<td>ConnectedParty[PartyNumber]</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ITRANSFERCONTROLER</td>
</tr>
<tr>
<td></td>
<td>GenerateCallDataUpdateArgs()***</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ENABLEMENTMASK</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ICMENTERPRISEUNIQUEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS.UniqueObjectID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_DEVICEUNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_FILTERTARGET**</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLSTATUS*</td>
</tr>
</tbody>
</table>
## Event Interfaces and Events

### Amount of Nonessential Call Object Parameters

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>eCallConferencedEvent</td>
<td>CTIOS_PRIMARYCALLID&lt;br&gt;CTIOS_SECONDARYCALLID&lt;br&gt;CTIOS_CONTROLLERDEVICEID&lt;br&gt;CTIOS_CONTROLLERDEVICEIDFULL&lt;br&gt;CTIOS_ADDEDPARTYDEVICEID&lt;br&gt;CTIOS_ADDEDPARTYDEVICEIDFULL&lt;br&gt;CTIOS_PRIMARYDEVICEID&lt;br&gt;CTIOS_PRIMARYDEVICEIDFULL&lt;br&gt;CTIOS_SECONDARYDEVICEID&lt;br&gt;CTIOS_SECONDARYDEVICEIDFULL&lt;br&gt;CTIOS_NUMPARTIES&lt;br&gt;ConnectedParty[PartyNumber]&lt;br&gt;GenerateCallDataUpdateArgs()<em><strong>&lt;br&gt;CTIOS_ENABLEMENTMASK&lt;br&gt;CTIOS_ICMENTERPRISEUNIQUEID&lt;br&gt;CTIOS_UNIQUEOBJECTID&lt;br&gt;CTIOS_DEVICEUNIQUEOBJECTID&lt;br&gt;CTIOS_FILTERTARGET</strong>&lt;br&gt;CTIOS_CALLSTATUS</em></td>
</tr>
<tr>
<td>eCallBeginEvent, eCallDataUpdateEvent</td>
<td>GenerateCallDataUpdateArgs()<em><strong>&lt;br&gt;CTIOS_DEVICEID&lt;br&gt;CTIOS_DIVERTINGDEVICEID&lt;br&gt;CTIOS_DIVERTINGDEVICEIDFULL&lt;br&gt;CTIOS_ENABLEMENTMASK&lt;br&gt;CTIOS_ICMENTERPRISEUNIQUEID&lt;br&gt;CTIOS_UNIQUEOBJECTID&lt;br&gt;CTIOS_DEVICEUNIQUEOBJECTID&lt;br&gt;CTIOS_FILTERTARGET</strong>&lt;br&gt;CTIOS_CALLSTATUS</em></td>
</tr>
<tr>
<td>Event Name</td>
<td>Parameters</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| eCallDivertedEvent | GenerateCallDataUpdateArgs()***  
CTIOS_DIVERTINGDEVICEID  
CTIOS_DIVERTINGDEVICEIDFULL  
CTIOS_ENABLEMENTMASK  
CTIOS_ICMENTERPRISEUNIQUEID  
CTIOS_UNIQUEOBJECTID  
CTIOS_DEVICEUNIQUEOBJECTID  
CTIOS_FILTERTARGET**  
CTIOS_CALLSTATUS* |
| eSnapshotCallConf | Includes all the parameters except for:  
CTIOS_ICMENTERPRISEUNIQUEID  
CTIOS_CALLCONNECTIONCALLID  
CTIOS_CALLCONNECTIONDEVICEIDTYPE  
CTIOS_CALLCONNECTIONDEVICEID  
CTIOS_CALLDEVICECONNECTIONSTATE  
CTIOS_CALLDEVICETYPE |
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>eCallEstablishedEvent</td>
<td>CTIOS_ANSWERINGDEVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ANSWERINGDEVICEIDFULL</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLINGDEVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLINGDEVICEIDFULL</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLEDDEVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLEDDEVICEIDFULL</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SKILLGROUPID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SKILLGROUPNUMBER</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SKILLGROUPPRIORITY</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SERVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SERVICENUMBER</td>
</tr>
<tr>
<td></td>
<td>CTIOS_LINETYPE</td>
</tr>
<tr>
<td></td>
<td>CTIOS_MEASUREDCALLQTIME</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CAMPAIGNID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_QUERYRULEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ENABLEMENTMASK</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ICMENTERPRISEUNIQUEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_UNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_DEVICEUNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_FILTERTARGET**</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLSTATUS*</td>
</tr>
<tr>
<td>Event Name</td>
<td>Parameters</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>eCallDeliveredEvent</td>
<td>CTIOS_ALERTINGDEVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ALERTINGDEVICEIDFULL</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLINGDEVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLEDDEVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLINGDEVICEIDFULL</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLEDDEVICEIDFULL</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SKILLGROUPID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SKILLGROUPNUMBER</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SKILLGROUPPPRIORITY</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SERVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SERVICENUMBER</td>
</tr>
<tr>
<td></td>
<td>CTIOS_LINETYPE</td>
</tr>
<tr>
<td></td>
<td>CTIOS_MEASUREDCALLQTIME</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CAMPAIGNID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_QUERYRULEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ENABLEMENTMASK</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ICMENTERPRISEUNIQUEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_UNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_DEVICEUNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_FILTERTARGET**</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLSTATUS*</td>
</tr>
<tr>
<td>Event Name</td>
<td>Parameters</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>eCallServiceInitiatedEvent,</td>
<td>CTIOS_CALLINGDEVICEIDFULL</td>
</tr>
<tr>
<td>eCallOriginatedEvent,</td>
<td>CTIOS_CALLEDDEVICEIDFULL</td>
</tr>
<tr>
<td>eCallQueuedEvent,</td>
<td>CTIOS_CALLINGDEVICEID</td>
</tr>
<tr>
<td>eCallDequeuedEvent</td>
<td>CTIOS_CALLEDDEVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SKILLGROUPID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SKILLGROUPNUMBER</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SKILLGROUPPRIORITY</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SERVICEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_SERVICENUMBER</td>
</tr>
<tr>
<td></td>
<td>CTIOS_LINETYPE</td>
</tr>
<tr>
<td></td>
<td>CTIOS_MEASUREDCALLQTIME</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CAMPAIGNID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_QUERYRULEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ENABLEMENTMASK</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ICMENTERPRISEUNIQUEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_UNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_DEVICEUNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_FILTERTARGET**</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLSTATUS*</td>
</tr>
<tr>
<td>eControlFailureConf</td>
<td>CTIOSPeriphERALERRORCODE</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ERRORMESSAGE</td>
</tr>
<tr>
<td></td>
<td>CTIOS_FAILURECODE</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ENABLEMENTMASK</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ICMENTERPRISEUNIQUEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_UNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_DEVICEUNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_FILTERTARGET**</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLSTATUS*</td>
</tr>
<tr>
<td>Event Name</td>
<td>Parameters</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>eFailureConf,</td>
<td>CTIOS_ERRORMESSAGE</td>
</tr>
<tr>
<td>eFailureEvent,</td>
<td>CTIOS_FAILURECODE</td>
</tr>
<tr>
<td>eCallFailedEvent</td>
<td>CTIOS_ENABLEMENTMASK</td>
</tr>
<tr>
<td></td>
<td>CTIOS_ICMENTERPRISEUNIQUEID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_UNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_DEVICEUNIQUEOBJECTID</td>
</tr>
<tr>
<td></td>
<td>CTIOS_FILTERTARGET**</td>
</tr>
<tr>
<td></td>
<td>CTIOS_CALLSTATUS*</td>
</tr>
</tbody>
</table>

* If the eCallFailedEvent notification is received, the CTIOS_CALLSTATUS parameter is not added to any more events for the call ID specified in the eCallFailedEvent.

** If there is an agent on the device, then CTIOS_FILTERTARGET is added to all events listed in table 6-90.

*** The GenerateCallDataUpdateArgs() method adds the following parameters to the event:

CTIOS_PERIPHERALID, CTIOS_PERIPHERALTYPE, CTIOS_CALLTYPE, CTIOS_UNIQUEOBJECTID, CTIOS_ROUTERCALLKEYDAY, CTIOS_ROUTERCALLKEYCALLID, CTIOS_CONNECTIONCALLID, CTIOSANI, CTIOS姘EtUSERINF0, CTIOS_DNIS, CTIOS_DIALEDNUMBER, CTIOS_CALLENTEREDDIGITS, CTIOS_SERVICENUMBER, CTIOS_SERVICEID, CTIOS_SKILLGROUPNUMBER, CTIOS_SKILLGROUPPRIORITY, CTIOS_CALLWRAPUPDATA, CTIOS_CAMPAIGNID, CTIOS_QUERYRULEID, CTIOS_CALLVARIABLE1, CTIOS_CALLVARIABLE2, CTIOS_CALLVARIABLE3, CTIOS_CALLVARIABLE4, CTIOS_CALLVARIABLE5, CTIOS_CALLVARIABLE6, CTIOS_CALLVARIABLE7, CTIOS_CALLVARIABLE8, CTIOS_CALLVARIABLE9, CTIOS_CALLVARIABLE10, CTIOS_CUSTOMERPHONENUMBER, CTIOS_CUSTOMERACCOUNTNUMBER, CTIOS_NUMNAMEDVARIABLES, CTIOS_NUMNAMEDARRAYS, CTIOS_ECC, CTIOS_CTICLIENTS
CtiOs Object

All of the interface objects in the CTIOs Client Interface Library support some common features, such as the IsValid and GetValue methods. This chapter describes these common features.

The CCtiOsObject class is the common base class for the objects in the CTIOs client interface library. You implement it as follows:

- In C++: All interface objects (CAgent, CCall, CCtiOsSession, CSkillGroup) derive from the CtiOs object. Thus, all the interface methods described in this chapter are directly available in the C++ objects.
- In COM (VB and C++): The COM objects for Agent, Call, Session, and SkillGroup publish a subset of these methods (as appropriate for the language), and the underlying implementation of the objects uses the C++ CCtiOsObject class to provide these features.
- In Java: All CTIOs interface objects (Agent, Call, Session, and SkillGroup) derive from the CtiOs object. Thus, all the interface methods described in this chapter are directly available in the Java objects.
- In .NET: All interface objects (Agent, Call, Session, and SkillGroup) derive from the CtiOs object. Thus, all the interface methods described in this chapter are directly available on the .NET objects.

The CCtiOsObject provides basic services including:

- Dynamic management of the object properties
- Object lifetime control using a reference counting mechanism
- Run-time class information

Methods

The following table lists the available CCtiOsObject class methods.
### Table 124: CCtiOsObject Class Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DumpProperties</td>
<td>Returns a string listing all of an object's properties' names and values.</td>
</tr>
<tr>
<td>GetAllProperties</td>
<td>Returns all of the object's properties as Args (name/value pairs).</td>
</tr>
<tr>
<td>GetElement</td>
<td>Returns the value of an element.</td>
</tr>
<tr>
<td>GetLast Error</td>
<td>Returns the last error that occurred on the calling thread.</td>
</tr>
<tr>
<td>GetNumProperties</td>
<td>Returns the number of properties of an object.</td>
</tr>
<tr>
<td>GetPropertyName</td>
<td>Returns a property name in a string format.</td>
</tr>
<tr>
<td>GetPropertyType</td>
<td>Returns the data type of the specified property.</td>
</tr>
<tr>
<td>GetValue, GetValueInt, GetValueString, GetValueArray</td>
<td>Returns the value of a specified property.</td>
</tr>
<tr>
<td>IsValid</td>
<td>Checks to see if the property of an object is valid.</td>
</tr>
</tbody>
</table>

### DumpProperties

The DumpProperties method returns all the properties of the object. This method builds a string showing all the properties in the form “key1 = value1; key2 = value2;...”.

**Syntax**

**C++**

```cpp
string DumpProperties ()
```

**COM**

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

**VB**

```vb
DumpProperties() As String
```

**Java**

```java
String DumpProperties()
```

**.NET**

```csharp
System.String DumpProperties()
```

**Parameters**

bstrValue

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

COM: Default HRESULT return value. For more information, see CIL Coding Conventions, on page 19
All Others: The string listing the names of all the object's properties.

GetAllProperties

The GetAllProperties method returns all the object's properties and their values. For the properties that are
calls, agents, or skillgroups, their string UniqueObjectIDs are returned, not the objects themselves. To get the
objects themselves use GetObjectFromObjectID, on page 328, explained in Session Object, on page 307

Syntax

C++

    bool GetAllProperties (Arguments** arguments)

COM

    HRESULT GetAllProperties /*[out]*/ IArguments** arguments, /*[out,retval]*/ VARIANT_BOOL* errorcode)

VB

    GetAllProperties arguments As (CTIOSCLIENTLib.IArguments) As Bool

Java, .NET

    Arguments GetAllProperties()

Parameters

C++, COM, VB: arguments

Output parameter in the form of an Arguments array that has all the property names and values of the object.
errorcode

An output parameter (return parameter in VB) that contains a boolean indicating success or lack thereof.

Return Value

C++, VB: True upon success and false upon failure.

COM: Always returns S_OK. Use the errorcode parameter to determine success or failure of the method call.

.NET, Java: NULL if the value requested is not found or if there is an error. If the method succeeds, it returns
a reference to an Arguments object containing all the properties of the object.

GetElement

Given a property of type Arguments whose name is specified by the key parameter, the GetElement method
returns the Arg at position element of that Arguments array.
**Syntax**

**C++**

```cpp
Arg& GetElement (string& key, int element)
Arg& GetElement (int key, int element)
Arg& GetElement (char* key, int element)
```

**COM**

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

**VB**

```vb
GetElement (key As VARIANT) As CTIOSCLIENTLib.IArg
```

**Java**

```java
Arg GetElement(String key, int element)
Arg GetElement(int key, int element)
```

**.NET**

```csharp
System.Boolean GetElement(System.String key, int element, out arg rArg)
```

**Parameters**

**key**
A key designating the name of the Arguments property whose element you want.

**element**
The integer index of the element to retrieve from the property key.

**pIArg**
An output parameter (return parameter in VB) containing an IArg with the value of the desired element.

**rArg**
An output parameter containing the value of the specified element. This parameter is null if the element is not found.

**Return Value**

An Arg reference containing the value of the desired element.

The C++ and Java versions of this method return NULL if an error occurs, such as the key or element is not found. The .NET version of this method returns true upon success and false upon error.

---

**GetLastError (Java and .NET Only)**

The GetLastError method returns the last error that occurred on the calling thread.

**Syntax**

**Java**

```java
Integer GetLastError()
```

**.NET**

```csharp
System.Boolean GetLastError(out System.Int32 nLastError)
```
Parameters

Java
None.

.NET
nLastError
Output parameter that is a 32-bit signed integer that contains the returned value of the last error.

Returns

Java: An Integer object containing the error, or null if the object is not found or if there is an error.

.NET: The Boolean value true if the value is successfully set; otherwise false.

Remarks

The following example code gets the last error on the current thread and logs the error message. If GetLastError fails, the code writes a warning message to the log file:

```csharp
// First get the last error System.Int32 myLastError;
bool success = GetLastError(out myLastError);
if (!success)
    // log a message indicating that GetLastError failed
else
    // log a message that indicates what the last error was
LOGBYID(Cisco.CtiOs.Cil.TraceLevel.WARN,"GetLastError returned last error" = + myLastError);
```

GetNumProperties

The GetNumProperties method returns the number of properties in the current object.

Syntax

C++

```cpp
int GetNumProperties ()
```

COM

```cpp
HRESULT GetNumProperties (/*[out,retval]*/ int * num
```

VB

```vbnet
GetNumProperties () As Long
```

Java, .NET

```java
int GetNumProperties()
```
GetPropertyName

The GetPropertyName method returns the name of a property in a string format.

Syntax

C++

    string GetPropertyName (int index)

COM

    HRESULT GetPropertyName (/* [in] index, /*[out,retval]*)*/ name)

VB

    GetPropertyName (index As Integer) As String

Java

    string GetPropertyName (int iIndex)

.NET

    virtual System.Boolean GetPropertyName(int iIndex, out System.String name)

Parameters

index

An integer parameter specifying the index number of the requested property.

name

A string output parameter (return value in C++, VB, and Java) containing the property's name.

Return Value

COM: Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19

.NET: Boolean value set to true if the method call succeeds, otherwise false.

All Others: A string that contains the property's name.

GetPropertyType

The GetPropertyType method returns the data type of the specified property.
**Syntax**

**C++**

```cpp
int GetPropertyType (string& key)
int GetPropertyType (int key)
int GetPropertyType (char* key)
```

**COM**

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

**VB**

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

**Java**

```java
int GetPropertyType(string sPropName)
int GetPropertyType(int key)
```

**.NET**

```csharp
virtual ArgDataType GetPropertyType(Enum_CtiOs eKeyId)
virtual ArgDataType GetPropertyType(System.String sPropName)
```

**Parameters**

**key**

Keyword ID name of the property whose type you want. In .NET, eKeyId is the Enum_CtiOs Keyword ID of the property.

**COM:value**

An integer pointer to the value of the type.

**Return Value**

COM: Default HRESULT return value. For more information, see [CIL Coding Conventions, on page 19](#)

Others: An integer indicating the property's type with the following possible values:

<table>
<thead>
<tr>
<th>Argument Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARG_NOTSET</td>
<td>Argument type not determined</td>
</tr>
<tr>
<td>ARG_INT</td>
<td>Signed integer</td>
</tr>
<tr>
<td>ARG_SHORT</td>
<td>2 bytes signed integer</td>
</tr>
<tr>
<td>ARG_BOOL</td>
<td>1 byte integer</td>
</tr>
<tr>
<td>ARG_STRING</td>
<td>C++, COM: STL character string VB String object</td>
</tr>
<tr>
<td>ARG_ARGARRAY</td>
<td>Variable length array of Arg</td>
</tr>
<tr>
<td>ARG_UINT</td>
<td>32 bit unsigned int</td>
</tr>
<tr>
<td>ARG_USHORT</td>
<td>16 bit unsigned short int</td>
</tr>
<tr>
<td>ARG_ARGUMENT</td>
<td>Arguments array</td>
</tr>
</tbody>
</table>
**GetValue**

The GetValue method returns the value of the specified property. Use this method if you do not know the type of the property. Otherwise, use the more specific GetValue methods discussed later in this chapter. When using the COM CIL, do not use this method for properties of type IDispatch*; instead, use GetCurrentCall, GetCurrentAgent, GetAllCalls, GetAllAgents, and GetAllSkillGroups as explained in Session Object, on page 307.

**Syntax**

**C++**

```
Arg& GetValue (string& key)
Arg& GetValue (int key)
Arg& GetValue (char* key)
```

**COM**

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

**VB**

```
GetValue (key As VARIANT) As CTIOSCLIENTLib.IArg
```

**Java**

```
Arg GetValue (String key)
Arg& GetValue (int key)
```

**.NET**

```
virtual System.Boolean GetValue (Enum_CtiOs eKeyID, out Arg obArg)
virtual System.Boolean GetValue (System.String sKey, out Arg obArg)
```

**Parameters**

**key**

The name of the property whose value you want.

**COM:** value

An output value of type Arg** containing the property with the designated name. To get the value of the property, call GetType() on the Arg and then call the specific GetValue method, based on the type.

**.NET:** obArg

Output parameter (return value in C++, VB, and Java) containing the specified property, as described in the explanation of the value parameter.

**Return Value**

**COM:** Default HRESULT return value. For more information, see CIL Coding Conventions, on page 19

**.NET:** Returns true if the value is retrieved, and false if the value is not found.

**Others:** An Arg containing the specified property. To get the value of the property, call GetType() on the Arg and then call the specific GetValue method, based on the type.
GetValueArray

The GetValueArray method returns the Arguments array value of the specified property. Use this method when you know that the property is of Arguments array type, such as ECC call variables.

**Syntax**

**C++**

```
Args GetValueArray (string& key)
Args GetValueArray (enum_Keywords key)
Args GetValue (char * key)
```

**COM**

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

**VB**

```
GetValueArray (key As VARIANT) As CTIOSCLIENTLib.IArguments
```

**Java**

```
Arguments GetValueArray(String key )
```

**.NET**

```
System.Boolean GetValueArray(Enum_CtiOs eKeyID, out Arguments arArguments)
```

**Parameters**

key
The name of the property whose value you want.

value
COM: An output parameter (return value in VB, C++, and Java) containing an IArguments** to the returned value of the property.

.NET: An output parameter containing the Arguments array value upon success. Upon failure, this parameter is set to null.

**Return Value**

COM: Default HRESULT return value. For more information, see CIL Coding Conventions, on page 19.

.NET: Returns true if the value is retrieved. Returns false if the value is not found.

Others: A reference to an Arguments array containing the value of the specified property.

GetValueBoolObj (Java and .NET Only)

The GetValueBool method retrieves the Boolean value of the specified property.

**Syntax**

```
Boolean GetValueBoolObj(int iKey)
Boolean GetValueBoolObj(String sKey)
```
GetValueInt

The GetValueInt method returns the integer value of the specified property. Use this method when you know that the property has an integer type.

Syntax

C++

```cpp
int GetValueInt (string& key)
int GetValueInt (int key)
int GetValueInt (char* key)
```

COM

```c
HRESULT GetValueInt /*[in]*/ VARIANT* key, /*[out,retval]*/ int* value)
```

VB

```vb
GetValueInt (key As VARIANT) As Integer
```

Java

```java
Not implemented, use GetValueIntObj
```

.NET

```csharp
System.Boolean GetValueInt(Enum_CtiOs eKeyID, out System.Int32 nValue)
System.Boolean GetValueInt(System.String sPropname, out System.Int32 nValue)
```

Parameters

C++: key

Depending on the method used, either a string or int that contains the name or ID of the property whose value you want to retrieve.

COM, VB: key

VARIANT containing the ID or name of the property to retrieve.

COM: value

An output parameter (return parameter in VB) containing an integer pointer to the returned value of the property.

.NET: sPropName

The name of the property.

.NET: nValue

Upon success, this output parameter contains the value of the specified property. Upon failure, this parameter is set to null.
.NET: eKeyID
Keyword ID of the property.

Return Value
COM: Default HRESULT return value. For more information, see CIL Coding Conventions, on page 19.
.NET: True if the method succeeds; false if the method fails.

GetValueIntObj (Java Only)

Gets the contained value as an integer.

Syntax
• Integer GetValueIntObj( int iKey)
• Integer GetValueIntObj( String sKey)

Parameters
key
Key ID of the value to be retrieved.

Returns
An Integer object containing a 32 bit signed integer value or null if error.

GetValueShortObj (Java Only)

Retrieves a 16 bit short with the specified key from the array.

Syntax
Short GetValueShortObj( int iKey)
Short GetValueShortObj(short sKey)

Parameters
key
Key ID of the value to be retrieved.

Returns
A Short object containing a 16 bit short value or null if error.

GetValueString

The GetValueString method returns the string value of the property with the specified name. Use this method when you know that the property is of string type. For all CILs and all types, a call to GetValueString for an argument that is of another type, for example Int, returns a string representation of the argument's Int value.
**Syntax**

**C++**

```cpp
    string GetValueString (string& key)
    string GetValueString (int key)
    string GetValueString (char* key)
```

**COM**

```c
    HRESULT GetValueString /*[in]*/ VARIANT* key, /*[out,retval]*/ BSTR* value
```

**VB**

```vb
    GetValueString (key As VARIANT) As String
```

**Java**

```java
    String GetValueString( String key )
    String GetValueString (int key)
```

**.NET**

```c
    System.Boolean GetValueString(Enum_CtiOs eKeyID, out System.String strValue)
    System.Boolean GetValueString(System.String sPropName, out System.String strValue)
```

**Parameters**

**C++, Java: key**

Depending on the method used, either a string or int that contains the name or ID of the property whose value you want to retrieve.

**COM, VB: key**

VARIANT containing the ID or name of the property to retrieve.

**.NET: sPropName**

The name of the property values to retrieve.

**.NET: strValue**

Upon success, this output parameter contains the value of the specified property. Upon failure, this parameter is set to null.

**.NET: eKeyID**

Keyword ID of the property.

In COM, an output parameter (return parameter in VB) containing a BSTR pointer to the returned string value of the property.

**Return Value**

**COM:** Default HRESULT return value. For more information, see [CIL Coding Conventions, on page 19](#).

**.NET:** Boolean value indicating the success or failure of the method call (true, if success; otherwise false).

**Others:** A string containing the value of the specified property.
GetValueUIntObj (Java Only)

Retrieves a 32 bit unsigned integer with the specified key from the array.

Syntax

Long GetValueUIntObj( int key)
Long GetValueUIntObj( String sKey)

Parameters

key

Key ID of the value to be retrieved.

Returns

A Long object containing the 32 bit unsigned integer value or null if error.

GetValueUShortObj (Java Only)

Retrieves a 16 bit unsigned short with the specified key from the array.

Syntax

• Integer GetValueUShortObj( String sKey)

Parameters

key

Key ID of the value to be retrieved.

Returns

An Integer object containing the 16 bit unsigned short value or null if error.

IsValid

The IsValid method tests to see if the object includes the specified property.

Syntax

C++

bool IsValid (string& key)
bool IsValid (char* key)
bool IsValid (int key)

COM

HRESULT IsValid /*[in]*/ VARIANT* key, /*[out retval]*/ VARIANT_BOOL* value

VB

IsValid (key As VARIANT)as Bool
### Java

```java
boolean IsValid(String key)
boolean IsValid(int key)
```

### .NET

```csharp
virtual bool IsValid(Enum_CtiOs eKeyID)
virtual bool IsValid(System.String sKey)
```

### Parameters

<table>
<thead>
<tr>
<th>Environment</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++, Java</td>
<td><code>key</code></td>
<td>A key containing the name or ID of the property whose validity you are testing.</td>
</tr>
<tr>
<td>COM, VB</td>
<td><code>key</code></td>
<td>VARIANT containing the name or ID of the property to retrieve.</td>
</tr>
<tr>
<td>.NET</td>
<td><code>eKeyID</code></td>
<td>The ID of the property whose validity you are testing.</td>
</tr>
<tr>
<td>.NET</td>
<td><code>sKey</code></td>
<td>The name of the property whose validity you are testing.</td>
</tr>
<tr>
<td>COM</td>
<td><code>value</code></td>
<td>An output parameter (return parameter in VB) containing a VARIANT_BOOL pointer indicating whether or not a property with the specified name exists for the object.</td>
</tr>
</tbody>
</table>

### Return Value

COM: Default HRESULT return value. For more information, see [CIL Coding Conventions](#), on page 19.

Others: A boolean indicating whether or not a property with the specified name exists for the object.

### ReportError (Java and .NET only)

The ReportError method sets the value of the LastError property to iErrCode and writes the error to the log as critical.

#### Syntax

```csharp
int ReportError(int iError)
```

#### Parameters

- **Error**
  - The error to report.

#### Returns

The same error code that was passed in through iErrCode.
SetValue (Java and .NET)

The SetValue method adds a new property to the object's property list and gives it the provided value. If the property already exists, it replaces its value with the one provided.

Syntax

```java
boolean SetValue(int iKeyID, Arg rArg)
boolean SetValue(int iKeyID, Arguments rArgs)
boolean SetValue(int iKeyID, boolean bValue)
boolean SetValue(int iKeyID, CtiOsObject rObj)
boolean SetValue(int iKeyID, int iValue)
boolean SetValue(int iKeyID, short nValue)
boolean SetValue(int iKeyID, java.lang.String sValue)
boolean SetValue(java.lang.String sPropName, Arg rArg)
boolean SetValue(java.lang.String sPropName, Arguments rArgs)
boolean SetValue(java.lang.String sPropName, boolean bValue)
boolean SetValue(java.lang.String sPropName, CtiOsObject rObj)
boolean SetValue(java.lang.String sPropName, int iValue)
boolean SetValue(java.lang.String sPropName, short nValue)
boolean SetValue(java.lang.String sPropName, java.lang.String sValue)
boolean SetValueUInt (int key, long value)
boolean SetValueUInt (String key, long value)
boolean SetValueUShort (int key, int value)
boolean SetValueUShort (String key, int value)
```

Parameters

- key
  - The key whose value is to be set.
- value
  - The value to use in setting the element with the designated key.

Returns

- True if successfully added, false if not.

SetValue (C++ COM and VB)

The SetValue method sets the value of the specified Agent property.

Syntax

```cpp
bool SetValue( string& key, string& value)
bool SetValue( string& keyValuePair)
bool SetValue( const char * key, const char * value)
bool SetValue( const char * keyValuePair)
bool SetValue( const char * key, int value)
```
SetValue (C++ COM and VB)

COM

HRESULT SetValue /*[in]*/ VARIANT *key, /*[in]*/ VARIANT *value, /*[out,retval]*/ VARIANT_BOOL * errorcode)

VB

SetValue (key As Variant, value As Variant) As Bool

Parameters

key

An input parameter that contains the name of the property whose value you want to set.

value

An input parameter containing the value to be used in setting the specified property.

keyValuePair

An input parameter containing a string in the format “key=value” where key is a property to set and value is the new value.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values

COM

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

All Others

A boolean indicating the success or failure of the method.

Remarks

You should only use this method when creating a new Agent in preparation for logging in. Therefore, use it to set the AgentID, AgentInstrument, AgentPassword, PeripheralID, and AutoLogin only.
Session Object

The Client Interface Library Session object is used to establish a connection to an active CTI OS server. The main functions of the Session object are:

- Managing the connection to the CTI OS Server  
- Distributing events to the appropriate objects and event subscribers  
- Creating and managing the collections of Agent, Call, and SkillGroup objects  
- Automatically recovering from failures

Typically, an application has a single instance of the Session object, which all other CIL objects use to send and receive events. However, there are no restrictions on the number or types of Session objects one application can employ. It is possible, and sometimes desirable, to establish and manage multiple independent Sessions, for example to use multiple current event streams. If there is more than one Session object monitoring the same Agent or Call, each Session object receives its own events. The order in which events are received is not guaranteed when there are multiple Session objects.

For more information about using the Session object to connect with CTI OS Server, see CTI OS Server Connection, on page 53 in Building Your Custom CTI Application, on page 33.

The Session object creates new Call, Agent, and SkillGroup objects upon receipt of an event for that object if the targeted object does not already exist. The Session object maintains collections of all Agents, Calls, SkillGroups, and WaitObjects. Object lifetime is managed by the Session object, and thus it is important that the client application not delete the objects, which would render the object reference invalid and lead to unpredictable results. When the Session is Released, the connection to CTI OS server is dropped. Any remaining Agent, Call, Skill Group, or WaitObjects are released.

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

The following table lists the available Session properties.

## Note

The data type listed for each keyword is the standardized data type discussed in the section CTI OS CIL data types. For more information about the appropriate language specific types for these keywords, see Table 5: CTI OS CIL Data Type, on page 20.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCMBasedSilentMonitor</td>
<td>INT</td>
<td>If this value is present and set to true, supervisor applications initiate silent monitor using the Agent.SuperviseCall() method. Agent applications do not need to do anything. If this value is not present, or set to false, supervisor and agent applications need to invoke silent monitor using the SilentMonitorManager object. This property only applies to the COM CIL.</td>
</tr>
<tr>
<td>ConnectedSince</td>
<td>INT</td>
<td>Time of day in milliseconds when connected.</td>
</tr>
<tr>
<td>ConnectionMode</td>
<td>INT</td>
<td>eAgentConnection (1), eMonitorConnection (2), or eNotConnected (0).</td>
</tr>
<tr>
<td>CticsA</td>
<td>STRING</td>
<td>Name or TCP/IP address passed as CTI OS server A.</td>
</tr>
<tr>
<td>CticsB</td>
<td>STRING</td>
<td>Name or TCP/IP address passed as CTI OS server B</td>
</tr>
<tr>
<td>CurrentAgent</td>
<td>object reference</td>
<td>Returns reference to current Agent object set by the SetAgent method. Object reference is incremented by one and must be released when no longer used.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CurrentCall</td>
<td>object reference</td>
<td>Valid only if in Agent Connect mode. When there is more than one call, this references the current call. The current call is the call selected. For more information, see CurrentCall in Call Object, on page 389</td>
</tr>
<tr>
<td>CurrentPort</td>
<td>INT</td>
<td>TCP/IP address of the current connected CTIOS server. Can be port A or B.</td>
</tr>
<tr>
<td>CurrentServer</td>
<td>STRING</td>
<td>Name or TCP/IP address of the current connected CTIOS server. The value is blank when the client is not connected to any server. The name may be blank while attempting to reconnect after a lost connection. Otherwise, the name of the server is the name of CTIOS server A or B.</td>
</tr>
<tr>
<td>ForcedDisconnect</td>
<td>INT</td>
<td>The presence of this keyword,</td>
</tr>
<tr>
<td>Heartbeat</td>
<td>INT</td>
<td>Heartbeat time, expressed in seconds. If not set, default heartbeats are configurable on the CTIOS server.</td>
</tr>
<tr>
<td>LastError</td>
<td>INT</td>
<td>Last error code, if any. Otherwise this value is 0.</td>
</tr>
<tr>
<td>MaxHeartbeats</td>
<td>INT</td>
<td>Max heartbeats that can be missed before switching CTIOS servers. Default is 3 missed heartbeats.</td>
</tr>
<tr>
<td>MessageFilter</td>
<td>STRING</td>
<td>The filter that controls the events received by the CIL.</td>
</tr>
<tr>
<td>Object References</td>
<td>ARGUMENTS</td>
<td>Array of object references maintained by the session object. Typically includes Agent References, CallReferences, and SkillGroupReferences. Can also include EmailReferences or ChatReferences.</td>
</tr>
<tr>
<td>PortA</td>
<td>INT</td>
<td>TCP/IP port for ctiosA.</td>
</tr>
<tr>
<td>PortB</td>
<td>INT</td>
<td>TCP/IP port for ctiosB.</td>
</tr>
</tbody>
</table>
### Methods

The following table lists the available session object methods.

#### Table 126: Session object methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddEventListener</td>
<td>Subscribes a .NET IGenericEvents object as a listener on a particular subscriber list.</td>
</tr>
<tr>
<td>AddListener methods</td>
<td>Registers the subscriber for an event listener.</td>
</tr>
<tr>
<td>Connect</td>
<td>Establishes a connection to a CTI OS server.</td>
</tr>
<tr>
<td>CreateSilentMonitorManager</td>
<td>Creates a SilentMonitorManager object instance.</td>
</tr>
<tr>
<td>CreateWaitObject</td>
<td>Creates and returns the pointer to a new CWaitObject.</td>
</tr>
<tr>
<td>DestroySilentMonitorManager</td>
<td>Deletes a SilentMonitorManager object instance.</td>
</tr>
<tr>
<td>DestroyWaitObject</td>
<td>Destroys the specified wait object.</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Closes the connection to the CTI OS server.</td>
</tr>
<tr>
<td>DumpProperties</td>
<td>For more information, see CtiOs Object, on page 291.</td>
</tr>
<tr>
<td>GetAllAgents</td>
<td>Returns a collection of all the agents in the session.</td>
</tr>
<tr>
<td>GetAllCalls</td>
<td>Returns a collection of all the calls in the session.</td>
</tr>
<tr>
<td>GetAllProperties</td>
<td>For more information, see CtiOs Object, on page 291.</td>
</tr>
<tr>
<td>GetAllSkillGroups</td>
<td>Returns a collection of all the skill groups in the session.</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TryingPort</td>
<td>INT</td>
<td>TCP/IP address of the server where a connection is being attempted. Can be port A or B.</td>
</tr>
<tr>
<td>TryingServer</td>
<td>STRING</td>
<td>Contains the name or TCP/IP address of the server where a connection is being attempted. The value is blank if no connection is being attempted (see CurrentServer). The name of the server is the name of CTI OS server A or B.</td>
</tr>
<tr>
<td>TryingSince</td>
<td>INT</td>
<td>Time of day in milliseconds when try began.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>GetCurrentAgent</td>
<td>Returns the currently selected agent.</td>
<td></td>
</tr>
<tr>
<td>GetCurrentCall</td>
<td>Returns the currently selected call.</td>
<td></td>
</tr>
<tr>
<td>GetCurrentSilentMonitor</td>
<td>Returns a pointer to the SilentMonitorManager object instance that is set as the current manager in the CTIOSSession object.</td>
<td></td>
</tr>
<tr>
<td>GetElement</td>
<td>For more information, see CtiOs Object, on page 291</td>
<td></td>
</tr>
<tr>
<td>GetNumProperties</td>
<td>For more information, see CtiOs Object, on page 291</td>
<td></td>
</tr>
<tr>
<td>GetObjectFromObjectID</td>
<td>Returns a Call, Agent, or SkillGroup, given the object's UniqueObjectID.</td>
<td></td>
</tr>
<tr>
<td>GetPropertyName</td>
<td>For more information, see CtiOs Object, on page 291</td>
<td></td>
</tr>
<tr>
<td>GetTypeProperty</td>
<td>For more information, see CtiOs Object, on page 291</td>
<td></td>
</tr>
<tr>
<td>GetValue</td>
<td>For more information, see CtiOs Object, on page 291</td>
<td></td>
</tr>
<tr>
<td>GetValueArray</td>
<td>For more information, see CtiOs Object, on page 291</td>
<td></td>
</tr>
<tr>
<td>GetValueInt</td>
<td>For more information, see CtiOs Object, on page 291</td>
<td></td>
</tr>
<tr>
<td>GetValueString</td>
<td>For more information, see CtiOs Object, on page 291</td>
<td></td>
</tr>
<tr>
<td>IsAgent</td>
<td>Checks the current agent and returns true if the current agent is an agent and not a supervisor.</td>
<td></td>
</tr>
<tr>
<td>IsCCMSilentMonitor</td>
<td>The IsCCMSilentMonitor method determines whether CTIOSS was configured to use CCM based silent monitor. This method is only supported with the C++, Java, COM, and .Net CILs.</td>
<td></td>
</tr>
<tr>
<td>IsSupervisor</td>
<td>Checks the current agent and returns true if the current agent is a supervisor.</td>
<td></td>
</tr>
<tr>
<td>IsValid</td>
<td>For more information, see CtiOs Object, on page 291</td>
<td></td>
</tr>
<tr>
<td>RemoveEventListener</td>
<td>Unregisters the subscriber from an event listener.</td>
<td></td>
</tr>
<tr>
<td>RequestDesktopSettings</td>
<td>Sends a message request to the CTIOSServer to retrieve the desktop settings configured for this site.</td>
<td></td>
</tr>
<tr>
<td>SetAgent</td>
<td>Sets an agent to a session object.</td>
<td></td>
</tr>
<tr>
<td>SetCurrentCall</td>
<td>Associates the current call to a session object.</td>
<td></td>
</tr>
<tr>
<td>SetCurrentSilentMonitor</td>
<td>Sets the SilentMonitorManager object instance specified as the current manager in the CTIOSSession object.</td>
<td></td>
</tr>
</tbody>
</table>
### AddEventListener (Java and .NET Only)

The `AddEventListener` method subscribes an `IGenericEvents` object as a listener on a particular subscriber list.

**Syntax**

**Java**

```java
int AddEventListener(IGenericEvents Listener, int iListID)
```

**.NET**

```csharp
CilError AddEventListener(IGenericEvents Listener, SubscriberList iListID)
```

**Parameters**

- **Listener**
  - The `IGenericEvents` object that is subscribing for events.

- **ListID**
  - The ID of the subscriber list to which the Listener is to be added.

**Returns**

A `CtiOs_Enums.CilError` code indicating success or failure.

### AddListener Methods (C++ Only)

The `AddListener` methods register the subscriber as a listener to the specified set of events.

**Syntax**

```cpp
int AddEventListener(Arguments & rArguments);
int AddSessionEventListener(ISessionEvents * pSessionEvents);
int AddCallEventListener(ICallEvents * pCallEvents);
int AddAgentEventListener(IAgentEvents * pAgentEvents);
int AddSkillGroupEventListener(ISkillGroupEvents * pSkillGroupEvents);
int AddButtonEnablementEventListener(IButtonEnablementEvents * pButtonEvents);
int AddAllInOneEventListener(IAllInOne * pAllInOneEvents);
int AddSilentMonitorEventListener(ISilentMonitorEvents * pSilentMonitorEvents);
int AddSessionEventGenericListener(IGenericEvents * pSessionEvents);
int AddCallEventGenericListener(IGenericEvents * pCallEvents);
```
int AddAgentEventGenericListener(IGenericEvents * pAgentEvents);
int AddSkillGroupEventGenericListener(IGenericEvents * pSkillGroupEvents);
int AddButtonEnablementEventGenericListener(IGenericEvents * pButtonEvents);
int AddAllInOneEventGenericListener(IGenericEvents * pAllInOneEvents);
int AddSilentMonitorEventGenericListener(IGenericEvents * pSilentMonitorEvents);

Remarks
For more information, see Event Subscription in C++, on page 51.

Connect

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

Syntax

C++
   int Connect(Arguments& args)

COM
   HRESULT Connect(IArguments *args, int * errorcode)

VB
   Connect(args As CTIOSCLIENTLib.IArguments) As Long

Java
   int Connect(Arguments args)

.NET
   CilError Connect(Arguments rArgs

Parameters

args
An Arguments array containing the connection parameters listed in the following table:

Table 127: Connect Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtiosA</td>
<td>STRING</td>
<td>Name or TCP/IP address of CTI OS server A. If this value is not provided, the value of Ctios B is used.</td>
</tr>
</tbody>
</table>

Note: If values of neither Ctios A or Ctios B is provided, an error is returned.
Connect

### Keyword

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtiosB</td>
<td>STRING</td>
<td>Name or TCP/IP address of CTIOS server B. If this value is not provided, the value of Ctios A is used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> If values of neither Ctios A or Ctios B is provided, an error is returned.</td>
</tr>
<tr>
<td>PortA (optional)</td>
<td>INT</td>
<td>TCP/IP port for ctiosA, default = 42028.</td>
</tr>
<tr>
<td>PortB (optional)</td>
<td>INT</td>
<td>TCP/IP port for ctiosB, default = 42028.</td>
</tr>
<tr>
<td>Heartbeat (optional)</td>
<td>INT</td>
<td>Heartbeat time, expressed in seconds. If not set, default heartbeats are configurable on CTIOS server.</td>
</tr>
</tbody>
</table>

### errorcode

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

### Return Values

Default CTIOS return values. For more information, see CIL Coding Conventions, on page 19

### Remarks

A successful request results in an OnConnection event.

A failure results in an OnConnectionFailure event. This means that the CIL is in failover. The CIL continues to attempt to connect, alternating between hosts CTIOS_CTIOSA and CTIOS_CTIOSB until connection succeeds, at which point CIL fires OnConnection. If the application wishes to stop failover, it must call Disconnect.

In some cases, additional failure codes and events may occur:

- Connect returns a failure code of -1 if it cannot connect with the initial side of the duplexed CTIOS server pair chosen from the connect parameters. This error code requires no action on the part of the developer as the CIL automatically attempts to connect using the parameters corresponding to the other side of the duplexed pair.

- The CIL retries the connection attempt five times and then does not attempt to reconnect any longer. The final OnConnectionFailure event contains the keyword "FinalAttempt," which informs the client application that the CIL has discontinued its attempts to reconnect.
This behavior only occurs after global settings download has completed. If global settings download is not complete, the CIL continues to retry until successful.

- The Connect method fires an OnCTIOSFailure event to the client indicating the current state of the system. This is in addition to OnConnection or OnConnectionFailure.

The following error codes can occur:

- CIL_OK - no obvious errors, application should wait for an event indicating whether or not Connect succeeded.

- CIL_FAIL - initial attempt to connect with host has failed. CIL fires OnConnectionFailure and goes into failover mode. CIL continues to attempt to connect, alternating between hosts CTIOS_CTIOSA and CTIOS_CTIOSB until connection succeeds at which point CIL fires OnConnection. If application wishes to stop failover, it must call Disconnect.

- E_CTIOS_INVALID_ARGUMENT - a null Arguments parameter was supplied. Connect is aborted. No events are fired.

- E_CTIOS_MISSING_ARGUMENT - indicates that method call provided no value for both CTIOS_CTIOSA or CTIOS_CTIOSB. At least one of these values must be provided. Connect is aborted. No events are fired.

- E_CTIOS_IN_FAILOVER - a previous call to connect failed and CIL is currently in failover attempting to establish a connection. This continues until a connection is established at which point the CIL sends OnConnection indicating that previous call to Connect succeeded. If the developer wishes to call Connect again with different parameters, they must call Disconnect prior to calling Connect again.

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

- E_CTIOS_SESSION_NOT_CONNECTED - unanticipated error. Connect is aborted. No events are fired.

---

**CreateSilentMonitorManager**

The CreateSilentMonitorManager method creates a SilentMonitorManager object instance. To delete the object you must call DestroySilentMonitorManager.

**Syntax**

**C++**

    CSilentMonitorManager * CreateSilentMonitorManager(Arguments & args);

**COM**

    HRESULT CreateSilentMonitorManager ( /*[in]*/ IArguments * args, /*[out,retval]*/ IISilentMonitorManager * * pISilentMonitor);
VB

    CreateSilentMonitorManager (ByVal args as CTIOSCLIENTLIB.IArguments) As 
    CTIOSCLIENTLIB.ISilentMonitorManager

Java

    Not available.

.NET

    Not available.

Parameters

args
Arguments array that contains the parameters listed below. When any of these parameters are specified, the object is constructed with the corresponding property initialized to the specified value. If you want the object initialized with the default values, specify an empty array.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Heartbeat interval for the silent monitor session.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Timeout for no activity.</td>
</tr>
<tr>
<td>MediaTerminationPort</td>
<td>INT</td>
<td>Required only if manager is used in monitoring mode.  TCP/IP port where monitored conversation is sent for playback on system sound card.</td>
</tr>
</tbody>
</table>

Return Value

If successful, a CSilentMonitorManager object is returned. Otherwise, NULL is returned. To identify the specific error, check the value of the LastError Session property (Table 125: Session Properties, on page 308).

Remarks

Supported for use with Unified CCE only.

CreateWaitObject (C++ Java and .NET)

The CreateWaitObject method creates and returns the pointer to a new CWaitObject with the specified event mask.

Syntax

C++

    CWaitObject * CreateWaitObject(Arguments & args);

Java

    WaitObject CreateWaitObject(Arguments rObjParam)
.NET

    CreateWaitObject(Arguments rObjParam)

Parameters

args (C++), rObjParam (Java)

A reference to an Arguments object that contains the list of events the object wait for. The Arguments contain values where the keys are “Event1” through “EventN” and the values are the enumerated event IDs.

Return Values

If successful it returns a pointer to the new Wait object. Otherwise, it returns NULL.

For more information about CWaitObject, see Helper Classes, on page 441

DestroySilentMonitorManager

The DestroySilentMonitorManager method deletes a SilentMonitorManager object instance.

Syntax

C++

    int DestroySilentMonitorManager(CSilentMonitorManager * pSilentMonitor);

COM

    HRESULT DestroySilentMonitorManager (/*[in]*/ ISilentMonitorManager * pSilentMonitor,
    /*[out,retval]*/ int * errorcode);

VB

    DestroySilentMonitorManager (ByVal pSilentMonitor As CTIOSCLIENTLIB.ISilentMonitorManager) As Long

Java

    Not available

.NET

    Not available

Parameters

pSilentMonitor

Valid pointer to a SilentMonitorManager object created via CreateSilentMonitorManager.

errorCode

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19

Remarks

Supported for use with Unified CCE only.
**DestroyWaitObject (C++ Java and .NET)**

The DestroyWaitObject method removes the specified CWaitObject from the Session and decrements its reference count.

**Syntax**

C++

```cpp
void DestroyWaitObject(CWaitObject * pWaitObject)
```

Java

```java
void DestroyWaitObject(WaitObject rWaitObj)
```

.NET

```csharp
DestroyWaitObject(WaitObject rWaitObj)
```

**Parameters**

WaitObject

A pointer to the CWaitObject to be destroyed.

**Return Values**

None.

**Remarks**

For more information about CWaitObject, see Helper Classes, on page 441

---

**DisableSkillGroupStatistics (C++ Java and .NET)**

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

**Syntax**

C++, Java

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

.NET

```csharp
CilError DisableSkillGroupStatistics(Arguments rArgs)
```

**Parameters**

args

This parameter has two required values for PeripheralId and SkillGroupNumber. For more information, see the Remarks section for a code example.

**Return Value**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19
Remarks

C++ code example:

```cpp
Arguments & argsStatBroadcast = Arguments::CreateInstance();
argsStatBroadcast.AddItem(CTIOS_SKILLGROUPNUMBER, intSG);
argsStatBroadcast.AddItem(CTIOS_PERIPHERALID, m_periphID);
m_pSkGrStatSession->DisableSkillGroupStatistics (argsStatBroadcast);
argsStatBroadcast.Release();
```

Disconnect

The Disconnect method disconnects the open connection to the CTI OS server. In Java and .NET, you can use the Disconnect method to interrupt failover.

Syntax

**C++**
```cpp
void Disconnect (Arguments& args);
```

**COM**
```cpp
HRESULT Disconnect (/* [in, optional */ IArguments *args, /*[out]*/ int * errorCode );
```

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

**Java**
```java
int Disconnect(Arguments args)
```

**.NET**
```csharp
CilError Disconnect(Arguments rArgs)
```

Parameters

**args**
An optional Arguments array containing the CTIOS_FORCEDDISCONNECT keyword, which forces a disconnect even if the Session object rejects the disconnect. Add this keyword to the array if the session mode is not set by SetAgent or SetSessionMode at the time of the disconnect.

**errorcode**
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

DumpProperties

For more information about the DumpProperties method, see CtiOs Object, on page 291.
EnableSkillGroupStatistics (C++ Java and .NET)

The EnableSkillGroupStatistics method starts sending real-time statistics to the session object. If the argument array is empty, then statistics for all skill groups are enabled. This is useful when a monitoring application needs to view all statistics without having to enumerate and loop over each statistic to enable it.

Syntax

C++, Java

    EnableSkillGroupStatistics(Arguments & args)

.NET

    CilError EnableSkillGroupStatistics(Arguments rArgs)

Parameters

args

This parameter has three required values for PeripheralId, SkillGroupNumber and SkillGroupPriority. See the Remarks section for a code example.

Important

You must pass a value of "0" for SkillGroupPriority when requesting Skill Group Statistics for Unified CCE agents from a Monitor mode application. Subskill groups are not supported for Unified CCE peripheral types.

Return Value

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19

Remarks

C++ code example:

    Arguments & argsStatBroadcast = Arguments::CreateInstance();
    argsStatBroadcast.AddItem(CTIOS_SKILLGROUPNUMBER, intSG);
    argsStatBroadcast.AddItem(CTIOS_PERIPHERALID, m_periphID);
    argsStatBroadcast.AddItem(CTIOS_SKILLGROUPPRIORITY, priority);
    m_pSkGrStatSession->EnableSkillGroupStatistics ( argsStatBroadcast );
    argsStatBroadcast.Release();

GetAllAgents

The GetAllAgents method returns an array of object IDs. Each object ID is associated with an Agent object stored in the CIL.

The number of object IDs returned from this method depends on the number of agents that the CIL discovered through agent events. For example, a CIL used in an agent desktop application returns one ID, which is the ID of the agent currently logged into the desktop. A supervisor desktop returns the supervisor’s ID as well as IDs for all agents on the supervisor’s team. A monitor mode application filtering all agent events returns IDs for each agent known by the CTI OS Server.
Syntax

C++
Arguments & GetAllAgents()

COM
HRESULT GetAllAgents(/*[out, retval]*/ VARIANT *args)

VB
GetAllAgents (args As VARIANT)

Java
Arguments GetAllAgents()

.NET
Arguments GetAllAgents()

Parameters

args

COM/VB: A pointer to a VARIANT containing a SAFEARRAY of pointers to IAgents.

Return Values

COM/VB
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19

Java/.NET
Returns NULL if the value requested is not found or if there is an error. If the method succeeds, it returns a pointer or a reference to an Arguments array where each member has a string key that is the UniqueObjectID of an agent and a value that is a reference to a CilRefArg that is a pointer to the Agent object.

C++
An empty Arguments array if the value requested is not found or if there is an error. If the method succeeds, it returns a pointer or a reference to an Arguments array where each member has a string key that is the UniqueObjectID of an agent and a value that is a reference to a CilRefArg that is a pointer to the Agent object.

Remarks

The following sample C++ code illustrates how to take the array returned from GetAllAgents() and use it to access the corresponding agents in the CIL’s object cache. The example uses the C++ CIL:

```cpp
Arguments &args = m_pSession->GetAllAgents() ;
// Iterate through all of the CILRefArg objects
// in the Arguments array.
for ( int i = 1 ; i <= args.NumElements() ; i++ )
{
    CILRefArg *pRefArg = NULL ;
    // Retrieve the CILRefArg at each position in the
    // array.
```
if ( args.GetElement(i, (Arg **)&pRefArg) )
{
    if ( pRefArg != NULL )
    {
        // The value method will return a pointer
        // to the agent object referenced by the
        // CILRefArg.
        //
        CAgent *pAgent = (CAgent *)pRefArg->GetValue() ;

        cout << "-- Agent Properties --" << endl ;
        if ( pAgent == NULL )
        {
            cout << "NULL" << endl ;
        }
        else
        {
            cout << pAgent->DumpProperties().c_str() << endl ;
        }
        cout << "--" << endl ;
    }
}

The following sample VB.NET code illustrates how to take the array returned from GetAllAgents() and use it to access the corresponding agents in the CIL’s object cache. The example uses the .NET CIL:

Dim args As Arguments = m_session.GetAllAgents()

' Iterate through all of the CILRefArg objects
' in the Arguments array.
Dim i As Integer
For i = 1 To args.NumElements()
    Dim refArg As CilRefArg

    ' Retrieve the CILRefArg at each position in the
    ' array.
    If (args.GetElement(i, refArg)) Then
        If ((refArg Is Nothing) = False) Then
            ' The value method will return a reference
            ' to the agent object referenced by the
            ' CILRefArg.
            Dim agent As Agent
            refArg.GetValue(agent)
            Console.Out.WriteLine("--")
            If (agent Is Nothing) Then
                Console.Out.WriteLine("Nothing")
            Else
                Console.Out.WriteLine(agent.DumpProperties())
            End If
        End If
    End If
End If
GetAllCalls

The GetAllCalls method returns an array of object IDs. Each object ID is associated with a Call object stored in the CIL.

The number of object IDs returned from this method depends on the number of calls that the CIL discovered through call events. For example, a CIL used in an agent desktop application returns IDs for all calls in which the agent is involved. A supervisor desktop returns IDs for any call in which the supervisor is involved as well as IDs for monitored calls. A monitor mode application filtering all call events returns IDs for each call known by the CTI OS Server.

Syntax

C++
Arguments & GetAllCalls()

COM
HRESULT GetAllCalls(/*[out, retval]*/ VARIANT *args)

VB
GetAllCalls (args As VARIANT)

Java
Arguments GetAllCalls()

.NET
Arguments GetAllCalls()

Parameters

args

COM /VB: A pointer to a VARIANT containing a SAFEARRAY of pointers to ICalls.

Return Values

COM/VB
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Java/.NET
Returns NULL if the value requested is not found or if there is an error. If the method succeeds, it returns a pointer or a reference to an Arguments array where each member has a string key that is the UniqueObjectID of a call and a value that is a reference to a CilRefArg that is a pointer to the Call object.

C++
An empty Arguments array if the value requested is not found or if there is an error. If the method succeeds, it returns a pointer or a reference to an Arguments array where each member has a string key that is the UniqueObjectID of a call and a value that is a reference to a CilRefArg that is a pointer to the Call object.
Remarks

The following sample C++ code illustrates how to take the array returned from GetAllCalls() and use it to access the corresponding calls in the CIL’s object cache. The example uses the C++ CIL:

```cpp
Arguments &args = m_pSession->GetAllCalls();
// Iterate through all of the CILRefArg objects
// in the Arguments array.
//
for ( int i = 1 ; i <= args.NumElements() ; i++ )
{
    CILRefArg *pRefArg = NULL;
    // Retrieve the CILRefArg at each position in the
    // array.
    //
    if ( args.GetElement(i, (Arg **)&pRefArg) )
    {
        if ( pRefArg != NULL )
        {
            // The value method will return a pointer
            // to the agent object referenced by the
            // CILRefArg.
            //
            CCall *pCall = (CCall *)pRefArg->GetValue();
            cout << "-- Call Properties --" << endl;
            if ( pCall == NULL )
            {
                cout << "NULL" << endl;
            }
            else
            {
                cout << pCall->DumpProperties().c_str() << endl;
            }
        }
    }
}
```

The following sample VB.NET code illustrates how to take the array returned from GetAllCalls() and use it to access the corresponding calls in the CIL’s object cache. The example uses the .NET CIL:

```vbnet
Dim args As Arguments = m_session.GetAllCalls()
' Iterate through all of the CILRefArg objects
' in the Arguments array.
'
Dim i As Integer
For i = 1 To args.NumElements()
    Dim refArg As CilRefArg
    ' Retrieve the CILRefArg at each position in the
    ' array.
    If (args.GetElement(i, refArg)) Then
        If ((refArg Is Nothing) = False) Then
            ' The value method will return a reference
            ' to the call object referenced by the
```
' CILRefArg.
Dim aCall As Cisco.CtiOs.Cil.Call
refArg.GetValue(aCall)
Console.Out.WriteLine("--")
Dim str As String
If (aCall Is Nothing) Then
   Console.Out.WriteLine("Nothing")
Else
   Console.Out.WriteLine(aCall.DumpProperties())
End If
Console.Out.WriteLine("--")
End If
Next

## GetAllProperties

For more information about the GetAllProperties method, see CtiOs Object, on page 291.

## GetAllSkillGroups

The GetAllSkillGroups method returns an array of object IDs. Each object ID is associated with a skill group stored in the CIL.

### Syntax

**C++**

```cpp
Arguments & GetAllSkillGroups()
```

**COM**

```
HRESULT GetAllSkillGroups(/*[out, retval]*/ VARIANT *args)
```

**VB**

```vb
GetAllSkillGroups (args As VARIANT)
```

**Java, .NET**

```java
Arguments GetAllSkillGroups()
```

### Parameters

**args**

C++, Java, and .NET: A pointer or a reference to an Arguments array where each member has a string key that is the UniqueObjectID of a skill group and a value that is a reference to a CilRefArg that is a pointer to the skill group object.

COM /VB: A pointer to a VARIANT containing a SAFEARRAY of pointers to ISkillGroups.
Return Values

COM/VB: Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Java/.NET: Returns NULL if the value requested is not found or if there is an error. If the method succeeds, it returns a pointer or a reference to an Arguments array where each member has a string key that is the UniqueObjectID of a skill group and a value that is a reference to a CilRefArg that is a pointer to the skill group object.

C++: An empty Arguments array if the value requested is not found or if there is an error. If the method succeeds, it returns a pointer or a reference to an Arguments array where each member has a string key that is the UniqueObjectID of a skill group and a value that is a reference to a CilRefArg that is a pointer to the skill group object.

GetCurrentAgent

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

Syntax

C++

Agent* GetCurrentAgent()

COM

HRESULT GetCurrentAgent(/*[out, retval]*/ IAgent *agent)

VB

GetCurrentAgent () As CTIOSCLIENTLib.IAgent

Java,.NET

Agent GetCurrentAgent()

Parameters

agent

An output parameter (return value in VB, C++, Java, and .NET) containing a pointer to a pointer to an IAgent that is the currently selected agent.

Return Values

COM: Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Others: A pointer or reference to an agent that is the current agent. This method returns NULL if the value requested is not found or if there is an error.

The C++, Java, and .NET versions of this method return NULL if the value requested is not found or if there is an error.

GetCurrentCall

The GetCurrentCall method returns the call that is currently selected. You can use this method as a way for controls to communicate between each other which call is selected and acted on. Use this method rather than GetValue(“CurrentCall”).
GetCurrentCall

Syntax
C++
CCall * GetCurrentCall()

COM
HRESULT GetCurrentCall(/*[out, retval]*/ ICall ** call)

VB
GetCurrentCall () As CTIOSCLIENTLib.ICall

Java/.NET
Call GetCurrentCall()

Parameters
call
An output parameter (return value in VB, C++, Java, and .NET) containing a pointer to a pointer to an ICall that is the currently selected call.

Return Values
COM: Default CTIOS return values. For more information, see CIL Coding Conventions, on page 19.
Others: A pointer or reference to a Call that is the current call. This method returns NULL if the value requested is not found or if there is an error.
The C++, Java, and .NET versions of this method return NULL if the value requested is not found or if there is an error.

GetCurrentSilentMonitor

The GetCurrentSilentMonitor method returns a pointer to the SilentMonitorManager object instance that is set as the current manager in the session object.

Syntax
C++
CSilentMonitorManager * GetCurrentSilentMonitor();

COM
HRESULT GetCurrentSilentMonitor(/*[out, retval]*/ ISilentMonitorManager ** pSilentMonitor);

VB
GetCurrentSilentMonitor () As CTIOSCLIENTLIB.ISilentMonitorManager

Java,.NET
Not available

Return Values
Pointer to the current Silent Monitor Manager in the session object.
GetElement

For more information about the GetElement method, see CtiOs Object, on page 291.

GetNumProperties

For more information about the GetNumProperties method, see CtiOs Object, on page 291.

GetObjectFromObjectID

Given a string containing the UniqueObjectID of a call, an agent, or a skill group, the GetObjectFromObjectID method returns a pointer to the associated object.

Syntax

C++

bool GetObjectFromObjectID (string& uniqueObjectID, CtiOsObject ** object);

COM

HRESULT GetObjectFromObjectID (/*[in]*/ BSTR uniqueObjectID, /*[out]*/ IDispatch ** object, /*[out, retval]*/ VARIANT_BOOL * errorcode);

VB

GetObjectFromObjectID(uniqueObjectID As String, object as IDispatch) As Boolean

Java

CtiOsObject GetObjectFromObjectID(java.lang.String sUID)

.NET

System.Boolean GetObjectFromObjectID(System.String sUID, out CtiOsObject rObj)

Parameters

COM/C++/VB: uniqueObjectID
A string reference that contains the UniqueObjectID of the requested Call, Agent, or Skillgroup object.

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

COM/C++: object
A pointer to either a CTIOSObject in C++ (which is a CILRefArg) or an IDispatch * pointing to either an ICall, an IAgent, or an ISkillGroup in COM.

.NET: rObj
A pointer to either a CTIOSObject in C++ (which is a CILRefArg) or an IDispatch * pointing to either an ICall, an IAgent, or an ISkillGroup in COM.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.
Return Values

COM: Default HRESULT return value. For more information, see CIL Coding Conventions, on page 19.
C++, VB, .NET: A boolean indicating success or failure of the method.
The Java version of this method returns NULL if the value requested is not found or if there is an error.

Remarks

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

GetPropertyName

For more information about the GetPropertyName method, see CtiOs Object, on page 291.

GetPropertyType

For more information about the GetPropertyType method, see CtiOs Object, on page 291.

GetSystemStatus (Java .NET and C++ Only)

The GetSystemStatus method returns the current system status bitmask.

Syntax

Java/C++

int GetSystemStatus()

.NET

SystemStatus GetSystemStatus()

Parameters

None.

Returns

The current system status bitmask. For more information about the SystemStatus, see OnQueryAgentStateConf, on page 250 in Event Interfaces and Events, on page 167.

GetValue Methods

For more information about the GetValue, GetValueArray, GetValueInt, and GetValueString methods, see CtiOs Object, on page 291.

IsAgent

The IsAgent method determines whether the current agent is an agent rather than a supervisor.
IsAgent

Syntax

C++
    bool IsAgent()

COM
    HRESULT IsAgent (VARIANT_BOOL *bIsAgent)

VB
    IsAgent () As Boolean

Java
    boolean IsAgent()

.NET
    bool IsAgent()

Parameters

bIsAgent

Output parameter (return parameter in VB) that returns true if the current AgentMode connection is for an agent and false if it is for a supervisor.

Return Values

If the current agent is an agent and not a supervisor it returns true, otherwise it returns false.

IsCCMSilentMonitor

The IsCCMSilentMonitor method determines whether CTI OS is configured to use Unified Communication Manager based silent monitor.

Syntax

C++
    bool IsCCMSilentMonitor()

COM
    HRESULT IsCCMSilentMonitor (VARIANT_BOOL * IsCCMSilentMonitor)

Java
    boolean IsCCMSilentMonitor()

.NET
    bool IsCCMSilentMonitor()

Parameters

None.

Return Values

If Unified Communication Manager based silent monitor is configured, this method returns true, otherwise it returns false.
IsSupervisor

The IsSupervisor method checks if the current agent is a supervisor.

Syntax

C++
bool IsSupervisor()

COM
HRESULT IsSupervisor (VARIANT_BOOL * bIsSupervisor)

VB
IsSupervisor () As Boolean

Java
boolean IsSupervisorMode()

.NET
bool IsSupervisorMode()

Parameters

bIsSupervisor
Output parameter (return parameter in VB) that returns true if the current AgentMode connection is for a 
supervisor and false if it is for an agent.

Return Values
If the current agent is a supervisor it returns true, otherwise it returns false.

IsValid

For more information about the IsValid method, see CtiOs Object, on page 291.

RemoveEventListener (Java and .NET)

The RemoveEventListener method unsubscribes a Java IGenericEvents object as a listener from a particular 
subscriber list.

Syntax

int RemoveEventListener(IGenericEvents Listener, int iListID)

Parameters

Listener
The IGenericEvents object that is unsubscribing from events.

ListID
The ID of the subscriber list from which the Listener is to be removed.
Returns

A CtiOsEnums.CilError code indicating success or failure.

RemoveListener Methods (C++ Only)

The RemoveListener methods unregisters the subscriber from a specified event listener.

Syntax

```cpp
int RemoveEventListener(Arguments & rArguments);
int RemoveSessionEventListener(ISessionEvents * pSessionEvents);
int RemoveCallEventListener(ICallEvents * pCallEvents);
int RemoveAgentEventListener(IAgentEvents * pAgentEvents);
int RemoveSkillGroupEventListener(ISkillGroupEvents * pSkillGroupEvents);
int RemoveButtonEnablementEventListener(IButtonEnablementEvents * pButtonEvents);
int RemoveAllInOneEventListener(IAllInOneEvents * pAllInOneEvents);
int RemoveSilentMonitorEventListener(ISilentMonitorEvents * pSilentMonitorEvents);
int RemoveSessionEventGenericListener(IGenericEvents * pSessionEvents);
int RemoveCallEventGenericListener(IGenericEvents * pCallEvents);
int RemoveAgentEventGenericListener(IGenericEvents * pAgentEvents);
int RemoveSkillGroupEventGenericListener(IGenericEvents * pSkillGroupEvents);
int RemoveButtonEnablementEventGenericListener(IButtonEnablementEvents * pButtonEvents);
int RemoveAllInOneEventGenericListener(IAllInOneEvents * pAllInOneEvents);
int RemoveSilentMonitorEventGenericListener(ISilentMonitorEvents * pSilentMonitorEvents);
```

Remarks

For more information, see Event Subscription in C++, on page 51.

RequestDesktopSettings

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

Syntax

**C++**

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

**COM**

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

**VB**

```vbnet
RequestDesktopSettings (args As CTIOSCLIENTLib.IArguments) As Long
```

**Java**

```java
int RequestDesktopSettings(int desktopType)
```
.NET

CilError RequestDesktopSettings(Arguments rArgs)

Parameters

args
C++, COM, VB, and .NET: Input parameter in the form of a pointer or reference to an Arguments array containing one number. This number has a keyword of “DesktopType” and an integer value that is either:

- eAgentDesktop (0)
- eSupervisorDesktop (1)

Java: desktopType
0 for agent
1 for supervisor

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

A successful RequestDesktopSettings request results in an OnGlobalSettingsDownloadConf event. For more information about the OnGlobalSettingsDownloadConf event, see OnFailure Event, on page 173.

SetAgent

The SetAgent method assigns an agent to this Session object. Set the following properties for the Agent object used as a parameter in this method:

- CTOI_AGENTID
- CTOI_PERIPHERALID

To sign on a mobile agent, you must set the following parameters:

- CTOI_AGENTCALLMODE
- CTOI_AGENTREMOTENUMBER

Syntax

C++

int SetAgent(CAgent& agent)

COM

HRESULT SetAgent(/*[in]*/IAgent *agent, /*[out, retval]*/ int * errorcode)
VB

    SetAgent (agent As CTIOSCLIENTLib.IAgent) As Long

Java

    int SetAgent(Agent agentObject)

.NET

    CinError SetAgent(Agent NewAgent)

Parameters

agent
The agent to be assigned to the Session object.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values

If the SetAgent request is successful, it returns a CIL_OK CtiOs_Enums.CilError code and sends an OnSetAgentMode event to the client application.

In CTI OS Release 7.1(1) , the SetAgent request returns the following error codes:

• CIL_FAIL – The request to authenticate failed. The SetAgent request is not sent.

• E_CTIOS_SET_AGENT_SESSION_DISCONNECT_REQUIRED – You attempted to execute SetAgent for a session in monitor mode. The SetAgent request is not sent. To correct, execute the Disconnect method to disconnect the session, then execute the SetAgent method.

• E_CTIOS_AGENT_ALREADY_IN_SESSION – You attempted to assign an agent that has already been assigned to this session. The SetAgent request is not sent.

Note
In the above error cases, the SetAgent request is not sent to the CTI OS server, and the client application does not receive any events in return.

• CIL_OK – The SetAgent request was sent to the CTI OS server.

In Java only, if SetAgent() is called on a session where the current agent is different from the agent that SetAgent is trying to set, the following occurs:

• The CIL automatically does a Disconnect on the current session object to Reset an agent.

• An OnCloseConnection event is received.

• A Connect is then performed.

• An OnConnection event is received, and the new agent is set.

In Java only, if the SetAgent request is unsuccessful it returns one of the following CtiOsEnums.CilError codes:

• E_CTIOS_INVALID_SESSION -- if session is not connected.
• E_CTIOS_PROP_ATTRIBUTES_ACCESS_FAILED -- if unable to get the connection mode property
• E_CTIOS_SET_AGENT_SESSION_DISCONNECT_REQUIRED -- if SetAgent request was during a Monitor Mode session. The client application calls Disconnect first to clean up the connection mode and then calls Connect again.
• E_CTIOS_AGENT_ALREADY_IN_SESSION -- if the agent is already assigned to the session object. The client application calls Disconnect first to clean up the connection mode and then calls Connect again.
• E_CTIOS_ARGUMENT_ALLOCATION_FAILED -- if the application is unable to allocate memory.
• E_CTIOS_PROP_ATTRIBUTES_ACCESS_FAILED -- if an error occurred while accessing agent properties.

SetCurrentCall

The SetCurrentCall method assigns a call as the session's current call.

**Syntax**

**C++**
```cpp
int SetCurrentCall(CCall& call)
```

**COM**
```cpp
HRESULT SetCurrentCall /*{in}*/ICall *call, /*{out, retval}*/ errorcode
```

**VB**
```vb
SetCurrentCall (call As CTIOSCLIENTLib.ICall)
```

**Java**
```java
int SetCurrentCall(Call callObject)
```

**.NET**
```csharp
CilError SetCurrentCall(Call rCall)
```

**Parameters**

call
Call to assign as current call.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

A successful request results in an OnCurrentCallChanged event.
In Java and .NET, if the Call object specified in the call parameter is already the current call, the OnCurrentCallChanged event is not fired to the client application and a E_CTIOS_CALL_ALREADY_CURRENT_IN_SESSION code is returned.

**SetCurrentSilentMonitor**

The `SetCurrentSilentMonitor` method sets the SilentMonitorManager object instance specified as the current manager in the CTI OS session object.

**Syntax**

**C++**

```cpp
int SetCurrentSilentMonitor(CSilentMonitorManager * pSilentMonitor);
```

**COM**

```com
HRESULT SetCurrentSilentMonitor (/*[in]*/ ISilentMonitorManager * pSilentMonitor,
/*[out,retval]*/ int * errorcode);
```

**VB**

```vb
SetCurrentSilentMonitor (ByVal pSilentMonitor As CTIOSCLIENTLIB.ISilentMonitorManager)
As Long
```

**Java**

Not available

**.NET**

Not available

**Parameters**

- `pSilentMonitor`
  - Valid pointer to a SilentMonitorManager object created via CreateSilentMonitorManager.
- `errorcode`
  - An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19

**Remarks**

Supported for use with Unified CCE only.

**SetMessageFilter**

The `SetMessageFilter` method specifies a filter for the CTI OS Server to use to determine which events are sent to a monitor mode client.
Syntax

C++
```cpp
int SetMessageFilter(string filter)
```

COM
```cpp
HRESULT SetMessageFilter(/*{in}*/ BSTR filter, /*{out, retval}*/ int* errorcode)
```

VB
```vbnet
SetMessageFilter (filter As String, retVal As Long)
```

Java
```java
int SetMessageFilter(Arguments messageFilter)
```

.NET
```csharp
CilError SetMessageFilter(Arguments rArgs)
```

Parameters

filter
A string containing the message filter, as explained in the section Notes on Message Filters, on page 338.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

The Session receives an OnMonitorModeEstablished event when the filter is set on the server.

SetSupervisorMonitorMode

You can use the SetSupervisorSilentMonitorMode method to force supervisors into monitored mode. It is used, for example, by the CTI OS Agent desktop to indicate that supervisors logging on to the Agent Desktop can be monitored.

Syntax

C++
```cpp
int SetSupervisorSilentMonitorMode (Arguments & args);
```

COM
```cpp
HRESULT SetSupervisorSilentMonitorMode /*{in}*/ IArguments * args, /*{out,retval}*/ int * errorcode);
```

VB
```vbnet
SetSupervisorSilentMonitorMode (args As CTIOSCLIENTLib.IArguments);
```

Java/.NET
```java
Not available
```
Parameters

args
Arguments array that contains the following parameters.

Table 129: SetSupervisorSilentMonitorMode Arguments Array Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTIOS_SILENTMONITOR FORCEMONITOREDMODE</td>
<td>INT</td>
<td>One of the following values: 1 -- supervisors can be monitored</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 (default) -- supervisors are put in monitoring mode and cannot be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>monitored</td>
</tr>
</tbody>
</table>

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Notes on Message Filters

A message filter is a condition that an event must meet to be sent to the client. It consists of a keyword/value pair, as explained in the following sections.

Note
Two filter mode applicationss are allowed for each CTI OS Server.

Message Filter Syntax

The CTI OS Server's event filter mechanism enables the rapid creation of powerful CTI integration applications. The event filter allows the developer to create a custom event notification stream using a simple filter expression. The filter expression is sent from the Client Interface Library (CIL) to the CTI OS server to request an event stream. The CTI OS server's event filter parses the filter expression, and registers the CIL client for all events that match any of the filter's conditions.

To set a filter expression, the Session object's SetMessageFilter() method is used:

```
' put filter expression in here
Dim sFilterExpression As String

' call SetMessageFilter
m_session.SetMesageFilter sFilterExpression
```

The general form for a filter expression is key=value.
Simple Example

The most basic event filter is for all events for a specific agent. CTI OS uniquely identifies an Agent object by UniqueObjectID (refer to CIL architecture chapter for explanation of the UniqueObjectID). To establish an event stream for a unique agent, the following syntax is used:

```
sFilterExpression = "UniqueObjectID=agent.5000.22866"
```

In this example, the key is the UniqueObjectID, and the value is agent.5000.22866. This is not the same filter expression created when a CIL client connects to CTI OS in Agent Mode. When a CIL client connects to CTI OS in agent mode, the filter includes events for the agent as well as call events for the agent's extension.

General Form of Filter Syntax

The event filter syntax is expressed in the following general form:

```
key1=value1 [,value2, ...] [,key2=valueA [,valueB, ...] ...]
```

In this form, the filter expression must start with a key name (key). Following the key must be an equal sign (=), and at least one value (value1) must be specified. Optionally, additional values (e.g. value2, ...) for the same key can be specified (optional parts of the expression are indicated with square brackets [ ]). This is interpreted as a logical OR among all of the values specified for that key, e.g. if any of those values is found, the condition is satisfied.

For example, a filter expression with one key and multiple values might look like the following:

```
sFilterExpression = "AgentID=22866, 22867, 22868"
```

The interpretation of this filter is to match on any event with AgentID of 22866, 22867, or 22868.

Multiple Filters

You can combine multiple filter expressions (as described above) to create more complex expressions. The general form allows for any number of filters to be concatenated using the semicolon (;), which produces a logical AND effect.

For example, a filter expression with multiple keys, each with multiple values might look like the following:

```
sFilterExpression = "AgentID=22866, 22867, 22868; SkillGroupNumber=20, 21"
```

The interpretation of this filter is to match on any event with AgentID of 22866, 22867, or 22868 and with SkillGroupNumber of 20 or 21.

Filters for Specific Events

One of the most powerful types of event filters for custom applications are filters that work on specific events. An example of such an application is an “all agents” real time display, listing the agent states of all known agents at the call center, using the eAgentStateEvent to receive agent updates. To request specific events, use the MessageID keyword, and the numeric value for the event ID that you wish to receive:
You can also obtain multiple specific events. For example, consider an all calls real-time display application, using eCallBeginEvent and eCallEndEvent to add or remove calls from a view:

```
' register for all eCallBeginEvents, eCallEndEventssFilterExpression = "MessageID = 23, 24"
```

### Events Not Allowed in Filter Expressions

You cannot use the following events in filter expressions:

- ePreLogoutEvent
- ePostLogoutEvent
- eOnConnection
- eOnConnectionClosed
- eOnConnectionFailure
- eOnHeartbeat
- eOnMissingHeartbeat
- eOnCurrentCallChanged
- eOnCurrentAgentReset

- Events that are part of the IMonitoredAgentEvents interface or the IMonitoredCallsInterface. This includes the following events:
  - eOnMonitoredAgentStateChange
  - OnMonitoredAgentInfoEvent
  - OnMonitoredCallDeliveredEvent
  - OnMonitoredCallEstablishedEvent
  - OnMonitoredCallHeldEvent
  - OnMonitoredCallRetrievedEvent
  - OnMonitoredCallClearedEvent
  - OnMonitoredCallConnectionClearedEvent
  - OnMonitoredCallOriginatedEvent
  - OnMonitoredCallFailedEvent
  - OnMonitoredCallConferencedEvent
  - OnMonitoredCallTransferredEvent
  - OnMonitoredCallDivertedEvent
  - OnMonitoredCallServiceInitiatedEvent
• OnMonitoredCallQueuedEvent
• OnMonitoredCallTranslationRouteEvent
• OnMonitoredCallBeginEvent
• OnMonitoredCallEndEvent
• OnMonitoredCallDataUpdateEvent
• OnMonitoredCallReachedNetworkEvent
• OnMonitoredCallDequeuedEvent
• OnMonitoredAgentPercallEvent
• OnMonitoredAgentPercallAbortEvent

To circumvent this restriction, use an equivalent message in the filter expression (for example, OnAgentStateEvent instead of OnMonitoredAgentStateChange) and check in the message handler for the CTIOS.MONITORED parameter to be TRUE.

```c++
void CMyEventSink::OnAgentStateEvent(Arguments & argParams)
{
    if (argParams.IsValid(CTIOS_MONITORED) &&
        argParams.GetValueBoolean(CTIOS_MONITORED))
    {
        // Do process the event
    }
}
```

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FilterTarget=SkillGroupStats</td>
<td>When set, this filter indicates that the CTI OS server forward skill group statistics to the client application, whether or not any agents are logged in.</td>
</tr>
<tr>
<td>HideSilentMonitorCallEvents</td>
<td>This keyword is used to block call events for silent monitor calls in monitor mode applications.</td>
</tr>
</tbody>
</table>

**Skill Group Statistics**

One of the most common applications for a filter mode application is the processing of only skill group statistics. For this purpose, the specialized filter `FilterTarget=SkillGroupStats` is defined. When set, this filter indicates that the CTI OS server forward skill group statistics to the client application, whether or not any agents are logged in.

After the filter is set, the client application needs to invoke the `EnableSkillGroupStatistics(...)` method for each skill group that it is expecting to receive statistics. To stop receiving statistics for a given skill group, the application must invoke the `DisableSkillGroupStatistics` method.
'register to receive skill group
statisticsFilterExpression="FilterTarget=SkillGroupStats"
'call SetMessageFilter
m_session.SetMessageFilter sFilterExpression
'Enable statistics for skills 78, 89 and 90 in peripheral 5004
Private Sub m_Session_OnMonitorModeEstablished(ByVal pArguments As Arguments)
Dim m_Args = new Arguments
'For Skill 78
m_Args.AddItem "SkillGroupNumber", 78
m_Args.AddItem "PeripheralID", 5004
m_session.EnableSkillGroupStatistics m_Args
'For Skill 89
m_Args.Clear
m_Args.AddItem "SkillGroupNumber", 89
m_Args.AddItem "PeripheralID", 5004
m_session.EnableSkillGroupStatistics m_Args
'For Skill 90
m_Args.Clear
m_Args.AddItem "SkillGroupNumber", 90
m_Args.AddItem "PeripheralID", 5004
m_session.EnableSkillGroupStatistics m_Args
'Don't need arguments any more
Set m_Arg = Nothing
End Sub
Private Sub MyObjectClass_OnCleanupApplication()
Dim m_Args = new Arguments
'For Skill 78
m_Args.AddItem "SkillGroupNumber", 78
m_Args.AddItem "PeripheralID", 5004
m_session.DisableSkillGroupStatistics m_Args
'For Skill 89
m_Args.Clear
m_Args.AddItem "SkillGroupNumber", 89
m_Args.AddItem "PeripheralID", 5004
m_session.DisableSkillGroupStatistics m_Args
'For Skill 90
m_Args.Clear
m_Args.AddItem "SkillGroupNumber", 90
m_Args.AddItem "PeripheralID", 5004
m_session.DisableSkillGroupStatistics m_Args
'Don't need arguments any more
Set m_Arg = Nothing
End Sub

CCM-Based Silent Monitor Calls

If a monitor mode application does not wish to receive events for silent monitor calls, it can include the
"HideSilentMonitorCalls" keyword in the filter given to CtiOsSession.SetMessageFilter() to tell CTI OS
Server to hide events for silent monitor calls. For more information about how to use this filter, see All Calls
Sample.NET.
Agent Object

The Agent object provides developers using the CTI OS Client Interface Library with an interface to agent behavior. The Agent object exposes methods to perform all agent behaviors, such as logging in and setting the agent state.

The object stores specific agent information as properties, including the AgentID, AgentPassword, AgentInstrument, AgentExtension, and SkillGroups. When the agent is logged in to an ACD, the Agent object receives updates through AgentStateEvents and Agent Statistics updates.

You can use the Agent object in two different modes:

- In Agent Mode, the application creates an Agent object and informs the Session about the agent using Session.SetAgent().

- In Monitor Mode, the client application sets a message filter, and if the event stream involves events for Agent objects, those objects are dynamically created at the CIL as needed.

Agent Object Properties

The following table lists the agent object properties.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>The data type listed for each keyword is the standardized data type discussed in CTI OS CIL data types in Chapter Three. For more information about the appropriate language specific types for these keywords Table 5: CTI OS CIL Data Type, on page 20.</td>
</tr>
</tbody>
</table>
### Table 130: Agent Object Properties

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent Availability Status</td>
<td>INT</td>
<td>One of the following values: UNKNOWN (-1), NOT AVAILABLE (0), ICM AVAILABLE (1), or APPLICATION AVAILABLE (2).</td>
</tr>
<tr>
<td>Agent CallMode</td>
<td>INT</td>
<td>A value that indicates the agent's call mode. Valid values are call-by-call (3) and nailed-up (4).</td>
</tr>
<tr>
<td>Agent Extension</td>
<td>STRING*</td>
<td>Extension associated by ACD to agent.</td>
</tr>
<tr>
<td>Agent ID</td>
<td>STRING*</td>
<td>Can be set prior to login or after logout.</td>
</tr>
<tr>
<td>Agent Instrument</td>
<td>STRING*</td>
<td>Instrument associated by ACD to agent.</td>
</tr>
<tr>
<td>Agent Remote Number</td>
<td>STRING</td>
<td>The phone number that the agent uses for remote login.</td>
</tr>
<tr>
<td>Agent State</td>
<td>SHORT</td>
<td>One of the values in Table 89: AgentState values, on page 234 representing the current state of the associated agent.</td>
</tr>
<tr>
<td>ClassIdentifier</td>
<td>INT</td>
<td>Identifies the type of this object.</td>
</tr>
<tr>
<td>SilentMonitorCallUID</td>
<td>STRING</td>
<td>The unique object ID of the silent monitor call. This is the call that results from calling SuperviseCall() with the SupervisorAction set to eSupervisorMonitor.</td>
</tr>
</tbody>
</table>

**Note** Only applies to Cisco Unified Communications Manager based silent monitor.
### Agent Statistics

You can access statistics by first using GetValueArray on the Agent object to obtain the “Statistics” Arguments array and then using GetValueInt on the “Statistics” arguments array to obtain the desired value:

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

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

### Table 131: Agent Statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailTime Session</td>
<td>Total time, in seconds, the agent was in the Available state for any skill group.</td>
</tr>
<tr>
<td>LoggedOnTime Session</td>
<td>Total time, in seconds, the agent has been logged in.</td>
</tr>
<tr>
<td>NotReadyTime Session</td>
<td>Total time, in seconds, the agent was in the Not Ready state for all skill groups.</td>
</tr>
<tr>
<td>ICMAvailable TimeSession</td>
<td>Total time, in seconds, the agent was in the Unified ICM Available state.</td>
</tr>
<tr>
<td>RoutableTime Session</td>
<td>Total time, in seconds, the agent was in the Routable state for all skill groups.</td>
</tr>
<tr>
<td>AgentOutCalls Session</td>
<td>Total number of completed outbound ACD calls made by agent.</td>
</tr>
<tr>
<td>AgentOutCalls TalkTimeSession</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCalls Time Session</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCalls Held Session</td>
<td>The total number of completed outbound ACD calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>AgentOutCalls HeldTime Session</td>
<td>Total number of seconds outbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>HandledCalls Session</td>
<td>The number of inbound ACD calls handled by the agent.</td>
</tr>
<tr>
<td>HandledCalls TalkTime Session</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by the agent. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HandledCalls AfterCall TimeSession</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.</td>
</tr>
<tr>
<td>HandledCalls Time Session</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by the agent. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>IncomingCalls Held Session</td>
<td>The total number of completed inbound ACD calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>IncomingCalls HeldTime Session</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>InternalCalls Session</td>
<td>Number of internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCalls TimeSession</td>
<td>Number of seconds spent on internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCalls RcvdSession</td>
<td>Number of internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCalls RcvdTime Session</td>
<td>Number of seconds spent on internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCalls Held Session</td>
<td>The total number of internal calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCalls HeldTime Session</td>
<td>Total number of seconds completed internal calls were placed on hold.</td>
</tr>
<tr>
<td>AutoOutCalls Session</td>
<td>Total number of AutoOut (predictive) calls completed by the agent.</td>
</tr>
<tr>
<td>AutoOutCalls TalkTime Session</td>
<td>Total talk time, in seconds, of AutoOut (predictive) calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCalls Time Session</td>
<td>Total handle time, in seconds, for AutoOut (predictive) calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCalls Held Session</td>
<td>The total number of completed AutoOut (predictive) calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AutoOutCalls HeldTime Session</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold.</td>
</tr>
<tr>
<td>PreviewCalls Session</td>
<td>Total number of outbound Preview calls completed by the agent.</td>
</tr>
<tr>
<td>PreviewCalls TalkTime Session</td>
<td>Total talk time, in seconds, of outbound Preview calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCalls TimeSession</td>
<td>Total handle time, in seconds, outbound Preview calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCalls HeldSession</td>
<td>The total number of completed outbound Preview calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>PreviewCalls HeldTime Session</td>
<td>Total number of seconds outbound Preview calls were placed on hold.</td>
</tr>
<tr>
<td>Reservation Calls Session</td>
<td>Total number of agent reservation calls completed by the agent.</td>
</tr>
<tr>
<td>Reservation CallsTalk TimeSession</td>
<td>Total talk time, in seconds, of agent reservation calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>Reservation CallsTime Session</td>
<td>Total handle time, in seconds, agent reservation calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>Reservation CallsHeld Session</td>
<td>The total number of completed agent reservation calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>Reservation CallsHeld TimeSession</td>
<td>Total number of seconds agent reservation calls were placed on hold.</td>
</tr>
<tr>
<td>BargeInCalls Session</td>
<td>Total number of supervisor call barge-ins completed.</td>
</tr>
<tr>
<td>InterceptCalls Session</td>
<td>Total number of supervisor call intercepts completed.</td>
</tr>
<tr>
<td>MonitorCalls Session</td>
<td>Total number of supervisor call monitors completed.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WhisperCalls Session</td>
<td>Total number of supervisor whisper calls completed.</td>
</tr>
<tr>
<td>EmergencyCallsSession</td>
<td>Total number of emergency calls.</td>
</tr>
<tr>
<td>AvailTimeToday</td>
<td>Total time, in seconds, the agent was in the Available state for any skill group.</td>
</tr>
<tr>
<td>LoggedOnTime Today</td>
<td>Total time, in seconds, the agent has been logged in.</td>
</tr>
<tr>
<td>NotReadyTime Today</td>
<td>Total time, in seconds, the agent was in the Not Ready state for all skill groups.</td>
</tr>
<tr>
<td>ICMAvailable TimeToday</td>
<td>Total time, in seconds, the agent was in the Unified ICM Available state.</td>
</tr>
<tr>
<td>RoutableTime Today</td>
<td>Total time, in seconds, the agent was in the Routable state for all skill groups.</td>
</tr>
<tr>
<td>AgentOutCalls Today</td>
<td>Total number of completed outbound ACD calls made by agent.</td>
</tr>
<tr>
<td>AgentOutCalls TalkTime Today</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCalls Time Today</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCalls HeldToday</td>
<td>The total number of completed outbound ACD calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>AgentOutCalls HeldTime Today</td>
<td>Total number of seconds outbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>HandledCalls Today</td>
<td>The number of inbound ACD calls handled by the agent.</td>
</tr>
<tr>
<td>HandledCalls TalkTime Today</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by the agent. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>HandledCalls AfterCall TimeToday</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HandledCalls Time Today</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by the agent. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>IncomingCalls HeldToday</td>
<td>The total number of completed inbound ACD calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>IncomingCalls HeldTime Today</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold.</td>
</tr>
<tr>
<td>InternalCalls Today</td>
<td>Number of internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCalls Time Today</td>
<td>Number of seconds spent on internal calls initiated by the agent.</td>
</tr>
<tr>
<td>InternalCalls RcvdToday</td>
<td>Number of internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCalls RcvdTime Today</td>
<td>Number of seconds spent on internal calls received by the agent.</td>
</tr>
<tr>
<td>InternalCalls HeldToday</td>
<td>The total number of internal calls the agent placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCalls HeldTime Today</td>
<td>Total number of seconds completed internal calls were placed on hold.</td>
</tr>
<tr>
<td>AutoOutCalls Today</td>
<td>Total number of AutoOut (predictive) calls completed by the agent.</td>
</tr>
<tr>
<td>AutoOutCalls TalkTime Today</td>
<td>Total talk time, in seconds, of AutoOut (predictive) calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCalls TimeToday</td>
<td>Total handle time, in seconds, for AutoOut (predictive) calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCalls HeldToday</td>
<td>The total number of completed AutoOut (predictive) calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>AutoOutCalls HeldTime Today</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PreviewCalls Today</td>
<td>Total number of outbound Preview calls completed by the agent.</td>
</tr>
<tr>
<td>PreviewCalls TalkTimeToday</td>
<td>Total talk time, in seconds, of outbound Preview calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCalls TimeToday</td>
<td>Total handle time, in seconds, outbound Preview calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCalls HeldToday</td>
<td>The total number of completed outbound Preview calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>PreviewCalls HeldTimeToday</td>
<td>Total number of seconds outbound Preview calls were placed on hold.</td>
</tr>
<tr>
<td>Reservation CallsToday</td>
<td>Total number of agent reservation calls completed by the agent.</td>
</tr>
<tr>
<td>Reservation CallsTalkTimeToday</td>
<td>Total talk time, in seconds, of agent reservation calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>Reservation CallsTimeToday</td>
<td>Total handle time, in seconds, agent reservation calls completed by the agent. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>Reservation CallsHeldToday</td>
<td>The total number of completed agent reservation calls the agent has placed on hold at least once.</td>
</tr>
<tr>
<td>Reservation CallsHeldTimeToday</td>
<td>Total number of seconds agent reservation calls were placed on hold.</td>
</tr>
<tr>
<td>BargeInCalls Today</td>
<td>Total number of supervisor call barge-ins completed.</td>
</tr>
<tr>
<td>InterceptCalls Today</td>
<td>Total number of supervisor call intercepts completed.</td>
</tr>
<tr>
<td>MonitorCalls Today</td>
<td>Total number of supervisor call monitors completed.</td>
</tr>
<tr>
<td>WhisperCalls Today</td>
<td>Total number of supervisor whisper calls completed.</td>
</tr>
</tbody>
</table>
### Methods

The following table lists the Agent object methods.

**Table 132: Agent Object Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisableAgentStatistics</td>
<td>Disables agent statistic messages.</td>
</tr>
<tr>
<td>DisableSkillGroupStatistics</td>
<td>Disables skill group statistic messages.</td>
</tr>
<tr>
<td>DumpProperties</td>
<td>For more information, see <a href="#">CtiOs Object, on page 291</a></td>
</tr>
<tr>
<td>EnableAgentStatistics</td>
<td>Enables agent statistic messages.</td>
</tr>
<tr>
<td>EnableSkillGroupStatistics</td>
<td>Enables skill group statistic messages.</td>
</tr>
<tr>
<td>GetAgentState</td>
<td>Returns the current agent state.</td>
</tr>
<tr>
<td>GetAllProperties</td>
<td>For more information, see <a href="#">CtiOs Object, on page 291</a></td>
</tr>
<tr>
<td>GetElement</td>
<td>For more information, see <a href="#">CtiOs Object, on page 291</a></td>
</tr>
<tr>
<td>GetMonitoredAgent</td>
<td>Returns the Agent object that is currently being monitored.</td>
</tr>
<tr>
<td>GetMonitoredCall</td>
<td>Returns the Call object that is currently being monitored.</td>
</tr>
<tr>
<td>GetNumProperties</td>
<td>For more information, see <a href="#">CtiOs Object, on page 291</a></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmergencyCalls Today</td>
<td>Total number of emergency calls.</td>
</tr>
<tr>
<td>AvailTime Session</td>
<td>Total time, in seconds, the agent was in the Available state for any skill group.</td>
</tr>
<tr>
<td>LoggedOnTime Session</td>
<td>Total time, in seconds, the agent has been logged in.</td>
</tr>
<tr>
<td>NotReadyTime Session</td>
<td>Total time, in seconds, the agent was in the Not Ready state for all skill groups.</td>
</tr>
<tr>
<td>ICMAvailable TimeSession</td>
<td>Total time, in seconds, the agent was in the Unified ICM Available state.</td>
</tr>
<tr>
<td>RoutableTime Session</td>
<td>Total time, in seconds, the agent was in the Routable state for all skill groups.</td>
</tr>
<tr>
<td>AgentOutCalls Session</td>
<td>Total number of completed outbound ACD calls made by agent.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetPropertyName</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetPropertyType</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetSkillGroups</td>
<td>Returns an array of SkillGroups objects</td>
</tr>
<tr>
<td>GetValue</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetValueArray</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetValueInt</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetValueString</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>IsAgent</td>
<td>Checks the current mode and returns true if agent mode.</td>
</tr>
<tr>
<td>IsSupervisor</td>
<td>Checks the current mode and returns true if supervisor mode.</td>
</tr>
<tr>
<td>IsValid</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>Login</td>
<td>Logs an agent in to the ACD.</td>
</tr>
<tr>
<td>Logout</td>
<td>Logs an agent out of the ACD.</td>
</tr>
<tr>
<td>MakeCall</td>
<td>Initiates a call to a device or agent.</td>
</tr>
<tr>
<td>MakeEmergencyCall</td>
<td>Lets an agent make an emergency call to the supervisor.</td>
</tr>
<tr>
<td>QueryAgentState</td>
<td>Gets the current agent state from CTI Server and retrieves it.</td>
</tr>
<tr>
<td>ReportBadCallLine</td>
<td>Informs the CTI OS Server of a bad line.</td>
</tr>
<tr>
<td>RequestAgentTeamList</td>
<td>Retrieves the current agent team list.</td>
</tr>
<tr>
<td>RequestSupervisorAssist</td>
<td>Allows the agent to call an available supervisor for assistance.</td>
</tr>
<tr>
<td>SendChatMessage</td>
<td>Send asynchronous messages between CTI clients.</td>
</tr>
<tr>
<td>SetAgentGreetingAction</td>
<td>Sets the value of the Agent Greeting Action to enable or disable Agent Greeting for the logged in agent.</td>
</tr>
<tr>
<td>SetAgentState</td>
<td>Requests a new agent state.</td>
</tr>
<tr>
<td>SetValue</td>
<td>Sets the value of the property whose name is specified.</td>
</tr>
<tr>
<td>StartMonitoringAgent</td>
<td>Enables monitoring of a specified agent.</td>
</tr>
<tr>
<td>StartMonitoringAgentTeam</td>
<td>Enables monitoring of a specified agent team.</td>
</tr>
</tbody>
</table>
Method | Description
--- | ---
StartMonitoringAllAgentTeams | Enables monitoring of all agent teams.
StartMonitoringCall | Enables monitoring of a specified Call object.
StopMonitoringAgent | Disables monitoring of a specified agent.
StopMonitoringAgentTeam | Disables monitoring of a specified agent team.
StopMonitoringAllAgentTeams | Disables monitoring of all agent teams.
SuperviseCall | Enables monitoring a call of an agent on your team.

Arguments Parameters

The following rules apply to the optional_args and reserved_args parameters in Call Object methods:

- In VB, you can ignore these parameters altogether. For example, you can treat the line:

```vbnet
Answer([reserved_args As IArguments]) As Long
```

as follows:

```vbnet
Answer()
```

- To ignore these parameters in COM you must send a NULL, as shown:

```csharp
Answer (NULL)
```

DisableAgentStatistics

The DisableAgentStatistics method is sent by an agent to request that real-time statistics stop being sent to that agent.

Syntax

C++

```c
int DisableAgentStatistics (Arguments& reserved_args)
```

COM

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

VB

```vbnet
DisableAgentStatistics (reserved_args As CTIOSCLIENTLib.IArguments) As Long
```

Java

```java
int DisableAgentStatistics (Arguments reservedargs)
```

.NET

```csharp
CilError DisableAgentStatistics(Arguments args)
```
Parameters

.NET: args
Not currently used, reserved for future use.

All Others: reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Value
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

DisableSkillGroupStatistics

The DisableSkillGroupStatistics method is sent by an agent to request that real-time statistics stop being sent to that agent.

Syntax

C++
int DisableSkillGroupStatistics (Arguments& optional_args)

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

VB
DisableSkillGroupStatistics (optional_args As CTIOSCLIENTLib.IArguments) As Long

Java
int DisableSkillGroupStatistics (Arguments optional_args

.NET
CilError DisableSkillGroupStatistics(Arguments args)

Parameters

optional_args
An optional input parameter containing a pointer or a reference to an Arguments array containing a member that is a nested Arguments array with the keyword SkillGroupNumbers. Within this array, for each skill group to be disabled, specify a string key of an integer starting with 1 and an integer value for skill group number and specify a string key of an integer and integer value for skill group priority. If the parameter is NULL or missing, statistics are disabled for all skill groups to which the agent belongs.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.
EnableAgentStatistics

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

Syntax

C++
int EnableAgentStatistics(Arguments& reserved_args)

COM
HRESULT EnableAgentStatistics /*[in]*/ IArguments* reserved_args, /* [out, retval]*/
int * errorcode)

VB
EnableAgentStatistics (reserved_args As CTIOSCLIENTLib.IArguments) As Long

Java
int EnableAgentStatistics(Arguments args)

.NET
CilError EnableAgentStatistics(Arguments args)

Parameters

reserved_args
Not currently used, reserved for future use.

Java/.NET:args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Value

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

The CTI OS server sends agent statistics in an OnAgentStatistics event. For more information about the PollingIntervalSec and PollForAgentStatsAtEndCall registry settings and how these settings affect the refresh rate of agent statistics, see OnAgentStatistics, on page 235 in Chapter 6, Event Interfaces and Events, on page 167

EnableSkillGroupStatistics

The EnableSkillGroupStatistics method is sent by an agent to request that real-time statistics be sent to that agent. If the Argument array is empty, then statistics for all skill groups are sent. This is useful when a
monitoring application needs to view all statistics without having to enumerate and loop over each statistic to enable it.

**Syntax**

**C++**

```cpp
int EnableSkillGroupStatistics (Arguments& optional_args)
```

**COM**

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

**VB**

```vb
EnableSkillGroupStatistics (optional_args As CTIOSCLIENTLib.IArguments) As Long
```

**Java**

```java
Java:int EnableSkillGroupStatistics(Arguments optional_args)
```

**.NET**

```c
CilError EnableSkillGroupStatistics(Arguments args)
```

**Parameters**

**optional_args**

An optional input parameter containing a pointer or a reference to an Arguments array containing a member that is a nested Arguments array with the keyword SkillGroupNumbers. Within this array, each member has a string key of an integer starting with 1 and an integer value that is a skill group number to be enabled and a string key of an integer and integer value that is a skill group priority to be enabled. If the parameter is NULL or missing, statistics are enabled for all skill groups to which the agent belongs.

**args**

Refer to the description for optional_args above.

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Value**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

The CTI OS server sends SkillGroup statistics in the OnSkillGroupStatisticsUpdated event of the SkillGroup object.

**GetAgentState**

The GetAgentState method returns the current state of the agent.
Syntax
C++
   enumCTIOS_AgentState GetAgentState()

COM
   HRESULT GetAgentState (/*[in]*/ long *state)

VB
   GetAgentState () As Long

Java
   int GetAgentState()

.NET
   AgentState GetAgentState()

Parameters
state

Output parameter (return parameter in VB) containing the current agent state in the form of one of the values in Table 89: AgentState values, on page 234.

Return Value
For C++, VB, Java, and .NET, this method returns the current state of the agent.

GetAllProperties

For more information about the GetAllProperties method, see CtiOs Object, on page 291.

GetElement

For more information about the GetElement method, see CtiOs Object, on page 291.

GetMonitoredAgent

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

Syntax
C++
   CAgent* GetMonitoredAgent()

COM
   HRESULT GetMonitoredAgent (/*[out, retval]*/IAgent **agent)

VB
   GetMonitoredAgent () As CTIOSCLIENTLib.IAgent

Java
   Agent GetMonitoredAgent()
.NET
   Agent GetMonitoredAgent()

Parameters
agent
Output parameter (return parameter in VB) that contains a pointer to a pointer to an Agent object containing
the currently monitored agent.

Return Value
This method returns the current monitored agent. The C++, Java, and .NET versions return null if no agent
is currently being monitored.

Remarks
Supported for use with Unified CCE only.

GetMonitoredCall
The GetMonitoredCall method returns the Call object that is currently being monitored.

Syntax
C++
   CCall* GetMonitoredCall()

COM
   HRESULT GetMonitoredCall(/*[out, retval]*/ICall **call)

VB
   GetMonitoredCall () As CTIOSCLIENTLib.ICall

Java
   Call GetMonitoredCall()

.NET
   Call GetMonitoredCall()

Parameters
call
Output parameter (return parameter in VB) that contains a pointer to a pointer to a Call object containing the
currently monitored call.

Return Value
This method returns the current monitored call. The C++, Java, and .NET versions return null if no call is
currently being monitored.

Remarks
Supported for use with Unified CCE only.
GetNumProperties

For more information about the GetNumProperties method, see CtiOs Object, on page 291.

GetPropertyType

For more information about the GetNumProperties method, see CtiOs Object, on page 291.

GetSkillGroups

If skillgroupstats is enabled, the GetSkillGroups method allows a client to retrieve a list that contains references to all the skill group objects to which the agent belongs. To retrieve skill groups enable skill group statistics, and turn off agent event minimization by setting its value to 0 on the CTI OS server in the registry key, for example:

HKLM\SOFTWARE\Cisco Systems,Inc.\Ctios\<Customer-Instancename>\CTIOS\Server\Agent\MinimizeAgentStateEvents

The skill group information is available on the agent state change event if the minimization is turned off. The following code example shows how to access the skill group properties of the Agent object:

```vbnet
Log m_Agent.DumpProperties
Dim i As Integer
For i = 1 To 20
If m_Agent.IsValid("SkillGroup[" & i & "]") Then
    Set argskills = m_Agent.GetValueArray("SkillGroup[" & i & "]")
    Log "SkillGroup[" & i & "]: & argskills.DumpArgs
Else
    Log "SkillGroup[" & i & "] args doesnt exist"
End If
Next i
```

Syntax

C++

Arguments & GetSkillGroups();

COM

HRESULT GetSkillGroups (/*[out retval]*/ VARIANT * pVariantArgs);

VB

GetSkillGroups () As Variant

Java

Arguments GetSkillGroups()

.NET

Arguments GetSkillGroups()
**Parameters**

None.

**Return Value**

This method returns -1 if skillgroupstats is not enabled.

**C++**

In C++ the GetSkillGroups method returns an Arguments array containing references to CSkillGroup objects. Each element in the returned Arguments array consists of a key/value pair, in which the element key is the Unique ObjectId of the skill group object and the value is a reference to a CILRefArg object instance that contains the actual reference to a CSkillGroup object. To retrieve a reference to a skill group object, you need to do something similar to what is shown in the following code example.

```cpp
Arguments & arSkills = m_Agent->GetSkillGroups();
if(Arguments::IsValidReference(arSkills)){
    for(int nI = 1; nI <= arSkills.NumElements(); nI ++){
        string strUOID = arSkills.GetKey(nI);
        CilRefArg & pRefArg = (CilRefArg &) arSkills.GetValue(strUOID);
        if(Arg::IsValidReference(*pRefArg)){
            CSkillGroup * pSkill = pRefArg->GetValue();
            pRefArg->Release();
            cout << "Skill Object (" << strUOID << ") ;
            cout << " Skill Group Number: " << ;
            pSkill->GetValueInt(CTIOS_SKILLGROUPNUMBER);
        }
    }
}
```

**COM**

In COM the GetSkillGroups method returns a pointer to a variant that encapsulates a Safearray where each element is a pointer to an ISkillGroup object.

To retrieve references to skill group objects, you need to do something similar to what is shown in the following code example.

```cpp
HRESULT hr = S_OK;VARIANT varSkills;
VariantInit(&varSkills)
hr = m_Agent->GetSkillGroups(&varSkills);
if(SUCCEEDED(hr)){
    if(varSkills.vt == (VT_ARRAY | VT_DISPATCH )){
        long lNumElements = 0;
        SafeArrayGetUBound(varSkills.parray,1,&lNumElements);
        for(long nI = 0; nI < lNumElements; nI ++){
            ISkillGroup * pSkill= NULL;
            hr=SafeArrayGetElement(varSkills.parray,&nI,&pSkill);
            if(SUCCEEDED(hr)){
                int nSkillGrpNumber = 0;
                VARIANT vPropKey;
                VariantInit(&vPropKey);
                pSkill->GetValueInt(CTIOS_SKILLGROUPNUMBER);
            }
        }
    }
}
```
VB

In VB, the GetSkillGroups method returns a variant array where each element is a reference to a CTIOSClientLib.SkillGroup object.

To retrieve references to skill group objects you need to do something similar to what is shown in the following code example:

```vb
Dim obSkill As CTIOSClientLib.SkillGroup
Dim arSkills As Variant
Dim lNumElements As Long

arSkills = m_Agent.GetSkillGroups()
lNumElements = UBound(arSkills,1)
For nI = 0 to lNumElements
    Set obSkill = arSkills(nI)
    Print "SkillGroup" & obSkill.GetValueString(CStr("UniqueObjectId")) & "Skill Group Number: " & obSkill.GetValueInt(CStr("SkillGroupNumber"))
Next
End For
```

GetValue Methods

For more information about the GetValue, GetValueInt, GetValueArray, and GetValueString methods, see CtiOs Object, on page 291.

IsAgent

The IsAgent method determines whether the AgentMode connection is for an agent rather than a supervisor.

Syntax

C++

```cpp
bool IsAgent()
```

COM

```c
HRESULT IsAgent (VARIANT_BOOL *bIsAgent)
```

VB

```vb
IsAgent () As Boolean
```

Java

```java
boolean IsAgent()
```
.NET

    bool IsAgent()

Parameters

IsAgent

Output parameter (return parameter in VB) that returns true if the current AgentMode connection is for an agent and false if it is for a supervisor.

Return Value

Returns true if the current AgentMode connection is for an agent and false if the connection is for a supervisor.

IsSupervisor

The IsSupervisor method determines whether the AgentMode connection is for a supervisor.

Syntax

C++

    bool IsSupervisor()

COM

    HRESULT IsSupervisor (VARIANT_BOOL * bIsSupervisor)

VB

    IsSupervisor () As Boolean

Java

    boolean IsSupervisorMode()

.NET

    bool IsSupervisor()

Parameters

bIsSupervisor

Output parameter (return parameter in VB) that returns true if the current AgentMode connection is for a supervisor and false if it is for an agent.

Return Values

If the current session is for a supervisor, this method returns true. Otherwise the method returns false.

Login

The Login method performs a login to the ACD (if supported). Generally, the minimum parameters required to log in to an ACD are AgentID and AgentInstrument. Often, based on customer configuration, the minimum requirements include an ACD password (AgentPassword). Some switches require PositionID in place of (or in addition to) AgentInstrument. Optional arguments include Extension or AgentWorkMode.

To sign on a mobile agent, you must set the following parameters:
- CTIOS_REMOTELOGIN set to true
- CTIOS_AGENTREMOTENUMBER
- CTIOS_AGENTCALLMODE

**Example**
```
rArgs.SetValue(Enum_CtiOs.CTIOS_REMOTELOGIN, "true");
rArgs.SetValue(Enum_CtiOs.CTIOS_AGENTREMOTENUMBER,"777989");
rArgs.SetValue(Enum_CtiOs.CTIOS_AGENTCALLMODE, 4);
```

**Syntax**

**C++**
```
virtual int Login(Arguments & args);
```

**COM**
```
HRESULT Login ( /*[in]*/ IArguments * pVariantArgs, /*[out]*/ int * errorCode );
```

**VB**
```
Login (args As CTIOSCLIENTLib.IArguments) As Long
```

**Java**
```
int Login(Arguments args)
```

**.NET**
```
CilError Login(Arguments args)
```

**Input Parameters**

`args`

Arguments array that contains the login parameters that are listed in the following table:

**Table 133: Login Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentID (required)**</td>
<td>STRING*</td>
<td>The agent's login ID.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING*</td>
<td>The agent's instrument number.</td>
</tr>
<tr>
<td>LoginName (required)**</td>
<td>STRING</td>
<td>The agent's login name.</td>
</tr>
<tr>
<td>AgentExtension</td>
<td>STRING*</td>
<td>The agent's teleset extension. Optional if AgentInstrument is provided.</td>
</tr>
<tr>
<td>AgentPassword (optional)</td>
<td>STRING*</td>
<td>The agent's password.</td>
</tr>
</tbody>
</table>
### Login

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentWorkMode (optional)</td>
<td>INT</td>
<td>A value representing the desired work mode of the agent. Used by Avaya Communications Manager (ACM) ECS with default value of ManualIn.</td>
</tr>
<tr>
<td>NumSkillGroups (optional)</td>
<td>INT</td>
<td>The number of Skill Groups that the agent is currently associated with, up to a maximum of 20.</td>
</tr>
<tr>
<td>PeripheralID (optional)</td>
<td>INT</td>
<td>The Unified ICM Peripheral ID of the ACD the agent is attached to.</td>
</tr>
<tr>
<td>SkillGroupNumber (optional)</td>
<td>INT</td>
<td>The number of an agent skill group associated with the agent.</td>
</tr>
<tr>
<td>SkillGroupPriority (optional)</td>
<td>INT</td>
<td>The priority of an agent skill group associated with the agent.</td>
</tr>
<tr>
<td>Agent CallMode</td>
<td>INT</td>
<td>A value that indicates the agent's call mode. Valid values are call-by-call (3) and nailed-up (4).</td>
</tr>
<tr>
<td>AgentRemote Number</td>
<td>STRING</td>
<td>The phone number that the agent uses for remote login.</td>
</tr>
<tr>
<td>RemoteLogin</td>
<td>INT</td>
<td>A value that indicates the agent is configured for remote login as a remote agent.</td>
</tr>
</tbody>
</table>

*The CTI OS server imposes no restriction on the maximum length of this string. However, such restrictions are generally imposed by your switch/ACD and Cisco CTI Server. Consult the documentation for the switch/ACD or CTI Server for information on length restrictions for this string.

**Either AgentID or LoginName is required.

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

If the Login request is successful, it returns a CIL_OK CtiOs_Enums.CilError code In addition, the requesting client can expect an AgentStateChange event if the request is successful with an Arguments member with keyword “AgentState” and value of the agent's current state. (For more information about possible values, see GetAgentState.)
If the Login request is unsuccessful, the client receives an OnControlFailureConf event and the request returns one of the following CtiOs Enums.CilErrorCode codes:

- E_CTIOS_INVALID_SESSION -- either the agent is not associated with the session or the session is not connected.
- E_CTIOS_INVALID_ARGUMENT -- null or invalid arguments were provided.
- E_CTIOS_LOGIN_INCONSISTENT_ARGUMENTS -- Login request argument values for AgentId and/or PeripheralID do not match the values that were set by SetAgent() prior to the Login request.

Logout

The Logout method logs the agent out of the ACD. If the ACD configuration requires or supports other parameters, you can pass these in as logout parameters. Examples are logout reason codes (supported on ACM ECS, Unified CCE).

Syntax

C++

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

COM

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

VB

```vbnet
Logout (args As CTIOSCLIENTLib.IArguments) As Long
```

Java

```java
int Logout(Arguments args)
```

.NET

```csharp
CilError Logout(Arguments args)
```

Input Parameters

args

Input parameter in the form of an Arguments array that contains the Logout parameters that are listed in the following table:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentPassword (optional)</td>
<td>STRING*</td>
<td>The agent's password.</td>
</tr>
<tr>
<td>NumSkillGroups (optional)</td>
<td>INT</td>
<td>The number of Skill Groups that the agent is currently associated with, up to a maximum of 20.</td>
</tr>
</tbody>
</table>
### MakeCall

The MakeCall method initiates a call to a device or agent. The simplest form of the request requires only a DialedNumber.

**Syntax**

**C++**

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

**COM**

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

**VB**

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

---

**Note**

You can select and make the call against the skillgroup. Do not set the value if the default skillgroup is desired.

---

*The CTI OS server imposes no restriction on the maximum length of this string. However, such restrictions are generally imposed by your switch/ACD and Cisco CTI Server. Consult the documentation for the switch/ACD or CTI Server for information on length restrictions for this string.*

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

If the request is successful, the client receives an OnAgentStateChange event with an Arguments member with keyword “AgentState” and value eLogout. If it is unsuccessful, the client receives an OnControlFailureConf event. The client also receives an OnPreLogout event before the OnAgentStateChange event, and an OnPostLogout event afterwards.
Java

    int MakeCall(Arguments args)

.NET

    CilError MakeCall(Arguments args)

Input Parameters

args

Input parameter in the form of an Arguments array that contains the MakeCall parameters that are listed in the following table:

**Table 135: MakeCall Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DialedNumber (required)</td>
<td>STRING, maximum length 40</td>
<td>The number to be dialed to establish the new call.</td>
</tr>
<tr>
<td>PeripheralID (optional)</td>
<td>INT</td>
<td>The Unified ICM Peripheral ID of the ACD the agent is attached to.</td>
</tr>
<tr>
<td>AgentInstrument (optional)</td>
<td>STRING*</td>
<td>The agent's instrument number.</td>
</tr>
<tr>
<td>CallPlacementType (optional)</td>
<td>STRING, maximum length 40</td>
<td>A value specifying how the call is to be placed is identified in Table 136: Call Placement Types, on page 370.</td>
</tr>
<tr>
<td>CallMannerType (optional)</td>
<td>INT</td>
<td>A value specifying additional call processing options is identified in Table 137: Unified CM Type, on page 370.</td>
</tr>
<tr>
<td>AlertRings (optional)</td>
<td>INT</td>
<td>The maximum amount of time that the call's destination remains alerting, specified as an approximate number of rings. A zero value indicates that the peripheral default (typically 10 rings) is used.</td>
</tr>
<tr>
<td>CallOption (optional)</td>
<td>INT</td>
<td>A value from Table 138: Peripheral-Specific Call Options, on page 371 specifying additional peripheral-specific call options.</td>
</tr>
<tr>
<td>FacilityType (optional)</td>
<td>INT</td>
<td>A value from Table 139: Facility Types, on page 371 indicating the type of facility to use.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AnsweringMachine (optional)</td>
<td>INT</td>
<td>A value from Table 140: Answering Machine Actions, on page 371 specifying the action to be taken if the call is answered by an answering machine.</td>
</tr>
<tr>
<td>Priority (optional)</td>
<td>BOOL</td>
<td>This field should be set to TRUE if the call receives priority handling.</td>
</tr>
<tr>
<td>PostRoute (optional)</td>
<td>BOOL</td>
<td>When this field is set to TRUE, the Post-Routing capabilities of the Unified ICM are used to determine the new call destination.</td>
</tr>
<tr>
<td>UserToUserInfo (optional)</td>
<td>STRING, maximum length 40</td>
<td>The ISDN user-to-user information.</td>
</tr>
<tr>
<td>CallVariable1 (optional)</td>
<td>STRING, maximum length 40</td>
<td>Call variable data set in the new call in place of the corresponding data in the active call.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>CallVariable10 (optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECC (optional)</td>
<td>ARGUMENTS</td>
<td>ECC data that is set in the new call in place of the corresponding data in the active call.</td>
</tr>
<tr>
<td>CallWrapupData (optional)</td>
<td>STRING, maximum length 40</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>FacilityCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>Set the FacilityType to 1 for trunk groups and enter the trunk access code in the FacilityCode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set the FacilityType to 2 for skill groups and enter the SkillGroupID in the FacilityCode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set the FacilityType to 0 for unspecified and enter a split extension or other data needed to access the chosen facility in the FacilityCode.</td>
</tr>
<tr>
<td>AuthorizationCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>An authorization code needed to access the resources required to initiate the call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> The AuthorizationCode parameter is not used and is not supported.</td>
</tr>
</tbody>
</table>
### Description

**Type Keyword**

- **AccountCode (optional)**: STRING, maximum length 40. A cost-accounting or client number used by the peripheral for charge-back purposes.
- **SkillGroupNumber**: INT. This keyword is not functional in MakeCall. Instead, to specify the skill group in MakeCall, enter a FacilityType of 2 and enter the SkillGroupID in the FacilityCode.

### Table 136: Call Placement Types

<table>
<thead>
<tr>
<th>CallPlacementType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPT_UNSPECIFIED</td>
<td>Use default call placement.</td>
<td>0</td>
</tr>
<tr>
<td>CPT_LINE_CALL</td>
<td>An inside line call.</td>
<td>1</td>
</tr>
<tr>
<td>CPT_OUTBOUND</td>
<td>An outbound call.</td>
<td>2</td>
</tr>
<tr>
<td>CPT_OUTBOUND_NO_ACCESS_CODE</td>
<td>An outbound call that does not require an access code.</td>
<td>3</td>
</tr>
<tr>
<td>CPT_DIRECT_POSITION</td>
<td>A call placed directly to a specific position.</td>
<td>4</td>
</tr>
<tr>
<td>CPT_DIRECT_AGENT</td>
<td>A call placed directly to a specific agent.</td>
<td>5</td>
</tr>
<tr>
<td>CPT_SUPERVISOR_ASSIST</td>
<td>A call placed to a supervisor for call handling assistance.</td>
<td>6</td>
</tr>
</tbody>
</table>

*The CTI OS server imposes no restriction on the maximum length of this string. However, such restrictions are generally imposed by your switch/ACD and Cisco CTI Server. Consult the documentation for the switch/ACD or CTI Server for information on length restrictions for this string.*

### Table 137: Unified CM Type

<table>
<thead>
<tr>
<th>CallMannerType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMT_UNSPECIFIED</td>
<td>Use default call manner.</td>
<td>0</td>
</tr>
<tr>
<td>CMT_POLITE</td>
<td>Attempt the call only if the originating device is idle.</td>
<td>1</td>
</tr>
<tr>
<td>CMT_BELLIGERENT</td>
<td>Always attempt the call, disconnecting any currently active call.</td>
<td>2</td>
</tr>
<tr>
<td>CMT_SEMI_POLITE</td>
<td>Attempt the call only if the originating device is idle or is receiving dial tone.</td>
<td>3</td>
</tr>
</tbody>
</table>
### Table 138: Peripheral-Specific Call Options

<table>
<thead>
<tr>
<th>CallOption</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPT_UNSPECIFIED</td>
<td>No call options specified, use defaults.</td>
<td>0</td>
</tr>
<tr>
<td>COPT_CALLINGAGENT_ONLINE</td>
<td>Attempt the call only if the calling agent is “online” (available to interact with the destination party).</td>
<td>1</td>
</tr>
<tr>
<td>COPT_CALLINGAGENT_RESERVED</td>
<td>Attempt the call only if ACDNR on the calling agent's set is activated.</td>
<td>2</td>
</tr>
<tr>
<td>COPT_CALLINGAGENT_NOT_RESERVED</td>
<td>Attempt the call only if ACDNR on the calling agent's set is not activated.</td>
<td>3</td>
</tr>
<tr>
<td>COPT_CALLINGAGENT_BUZZ_BASE</td>
<td>Applies a buzz to the base of the telephone set as the call is initiated.</td>
<td>4</td>
</tr>
<tr>
<td>COPT_CALLINGAGENT_BEEP_HSET</td>
<td>Applies a tone to the agent headset as the call is initiated.</td>
<td>5</td>
</tr>
<tr>
<td>COPT_SERVICE_CIRCUIT_ON</td>
<td>Applies a call classifier to the call (ACM ECS).</td>
<td>6</td>
</tr>
</tbody>
</table>

### Table 139: Facility Types

<table>
<thead>
<tr>
<th>FacilityType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT_UNSPECIFIED</td>
<td>Use default facility type.</td>
<td>0</td>
</tr>
<tr>
<td>FT_TRUNK_GROUP</td>
<td>Facility is a trunk group.</td>
<td>1</td>
</tr>
<tr>
<td>FT_SKILL_GROUP</td>
<td>Facility is a skill group or split.</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 140: Answering Machine Actions

<table>
<thead>
<tr>
<th>AnsweringMachine</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM_UNSPECIFIED</td>
<td>Use default behavior.</td>
<td>0</td>
</tr>
<tr>
<td>AM_CONNECT</td>
<td>Connect call to agent when call is answered by an answering machine.</td>
<td>1</td>
</tr>
<tr>
<td>AM_DISCONNECT</td>
<td>Disconnect call when call is answered by an answering machine.</td>
<td>2</td>
</tr>
<tr>
<td>AM_NONE</td>
<td>Do not use answering machine detection.</td>
<td>3</td>
</tr>
<tr>
<td>AnsweringMachine</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>AM_NONE_NO_MODEM</td>
<td>Do not use answering machine detection, but disconnect call if answered by a modem.</td>
<td>4</td>
</tr>
<tr>
<td>AM_CONNECT_NO_MODEM</td>
<td>Connect call when call is answered by an answering machine, disconnect call if answered by a modem.</td>
<td>5</td>
</tr>
</tbody>
</table>

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Value**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

If the request is successful, the client receives one or more of the following call related events:

- OnCallBegin
- OnCallDelivered
- OnServiceInitiated
- OnCallOriginated
- OnCallReachedNetwork

If the request is unsuccessful, the client receives an OnControlFailureConf event.

**MakeEmergencyCall**

The MakeEmergencyCall method makes an emergency call to the Agent's supervisor.

**Syntax**

**C++**

```cpp
int MakeEmergencyCall ()
int MakeEmergencyCall (Arguments& reserved_args)
```

**COM**

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

**VB**

```vbnet
MakeEmergencyCall () As Long
MakeEmergencyCall (reserved_args As CTIOSCLIENTLib.IArguments) As Long
```

**Java**

```java
int MakeEmergencyCall (Arguments args)
```
.NET

CilError MakeEmergencyCall(Arguments args)

Parameters

reserved_args
Not currently used, reserved for future use.

args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Value

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19

Remarks

The MakeEmergencyCall request is very similar to the RequestSupervisorAssist request in the following two ways:

• Both requests place a call from the requesting agent to a supervisor and are routed employing the same script. A typical script might attempt to route the call to the primary supervisor first (if logged in and in available state) and, failing that, to route the call to a skillgroup that all supervisors belong to.

• You can configure Unified ICM Agent Desk Settings to make both call requests via a single step conference or consult call. If the consult method is chosen, the agent can complete the established consult call as a transfer or conference.

These two requests have the following important differences:

• Only Emergency calls can be recorded, if so configured in the Unified ICM Agent Desk Settings.

• The calls are reported separately in Unified ICM reporting.

Having these two separate requests gives a site some flexibility in implementing supervisor help for its agents, instructing agents to use one for certain cases and the other for different situations. In general, use the MakeEmergencyCall method for higher priority calls than calls made with the RequestSupervisorAssist method. For example, you can train agents to click the Emergency button if the customer has more than $1,000,000 in an account, and otherwise to click the Supervisor Assist button. The Supervisor can differentiate the agent's request by noting the CallType.

The MakeEmergencyCall request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, QueryAgentState

The QueryAgentState method lets a client retrieve the current state of the agent.
**Syntax**

**C++**

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

**COM**

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

**VB**

```vb
QueryAgentState (ByVal args as CTIOSCLIENTLIB.IArguments) As Long
```

**Java**

```java
int QueryAgentState (Arguments args)
```

**.NET**

```c
CilError QueryAgentState(Arguments args)
```

**Input Parameters**

*args*

Arguments array that contains the parameters listed in the following table.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent ID</td>
<td>STRING</td>
<td>Agent's login ID.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING</td>
<td>Agent's instrument number.</td>
</tr>
</tbody>
</table>

**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

If the request is successful, the client receives an OnQueryAgentStateConf event. If it is unsuccessful, the client receives an OnControlFailureConf event.

**ReportBadCallLine**

The ReportBadCallLine method informs the CTI OS server of the poor quality of the agent's line. A note of this is recorded in the database.

**Syntax**

**C++**

```cpp
int ReportBadCallLine ()
int ReportBadCallLine (Arguments& reserved_args)
```

**COM**

```c
HRESULT ReportBadCallLine (/*[in, optional]*/ IArguments reserved_args, /* [out, retval]*/
int * errorcode)
```
VB
    ReportBadCallLine () As Long

Java
    int ReportBadCallLine (Arguments args)

.NET
    CilError ReportBadCallLine(Arguments args)

Parameters
reserved_args
Not currently used, reserved for future use.
Java/.NET: args
Not currently used, reserved for future use.
errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19

RequestAgentTeamList

The RequestAgentTeamList method is called by a supervisor to make a request to the CTI OS server for a list of agents in the supervisor's team.

Syntax
C++
    int RequestAgentTeamList ()
    int RequestAgentTeamList (Arguments & reserved_args)

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

VB
    RequestAgentTeamList () As Long

Java
    int RequestAgentTeamList ()
    int RequestAgentTeamList (Arguments args)

.NET
    CilError RequestAgentTeamList(Arguments args)

Parameters
reserved_args
Not currently used, reserved for future use.
Java/.NET: args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Value
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks
Supported for use with Unified CCE only.
If this request is successful, the CTI OS server sends a separate OnNewAgentTeamMember event for each agent in the supervisor's team. If this request is unsuccessful, the client receives an OnControlFailureConf event.

RequestSupervisorAssist
The RequestSupervisorAssist method allows the agent to call an available supervisor for assistance.

Syntax
C++

    virtual int RequestSupervisorAssist();
    int RequestSupervisorAssist (Arguments& reserved_args)

COM

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

VB

    RequestSupervisorAssist () As Long

Java

    int RequestSupervisorAssist(Arguments args)

.NET

    CilError RequestSupervisorAssist(Arguments args)

Parameters
reserved_args
Not currently used, reserved for future use.

Java/.NET: args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.
**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

Supported for use with Unified CCE only. For more information, see MakeEmergencyCall, on page 372.

**SendChatMessage**

The SendChatMessage method sends asynchronous chat-like messages between CTI OS clients. Users can specify a distribution of one or more clients, and attach a text message.

**Syntax**

**C++**

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

**COM**

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

**VB**

```vbnet
SendChatMessage (args As CTIOSCLIENTLib.IArguments) As Long)
```

**Java**

```java
int SendChatMessage(Arguments args)
```

**.NET**

```csharp
CilError SendChatMessage(Arguments args)
```

**Parameters**

**args**

Input parameter in the form of an Arguments array that contains one or more of the SendChatMessage parameters listed in the following table.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution (required)</td>
<td>STRING</td>
<td>Currently the only supported value is “agent”.</td>
</tr>
<tr>
<td>Target (optional)</td>
<td>STRING</td>
<td>When the Distribution is set to DistributeToAgent, you must include this field with the AgentID of the intended recipient. When the LoginName is set to the LoginName of the agent to receive the chat message, you must also set this field to the login name of the agent to which to chat.</td>
</tr>
<tr>
<td>Message (optional)</td>
<td>STRING</td>
<td>The text of the user message. Maximum message size is 255 bytes.</td>
</tr>
<tr>
<td>LoginName (optional)</td>
<td>STRING</td>
<td>Login name of the agent to receive the chat message. To chat to an agent by login name, set “LoginName” and “Target” to the login name of the agent to which to chat.</td>
</tr>
</tbody>
</table>

**errorcode**
SetAgentGreetingAction

The SetAgentGreetingAction Sets the value of the Agent Greeting Action to enable or disable Agent Greeting for the logged in agent.

Agent Greeting is supported with CTIOS desktops created using the COM or C++ CILs.

Syntax

C++

    int SetAgentGreetingAction(Arguments& args)

COM

    HRESULT SetAgentGreetingAction(/*[in]*/ IArguments *args, int * errorcode)

VB

    SetAgentGreetingAction (args As CTIOSCLIENTLib.IArguments) As Long

Input Parameters

args

Arguments array containing the following fields.

Table 142: SetAgentGreetingAction Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentAction</td>
<td>INT</td>
<td>1 = Disable Agent Greeting for the logged in agent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Enable agent greeting for the logged in agent/-The state to which</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to set the specified agent. The value of this field must be one of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>values in Table 89: AgentState values, on page 234.</td>
</tr>
</tbody>
</table>

Return Values

Default CTIOS return values. For more information, see CIL Coding Conventions, on page 19.
Remarks
A successful request results in an OnAgentGreetingControlConf event. If this request is unsuccessful, the client receives an OnControlFailureConf event.

SetAgentState

The SetAgentState method requests a new agent state. Login and Logout are valid agent states and can be set using the SetAgentState method as well as by using the Login and Logout methods.

Syntax

C++
int SetAgentState(Arguments& args)

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

VB
SetAgentState (args As CTIOSCLIENTLib.IArguments) As Long

Java
int SetAgentState(Arguments args)

.NET
CilError SetAgentState(Arguments args)

Input Parameters

args
Input parameter in the form of an Arguments array that contains one or more of the SetAgentState parameters listed in the following table.

Table 143: SetAgentState Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentState (required)</td>
<td>INT</td>
<td>The state to which to set the specified agent. The value of this field must be one of the values in Table 89: AgentState values, on page 234.</td>
</tr>
<tr>
<td>AgentID (required)</td>
<td>STRING*</td>
<td>The agent's login ID.</td>
</tr>
<tr>
<td>AgentInstrument</td>
<td>STRING*</td>
<td>The agent's instrument number. Optional if Agent Extension is provided.</td>
</tr>
<tr>
<td>AgentPassword (optional)</td>
<td>STRING*</td>
<td>The agent's password.</td>
</tr>
</tbody>
</table>
### StartMonitoringAgent

The `StartMonitoringAgent` method allows the client, which must be a supervisor, to start monitoring the specified `Agent` object. This call causes the supervisor to receive all of the monitored call events (See `IMonitoredCallEvents Interface`, on page 262 in `Event Interfaces and Events`, on page 167) for this agent until the supervisor calls `StopMonitoringAgent`.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentWorkMode (optional)</td>
<td>INT</td>
<td>A value representing the desired work mode of the agent. Used by ACM ECS with default value of ManualIn.</td>
</tr>
<tr>
<td>NumSkillGroups (optional)</td>
<td>INT</td>
<td>The number of Skill Groups that the agent is currently associated with, up to a maximum of 20.</td>
</tr>
<tr>
<td>EventReasonCode (optional)</td>
<td>INT</td>
<td>Reason for logging out. Required for Unified CCE, optional for all other switches.</td>
</tr>
<tr>
<td>PeripheralID (optional)</td>
<td>INT</td>
<td>The Unified ICM Peripheral ID of the ACD the agent is attached to.</td>
</tr>
<tr>
<td>SkillGroupNumber (optional)</td>
<td>INT</td>
<td>The optional, user-defined number of an agent skill group associated with the agent.</td>
</tr>
<tr>
<td>SkillGroupPriority (optional)</td>
<td>INT</td>
<td>The priority of an agent skill group associated with the agent.</td>
</tr>
</tbody>
</table>

*The CTI OS server imposes no restriction on the maximum length of this string. However, such restrictions are generally imposed by your switch/ACD and Cisco CTI Server. Consult the documentation for the switch/ACD or CTI Server for information on length restrictions for this string.

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

A successful request results in an OnAgentStateChanged event. It can also result in OnPreLogout, OnPostLogout, and/or OnLogoutFailed events. If this request is unsuccessful, the client receives an OnControlFailureConf event.
Syntax

C++

    int StartMonitoringAgent(Arguments& args)

COM

    HRESULT StartMonitoringAgent /*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode

VB

    StartMonitoringAgent (args As CTIOSCLIENTLib.IArguments) As Long

Java

    int StartMonitoringAgent(Arguments args)

.NET

    CilError StartMonitoringCall(Arguments args)

Parameters

args

Arguments array that contains the constant CTIOS_AGENTREFERENCE set to the string value of the UniqueObjectID of the agent to be monitored.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Value

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Unified CCE only).

The following code snippet gets the unique object ID string for an agent, then uses uses the SetValue method to store the Agent object ID and string constant CTIOS_AGENTREFERENCE in an Arguments array.

    String StrUID = agent.GetValueString(CTIOS_UNIQUEOBJECTID Id);
    arg.SetValue(CTIOS_AGENTREFERENCE, StrUID);

StartMonitoringAgentTeam

The StartMonitoringAgentTeam method allows the client, which must be a supervisor, to start monitoring the specified agent team. A client supervisor uses this method to receive all of the OnMonitorAgentStateChanged events for every agent on the specified team.

Syntax

C++

    int StartMonitoringAgentTeam (Arguments& args)
StartMonitoringAllAgentTeams

The StartMonitoringAllAgentTeams method allows the client, which must be a supervisor, to start monitoring all the agents on all the supervisor's teams. This causes the supervisor to receive monitored agent events for all of the agents in the supervisor's team (for more information, see IMonitoredAgentEvents Interface, on page 262 in Event Interfaces and Events, on page 167).

Syntax

C++

```cpp
int StartMonitoringAllAgentTeams (Arguments& reserved_args)
```

COM

```c
HRESULT StartMonitoringAllAgentTeams /*[in, optional]*/ IArguments reserved_args, /*[out,retval]*/ int * errorcode)
```

VB

```vb
StartMonitoringAllAgentTeams ([reserved_args as CTIOSCLIENTLib.IArguments]) As Long
```

Java

```java
int StartMonitoringAllAgentTeams (Arguments args)
```

.NET

```csharp
CilError StartMonitoringAllAgentTeams(Arguments args)
```

Parameters

- `args`:
  Arguments array that contains the constant CTIOS_TEAMID set to the integer TeamID to be monitored.

- `errorcode`:
  An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Value

Default CTIOS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Unified CCE only).
Parameters

reserved_args
Not currently used, reserved for future use.

args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Value
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks
This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Unified CCE only).

StartMonitoringCall

Description
The StartMonitoringCall method allows the client, which must be a supervisor, to set the value of the currently monitored call that is used in the SuperviseCall method. Since there is no StopMonitoringCall, call this method with an empty args parameter to clear the value of the currently monitored call.

Syntax

C++
int StartMonitoringCall(Arguments& args)

COM
HRESULT StartMonitoringCall (/*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode)

VB
StartMonitoringCall (args As CTIOSCLIENTLib.IArguments) As Long

Java
int StartMonitoringCall(Arguments args)

.NET
CilError StartMonitoringCall(Arguments args)

Parameters

args
Arguments array that contains the constant CTIOS_CALLREFERENCE set to the string value of the UniqueObjectId of the call to be monitored.

errorCode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Value**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, StopMonitoringAgent).

**StopMonitoringAgent**

The StopMonitoringAgent method allows the client, which must be a supervisor, to stop monitoring the specified Agent object. This stops all Monitored Call events being sent to the supervisor.

**Syntax**

- **C++**
  ```cpp
  int StopMonitoringAgent(Arguments& args)
  ```

- **COM**
  ```csharp
  HRESULT StopMonitoringAgent(/*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode)
  ```

- **VB**
  ```vbnet
  StopMonitoringAgent (args As CTIOSCLIENTLib.IArguments) As Long
  ```

- **Java**
  ```java
  int StopMonitoringAgent(Arguments args)
  ```

- **.NET**
  ```csharp
  CilError StopMonitoringAgent(Arguments args)
  ```

**Parameters**

- **args**
  Arguments array that contains the constant CTIOS_AGENTREFERENCE set to the string value of the UniqueObjectID of the agent to stop monitoring.

- **errorcode**
  An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Value**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, StopMonitoringAgent).
StopMonitoringAgentTeam

The StopMonitoringAgentTeam method allows the client, which must be a supervisor, to stop monitoring all the agents on all the supervisor's teams.

Syntax

C++

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

COM

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

VB

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

Java

```java
int StopMonitoringAgentTeam(Arguments args)
```

.NET

```csharp
CilError StopMonitoringAgentTeam(Arguments args)
```

Parameters

args

Arguments array that contains a constant CTIOS_TEAMID set to the integer TeamID of the team to stop monitoring.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Value

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, StopMonitoringAllAgentTeams

The StopMonitoringAllAgentTeams method allows the client, which must be a supervisor, to stop monitoring all of the agents on all the supervisor's teams.

Syntax

C++

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

VB
StopMonitoringAllAgentTeams([reserved_args as CTIOSCLIENTLib.IArguments]) As Long

Java
int StopMonitoringAllAgentTeams(Arguments args)

.NET
CilError StopMonitoringAgentTeam(Arguments args)

Parameters
reserved_args
Not currently used, reserved for future use.
Java/.NET: args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Value
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks
This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently,

**SuperviseCall**

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

The SuperviseCall method is the CTI OS version of the SUPERVISE_CALL_REQ message. This method is used to barge-into and intercept agent calls by specifying a supervisory action of eSupervisorBargeIn and eSupervisorIntercept respectively. To support Cisco Unified Communications Manager silent monitor, the supervisory action eSupervisorMonitor was added. For more information, see Unified CM-Based Silent Monitoring in Your Application, on page 92.

Syntax

C++
int SuperviseCall(Arguments& args)

COM
HRESULT SuperviseCall /*[in]*/ IArguments * args, /*[out,retval]*/ int errorCode)
VB
SuperviseCall (args As CTIOSCLIENTLib.IArguments ) As Long

Java
int SuperviseCall(Arguments args)

.NET
CilError SuperviseCall(Arguments args)

Parameters

args

An input parameter in the form of a pointer to an Arguments array that contains members with string values that are the UniqueObjectIDs of the desired agent (AgentUniqueObjectID) and call (CallUniqueObjectID). Package these with the keywords “AgentReference” and “CallReference” respectively.

The third required parameter is one of the following integers representing the desired supervisory action.

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>eSupervisorBargeIn</td>
<td>BargeIn to the specified call of the specified agent.</td>
</tr>
<tr>
<td>4</td>
<td>eSupervisorIntercept</td>
<td>Intercept the specified call of the specified agent.</td>
</tr>
<tr>
<td>1</td>
<td>eSupervisorMonitor</td>
<td>Used to silently monitor the call of the specified agent.</td>
</tr>
<tr>
<td>0</td>
<td>eSupervisorClear</td>
<td>Used to clear the silent monitor call.</td>
</tr>
</tbody>
</table>

Table 144: SuperviseCall Parameters

Both SupervisorMonitor and eSupervisorClear only apply to Cisco Unified Communications Manager based silent monitor.

This is packaged with the constant CTIOS_SUPERVISORYACTION or the string “SupervisoryAction”.

Note

Return Values

Default CTIOS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, a BargeIn action is very similar to a Single Step Conference where the agent is the conference controller. As such, only this agent can add other parties to the conference; the supervisor cannot do this.

An Intercept can only be performed by a supervisor who has already performed a BargeIn. The Intercept simply hangs up the original agent, leaving only the customer and the supervisor talking.
E_CTIOS_INVALID_SILENT_MONITOR_MODE is returned when Agent.SuperviseCall() is called when CTIOS Based silent monitor is configured.
Call Object

The Call object provides developers using the CTI OS Client Interface Library with an interface to Call behavior and control. The Call object enables you to perform all call behaviors, such as answering, hanging up, or transferring a call. The Call object represents one call connection of a call. For a call between two parties there are two call connections, and thus there are two distinct CIL Call objects.

The object stores specific call information as properties, including the ICMEnterpriseUniqueID, ANI, DNIS, Call variables, and ExpandedCallContext variables. The Call object is created in response to call events received at the CIL. The Call object properties and state are updated throughout the lifetime of the call connection.

For more information about accessing Call and ECC variables via the GetValue mechanism, see CIL Coding Conventions, on page 19.

Current Call

The Client Interface Library uses the concept of a Current Call. The CTI OS Toolkit uses the Current Call concept as a way for the controls and the application to communicate with each other regarding which call is currently selected and is the one to act on. For example, if an agent has a call and receives a new Ringing call, they might select the Talking call on the grid. At this click, CallAppearanceMgr control calls SetCurrentCall() to make this call the Current Call. When the agent clicks the Hold control, this control calls GetCurrentCall() to obtain a call pointer through which to call the Hold() method. The agent can then select the Ringing call, which again causes the CallAppearanceMgr control to call SetCurrentCall() to make this new call the current call. Then, when the agent clicks the Answer control, this control again calls GetCurrentCall() to obtain a call pointer through which to call the Answer() method.
If your application uses Cisco's out-of-the-box button controls (for more information, see CTI OS ActiveX Controls, on page 119), but not the CallAppearanceMgr grid control, you need to use SetCurrentCall() and GetCurrentCall() for the button controls to enable and disable correctly when switching between multiple calls.

The CurrentCall concept does not place any limitations on call control of non-current calls. All of the call behaviors implemented by method calls on the Call object work on any Call object that is available at the CIL, even if it is not the CurrentCall.

### ECC Variables

The Unified ICM provides a customer-defined data layout for sending call context data with a call. This mechanism is called Expanded Call Context, or ECC. You define ECC variables in the Unified ICM Configuration Manager. You send ECC variables between Unified ICM servers in ECC payloads. After configuring an ECC variable, you must include it in an ECC payload before using it. The mechanism for accessing ECC variables from CTI OS is similar to accessing all other call variables.

To simplify the organization of properties on the Call object, the ECC variables are stored in their own Arguments structure which is nested in the Call object Arguments structure.

### Passing Call Variables

- A consultative transfer is one in which the transferring or forwarding party either connects the caller to a ringing phone or speaks with the third party before connecting the caller to the third party. In a consultative transfer on the same peripheral gateway, if a variable is updated with data during the primary call, and the same variable is then updated with data during the transferred call, the call data from the initial call takes precedence and replaces the call data from the transferred call.

- For calls that are transferred between peripheral gateways, update call variables on the primary call before transferring the call. Only call variable information from the primary call is included in the route request to the other peripheral gateway. Any call variable information that you change after the call is transferred is lost because the call variable information was not included in the route request when the call was transferred.

- The Unified ICM call control variable map is a string that describes the mappings of a peripheral's call control variables to Unified ICM call control variables. You can edit this string to identify the call variables that an agent can change.

### ECC Variable Value Retrieval

To retrieve an ECC variable from the Call object, first retrieve the ECC (Arguments) structure from the Call object using GetValueArray with keyword ECC. Then, retrieve the specific ECC variable required by using its name as the keyword to GetValueInt, GetValueArray, or GetValueString, depending on its type. The following is some sample code for C++ without COM:
Arguments * pECCData = NULL; string sMyECCVariable;
int nMyECCArrayVariable;

if (pCall->IsValid(CTIOS_ECC))
{
pCall->GetValueArray(CTIOS_ECC, &pECCData);

if (pECCData)
{
if (pECCData->IsValid("user.MyECC"))
pECCData->GetValueString->("user.MyECC", &sMyECCVariable);

if(pECCData->IsValid("user.MyArray[2]"))
pECCData->GetValueInt("user.MyArray[2]", &nMyECCArrayVariable);

pECCData->Release();
pECCData = NULL;
}
}

Sample code for VB without COM:

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

If MyCall.IsValid(CTIOS_ECC) = True Then
Set MyECCData = MyCall.GetValueArray(CTIOS_ECC)

If MyECCData.IsValid("user.MyECC") Then
MyECCVariable = MyECCData.GetValueString("user.MyECC")
End If

If MyECCData.IsValid("user.MyArray[2]") Then
MyECCArrayVariable = MyECCData.GetValueInt("user.MyArray[2]")
End If
End If

The same thing in Java is as follows:

if(Call != null){
Arguments rArgEcc = new Arguments();
rArgEcc = Call.GetValueArray(CTIOS_ECC);
if(null != rArgEcc)
{
rArgEcc.NumElements();
Integer intVal = rArgEcc.GetValueIntObj("user.MyECC");
String strVal = rArgEcc.GetValueString("userMyArray[2]");
}
}

ECC Values

If you want to add ECC values to a call without deleting ones that are already set in the call, retrieve the ECC variables and then add the new ones as shown in C++ without COM:
Arguments & RequestArgs = Arguments::CreateInstance();
Arguments * pECCData = NULL;

// presumes that we have a Call object pointer in pCall
if (pCall->IsValid(CTIOS_ECC))
pCall->GetValueArray(CTIOS_ECC, &pECCData);
else
Arguments::CreateInstance(&pECCData);

pECCData->AddItem("user.MyECC", "FirstECCVariable");
pECCData->AddItem("user.MyArray[2]", 2222);
RequestArgs.AddItem(CTIOS_ECC, *pECCData);
pCall->SetCallData(RequestArgs);
RequestArgs.Release();
pECCData->Release();

The same thing in VB is as follows:

Dim MyRequestArgs As New CTIOSARGUMENTSLib.Arguments
Dim MyECCData As CTIOSARGUMENTSLib.Arguments

If MyCall.IsValid(CTIOS_ECC) Then
Set MyECCData = MyCall.GetValueArray(CTIOS_ECC)
Else
Set MyECCData = New CTIOSARGUMENTSLib.Arguments
End If

MyECCData.AddItem("user.MyECC", "FirstECCVariable")
MyECCData.AddItem("user.MyArray[2]", 2222)
MyRequestArgs.AddItem("ECC", MyECCData)
MyCall.SetCallData(MyRequestArgs)

The same thing in Java is as follows:

Arguments rRequestArgs = new Arguments();
if(Call != null)
{
    Arguments rArgEcc = Call.GetValueArray(CTIOS_ECC);
    if(null == rArgEcc)
    {
        rArgEcc = new Arguments();
    }
    rArgEcc.SetValue("user.MyEcc", 22222);
    rArgEcc.SetValue("user.MyArray[3]", "new data");
    rRequestArgs.SetValue(CTIOS_ECC, rArgEcc);
    Call.SetCallData(rRequestArgs);
}
Properties

The following table lists the available Call object properties.

Note

The data type listed for each keyword is the standardized data type discussed in CTI OS CIL Data Types, on page 19. For more information, see Table 5: CTI OS CIL Data Type, on page 20 for the appropriate language specific types for these keywords.

Table 145: Call Object Properties

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANI</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>CallerEnteredDigits</td>
<td>STRING</td>
<td>The digits entered by the caller in response to VRU prompting.</td>
</tr>
<tr>
<td>CallStatus</td>
<td>SHORT</td>
<td>The current status of the call.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>CallVariable1</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable2</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable3</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable4</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable5</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable6</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable7</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable8</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable9</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallVariable10</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>CallWrapupData</td>
<td>STRING</td>
<td>Call-related variable data.</td>
</tr>
<tr>
<td>ClassIdentifier</td>
<td>INT</td>
<td>Private; for internal use only.</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>DNIS</td>
<td>STRING</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ECC</td>
<td>ARGUMENTS</td>
<td>Arguments structure of key-value pairs of ECC variables.</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID</td>
<td>STRING</td>
<td>Required only when the call is pre-routed.</td>
</tr>
<tr>
<td>LineType</td>
<td>SHORT</td>
<td>Indicates the type of the teleset line.</td>
</tr>
<tr>
<td>MeasuredCallQTime</td>
<td>INT</td>
<td>Number of seconds this call was in a local queue before being delivered to the agent.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>The Unified ICM PeripheralID of the ACD where the call activity occurred.</td>
</tr>
<tr>
<td>RouterCallKeyCallID</td>
<td>INT</td>
<td>The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.</td>
</tr>
<tr>
<td>Router CallKeyDay</td>
<td>INT</td>
<td>Together with the RouterCall KeyCallID field forms the unique 64-bit key for locating this call's records in the Unified ICM database. Only provided for Post-routed and Translation-routed calls.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The Unified ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The system-assigned identifier of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.</td>
</tr>
</tbody>
</table>
### Methods

The following table lists the available Call object methods.

**Table 146: Call Object Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternate</td>
<td>Places the current call on hold and retrieves a previously held call.</td>
</tr>
<tr>
<td>Answer</td>
<td>Answers a call that is in the alerting or ringing state.</td>
</tr>
<tr>
<td>Clear</td>
<td>Clears a call, dropping all parties to the call.</td>
</tr>
<tr>
<td>ClearConnection</td>
<td>Hangs up a call, leaving other parties in a conference call. If there are only two parties on the call it clears the call.</td>
</tr>
<tr>
<td>Conference</td>
<td>Either establishes a three party conference call or adds a new party to an existing conference call.</td>
</tr>
<tr>
<td>DumpProperties</td>
<td>For more information, see <a href="#">CtiOs Object, on page 291</a>.</td>
</tr>
<tr>
<td>GetAllProperties</td>
<td>For more information, see <a href="#">CtiOs Object, on page 291</a>.</td>
</tr>
<tr>
<td>GetCallContext</td>
<td>Gets data associated with the call other than call and expanded call context (ECC) variables.</td>
</tr>
<tr>
<td>GetCallData</td>
<td>Obtains call and expanded call context (ECC) variables.</td>
</tr>
<tr>
<td>GetElement</td>
<td>For more information, see <a href="#">CtiOs Object, on page 291</a>.</td>
</tr>
<tr>
<td>GetLastError (.NET only)</td>
<td>Returns the last error that occurred on the calling thread.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetNumProperties</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetName</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetPropertyType</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetValue methods</td>
<td>Retrieve a property from the Call object based on the property's name key.</td>
</tr>
<tr>
<td>Hold</td>
<td>Places a current call on hold.</td>
</tr>
<tr>
<td>IsValid</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>MakeConsultCall</td>
<td>Places a current call on hold and makes a new call.</td>
</tr>
<tr>
<td>Reconnect</td>
<td>Clears the current call and then retrieves a held call.</td>
</tr>
<tr>
<td>Retrieve</td>
<td>Retrieves a held call.</td>
</tr>
<tr>
<td>SetCallData</td>
<td>Sets call and expanded call context (ECC) variables.</td>
</tr>
<tr>
<td>SendDTMFSignal</td>
<td>Requests the ACD to send a sequence of DTMF tones.</td>
</tr>
<tr>
<td>SingleStepConference</td>
<td>Performs a single step conference.</td>
</tr>
<tr>
<td>SingleStepTransfer</td>
<td>Performs a single step transfer.</td>
</tr>
<tr>
<td>Snapshot</td>
<td>Issues a server request to get the current call information, including call data and a list of associated devices and the connection state for the call of each device.</td>
</tr>
<tr>
<td>StartRecord</td>
<td>Starts recording of a call.</td>
</tr>
<tr>
<td>StopRecord</td>
<td>Stops recording of a call.</td>
</tr>
<tr>
<td>Transfer</td>
<td>Transfers a call to a third party.</td>
</tr>
</tbody>
</table>

### Argument Parameters

The following rules apply to the optional_args and reserved_args parameters in Call object methods:

- In VB, you can ignore these parameters altogether. For example, you can treat the line:

  ```vbnet
  Answer([reserved_args As IArguments]) As Long
  ```

  as follows:

  ```vbnet
  Answer()
  ```

- To ignore these parameters in COM you must send a NULL, as shown:
Alternate

The Alternate method combines the action of placing a talking call on hold and then retrieving a previously held call at the same device. If there are only two calls at the device, this method can be called via either the current or the held call.

Syntax

C++
```
int Alternate()
int Alternate(Arguments & reserved_args);
```

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

VB
```
Alternate([reserved_args As IArguments]) As Long
```

Java
```
int Alternate(Arguments rArgs);
```

.NET
```
CilError Alternate(Arguments args)
```

Parameters

reserved_args
A valid Arguments object, which can be empty. Not currently used, reserved for future use.
errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

For switches that allow more than two calls at a device (for example G3), make this request only through the desired held call, because of the ambiguity caused by multiple held calls at the device.

You must make the Alternate request via a call whose status is either LCS_CONNECTED or LCS_HELD or it fails.

The following events are received if this request is successful.

For the call making the Alternate request:

- OnAlternateCallConf event
For the originally current call:
  • OnCallHeld event

For the originally held call:
  • OnCallRetrieved event

The following events are received by the call making the Alternate request if this request fails:
  • OnControlFailureConf event

**Answer**

The Answer method answers a call that is in the alerting or ringing state (i.e., call status of LCS_ALERTING).

**Syntax**

C++
```
int Answer()
int Answer(Arguments & reserved_args)
```

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

VB
```
Answer([reserved_args As IArguments]) As Long
```

Java
```
int Answer(Arguments rArgs)
```

.NET
```
CilError Answer(Arguments args)
```

**Parameters**

reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Value**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

You can answer a call after the OnCallDelivered event is received. You must make the Answer request via a call whose call status LCS_ALERTING or it fails.

The following events are received if this request is successful:
The following events are received if this request fails:
  • OnControlFailureConf event

Clear

The Clear method clears the call and drops all parties to the call.

Syntax

C++
int Clear() / int Clear(Arguments & reserved_args);

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

VB
Clear([reserved_args As IArguments]) As Long

Java
int Clear(Arguments rArgs);

.NET
CilError Clear(Arguments args);

Parameters

reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Value

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

In the case of a multi-party Conference call, calling Clear() results in all of the parties to the call being hung up. (If this is not the desired behavior, see the ClearConnection method.) Under certain switches the Clear request is made via a call whose status is LCS_CONNECT or LCS_INITIATE or it fails. Many other switches allow the Clear method to be called via a call whose status is LCS_ALERTING or LCS_HOLD. It can never be made via a call whose status is LCS_NULL indicating that it is already cleared.

The following events are received if this request is successful:
  • OnClearCallConf event
The following events are received if this request fails:

- OnControlFailureConf event

---

**ClearConnection**

If there are only two parties to the call, ClearConnection clears the call. However, for a multi-party conference call, only one connection is dropped, which is its own connection.

**Syntax**

**C++**

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

**COM**

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

**VB**

```vb
ClearConnection([reserved_args As IArguments]) As Long
```

**Java**

```java
int ClearConnection(Arguments rArgs);
```

**.NET**

```c
CilError ClearConnection(Arguments args);
```

**Parameters**

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

- **errorcode**
  An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Value**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

As with the Clear method, under certain switches you must make the ClearConnection request via a call whose status is LCS_CONNECT or LCS_INITIATE or it fails. Many other switches allow the Clear method to be
called via a call whose status is LCS_ALERTING or LCS_HOLD. It can never be made via a call whose status is LCS_NULL indicating that it is already cleared.

The following events are received if this request is successful:

- OnClearConnectionConf event
- OnCallConnectionICleared event

If this is a two party call, these events are followed by:

- OnCallICleared event

The following events are received if this request fails:

- OnControlFailureConf event

### Conference

The Conference method either begins a new conference call or adds an additional call to an existing conference call. When it begins a new conference call, it combines an original two-party call with a two-party consult call (where the two calls have a common party) into a single three party call. Only the common party (which is called the “Conference Controller”) can call this method to make the new conference call. You can call this method on either of the Conference Controller's calls.

#### Syntax

**C++**

```cpp
int Conference();
int Conference(Arguments& optional_args)
```

**COM**

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

**VB**

```vb
Conference([optional_args As IArguments]) As Long
```

**Java**

```java
int Conference(Arguments optional_args)
```

**.NET**

```csharp
CilError Conference(Arguments optional_args)
```

#### Parameters

**optional_args**

An optional input parameter, which is a pointer or reference to an Arguments array that contains a member with the string value that is the UniqueObjectID of the call to which this call is conferenced. If this argument is used, add it to the Arguments parameter with the keyword of “CallReferenceObjectId”. This is only necessary in an environment where there are multiple held calls and the request is made through the talking call. If the request is made through a specific held call in this scenario, or if there are only two calls at the device, this parameter is unnecessary.

**errorcode**
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Value**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

Before making this request, the original call must be in the held state and the consult call in the talking state or the request fails. Therefore, if the calls are alternated (see Alternate), they must be alternated again to return the two calls to their appropriate states.

If there are only two calls at the device, you can call this method using either the current or held call. For switches which allow more than two calls at a device (for example G3), make this request through the desired held call to avoid the ambiguity caused by multiple held calls at the device. Otherwise, indicate the desired held call using the optional parameter.

You must make the Conference request via a call whose call status is LCS_CONNECT or LCS_HELD or it fails.

On certain switches (notably Unified CCE), only the Conference Controller (the party that first initiated the conference call) can add additional parties to an existing conference call.

The following events are received if this request is successful:

- OnConferenceCallConf event
- OnCallConferenced event

The following events are received if this request fails:

- OnControlFailureConf event

### GetCallContext

The GetCallContext method returns an Arguments array containing the values for call properties other than CallVariables and ECC Variables, such as ANI, DNIS, and the other properties listed in the following table.

#### Syntax

**C++**

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

**COM**

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

**VB**

```vbnet
GetCallContext (CTIOSCLIENTLib.IArguments args)
```

**Java**

```java
Arguments GetCallContext()
```

**.NET**

```csharp
Arguments GetCallContext()
```
**Parameters**

args

C++, COM, and VB: An output parameter containing a reference or a pointer to an Arguments array containing any of the members in the following table that are present in the call.

**Return Value**

C++, COM, and VB: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.

Java/.NET: A reference to an Arguments array that, on return, holds name/value pairs from the following table. You can access any of these parameters included from the Arguments array using the associated keyword.

---

**Table 147: GetCallContext Arguments Array Contents**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANI</td>
<td>STRING</td>
<td>The calling line ID of the caller.</td>
</tr>
<tr>
<td>CallerEnteredDigits</td>
<td>STRING</td>
<td>The digits entered by the caller in response to VRU prompting.</td>
</tr>
<tr>
<td>CallType</td>
<td>SHORT</td>
<td>The general classification of the call type.</td>
</tr>
<tr>
<td>CallWrapupData</td>
<td>STRING</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>ConnectionCallID</td>
<td>UINT</td>
<td>The Call ID value assigned to this call by the peripheral or the Unified ICM.</td>
</tr>
<tr>
<td>DialedNumber</td>
<td>STRING</td>
<td>The number dialed.</td>
</tr>
<tr>
<td>DNIS</td>
<td>STRING</td>
<td>The DNIS provided with the call.</td>
</tr>
<tr>
<td>ICMEnterpriseUniqueID</td>
<td>STRING</td>
<td>A unique identifier for this contact throughout the enterprise. This can track a single customer contact across multiple sites, for example, when a call is transferred between agents.</td>
</tr>
<tr>
<td>ServiceID</td>
<td>INT</td>
<td>The Unified ICM identifier for the Service to which this call was routed.</td>
</tr>
<tr>
<td>ServiceNumber</td>
<td>INT</td>
<td>The ACD number of the Service to which this call was routed.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>INT</td>
<td>The system-assigned identifier for the SkillGroup to which this call was routed.</td>
</tr>
</tbody>
</table>
### Call Object

#### GetCallData

The GetCallData method returns the values of CallVariable1 through CallVariable10 and all of the ECC (Extended CallContext) variables.

**Syntax**

**C++**

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

**COM**

```csharp
HRESULT GetCallData (/*[out,retval]*/ IArguments ** args)
```

**VB**

```vb
GetCallData (CTIOSCLIENTLib.IArguments args)
```

**Java**

```java
Arguments GetCallData()
```

**.NET**

```csharp
Arguments GetCallData()
```

**Parameters**

`args`

C++, COM, and VB: An output parameter containing a reference or a pointer to an Arguments array containing the call data, as described under Remarks.

**Return Value**

C++, COM, and VB: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.

Java/.NET: A reference to an Arguments array that, on return, holds parameters described under Remarks.

---

#### Description

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>An optional, user-defined number of the SkillGroup at the ACD to which this call was routed.</td>
</tr>
<tr>
<td>UniqueObjectID</td>
<td>STRING</td>
<td>A unique object ID for the call.</td>
</tr>
<tr>
<td>UserToUserInfo</td>
<td>STRING</td>
<td>The ISDN user-to-user information element.</td>
</tr>
</tbody>
</table>

**Remarks**

This is a convenience method to call and get all of a call's non-CallVariable data at one time. If only certain data members are desired, call the appropriate GetValue method for each instead.
Remarks
This is a convenience method to call and get all of a call’s CallVariables (1 through 10) and ECC Call Variables at one time. If only certain call variables are desired, call the appropriate GetValue method for each instead.

Access the data in the following way:

- To access the values for individual CallVariables from the arguments parameter, use GetValueString with either the keywords of “CallVariable1” through “CallVariable10”.

To access ECC call data, use the following procedure:

- First, get the ECC variables as a whole from the arguments parameter, using GetValueArray with the keyword “ECC”. This returns another Arguments array that is nested in the Arguments array returned from GetCallData.

- To access an individual ECC scalar variable from this Arguments array, use the appropriate GetValueString, GetValueInt, etc. depending on the variable's type, using the string keyword “user.VariableName”.

- To access an individual ECC array variable from this Arguments array, use the appropriate GetValueString, GetValueInt, etc. depending on the variable's type, using the string keyword “user.ArrayName[n]”, where n is a zero based integer that notes the offset in the array.

Hold

The Hold method holds a current call.

Syntax

C++

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

COM

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

VB

```vb
Hold([reserved_args As IArguments]) As Long
```

Java

```java
Arguments Hold(Arguments rArgs)
```

.NET

```csharp
Arguments Hold(Arguments args)
```

Parameters

reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.
Return Value
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks
You must make the Hold request via a call whose call status is LCS_CONNECT or it fails.
The following events are received if this request is successful:
- OnHoldCallConf event
- OnCallHeld event

The following events are received if this request fails:
- OnControlFailureConf event

MakeConsultCall

The MakeConsultCall method initiates the combined action of placing the associated current call on hold and then making a new call. By default, the call context data (including call variables) of the current call is used to initialize the context data of the new consultation call. The application can override some or all of the original call context in the consultation call by providing the desired values in this request.

The simplest form of the request only requires a dialed number and a consult type. The request can also include optional parameters, as listed in the following table.

Syntax
C++
```cpp
int MakeConsultCall (Arguments& args)
```

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

VB
```
MakeConsultCall (args As CTIOSCLIENTLib.IArguments) As Long
```

Java
```
int MakeConsultCall(Arguments args)
```

.NET
```
CilError MakeConsultCall(Arguments args)
```

Parameters
args
An output parameter of either a reference or a pointer to an Arguments array that contains parameters from the following table. Any of these parameters included are added to the Arguments array using the associated key word.
### Table 148: MakeConsultCall Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DialedNumber (required)</td>
<td>STRING, maximum length 40</td>
<td>Dialed number; the number to be dialed to establish the new call.</td>
</tr>
<tr>
<td>ConsultType (required)</td>
<td>INT</td>
<td>A value specifying whether this consult call is in preparation for either a transfer or a conference, as specified in the ConsultType Table.</td>
</tr>
<tr>
<td>CallPlacementType (optional)</td>
<td>STRING, maximum length 40</td>
<td>A value specifying how the call is to be placed identified in Table 149: CallPlacementType Values, on page 408.</td>
</tr>
<tr>
<td>CallMannerType (optional)</td>
<td>INT</td>
<td>A value specifying additional call processing options identified in Table 150: CallMannerType Values, on page 408.</td>
</tr>
<tr>
<td>CallOption (optional)</td>
<td>INT</td>
<td>A value from Table 151: CallOption Values, on page 409 specifying additional peripheral-specific call options.</td>
</tr>
<tr>
<td>FacilityType (optional)</td>
<td>INT</td>
<td>A value from Table 152: FacilityType Values, on page 409 indicating the type of facility to be used.</td>
</tr>
<tr>
<td>Priority (optional)</td>
<td>BOOL</td>
<td>Set this field to TRUE if the call should receive priority handling.</td>
</tr>
<tr>
<td>PostRoute (optional)</td>
<td>BOOL</td>
<td>When this field is set to TRUE, the Post-Routing capabilities of the Unified ICM determine the new call destination.</td>
</tr>
<tr>
<td>UserToUserInfo (optional)</td>
<td>STRING, maximum length 40</td>
<td>The ISDN user-to-user information.</td>
</tr>
<tr>
<td>CallVariable1 (optional)</td>
<td>STRING, maximum length 40</td>
<td>Call variable data that is set in the new call in place of the corresponding data in the current call.</td>
</tr>
<tr>
<td>CallVariable10 (optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECC</td>
<td>ARGUMENTS</td>
<td>ECC data that is set in the new call in place of the corresponding data in the current call.</td>
</tr>
</tbody>
</table>
### Call Object

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallWrapupData (optional)</td>
<td>STRING, maximum length 40</td>
<td>Call-related wrapup data.</td>
</tr>
<tr>
<td>FacilityCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>A trunk access code, split extension, or other data needed to access the chosen facility.</td>
</tr>
<tr>
<td>AuthorizationCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>An authorization code needed to access the resources required to initiate the call.</td>
</tr>
<tr>
<td>AccountCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>A cost-accounting or client number used by the peripheral for charge-back purposes.</td>
</tr>
</tbody>
</table>

#### Table 149: CallPlacementType Values

<table>
<thead>
<tr>
<th>CallPlacementType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPT_UNSPECIFIED</td>
<td>Use default call placement.</td>
<td>0</td>
</tr>
<tr>
<td>CPT_LINE_CALL</td>
<td>An inside line call.</td>
<td>1</td>
</tr>
<tr>
<td>CPT_OUTBOUND</td>
<td>An outbound call.</td>
<td>2</td>
</tr>
<tr>
<td>CPT_OUTBOUND_NO_ACCESS_CODE</td>
<td>An outbound call that does not require an access code.</td>
<td>3</td>
</tr>
<tr>
<td>CPT_DIRECT_POSITION</td>
<td>A call placed directly to a specific position.</td>
<td>4</td>
</tr>
<tr>
<td>CPT_DIRECT_AGENT</td>
<td>A call placed directly to a specific agent.</td>
<td>5</td>
</tr>
<tr>
<td>CPT_SUPERVISOR_ASSIST</td>
<td>A call placed to a supervisor for call handling assistance.</td>
<td>6</td>
</tr>
</tbody>
</table>

#### Table 150: CallMannerType Values

<table>
<thead>
<tr>
<th>CallMannerType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMT_UNSPECIFIED</td>
<td>Use default call manner.</td>
<td>0</td>
</tr>
<tr>
<td>CMT_POLITE</td>
<td>Attempt the call only if the originating device is idle.</td>
<td>1</td>
</tr>
<tr>
<td>CMT_BELLIGERENT</td>
<td>Always attempt the call, disconnecting any currently active call.</td>
<td>2</td>
</tr>
<tr>
<td>CMT_SEMI_POLITE</td>
<td>Attempt the call only if the originating device is idle or is receiving dial tone.</td>
<td>3</td>
</tr>
</tbody>
</table>
### Table 151: CallOption Values

<table>
<thead>
<tr>
<th>CallOption</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPT_UNSPECIFIED</td>
<td>No call options specified, use defaults.</td>
<td>0</td>
</tr>
<tr>
<td>COPT_CALLING_AGENT_ONLINE</td>
<td>Attempt the call only if the calling agent is “online” (available to interact with the destination party).</td>
<td>1</td>
</tr>
<tr>
<td>COPT_CALLING_AGENT_RESERVED</td>
<td>Attempt the call only if ACDNR on the calling agent's set is activated.</td>
<td>2</td>
</tr>
<tr>
<td>COPT_CALLING_AGENT_NOT_RESERVED</td>
<td>Attempt the call only if ACDNR on the calling agent's set is not activated.</td>
<td>3</td>
</tr>
<tr>
<td>COPT_CALLING_AGENT_BUZZ_BASE</td>
<td>Applies a buzz to the base of the telephone set as the call is initiated.</td>
<td>4</td>
</tr>
<tr>
<td>COPT_CALLING_AGENT_BEEP_HSET</td>
<td>Applies a tone to the agent headset as the call is initiated.</td>
<td>5</td>
</tr>
<tr>
<td>COPT_SERVICE_CIRCUIT_ON</td>
<td>Applies a call classifier to the call (ACM ECS)</td>
<td>6</td>
</tr>
</tbody>
</table>

### Table 152: FacilityType Values

<table>
<thead>
<tr>
<th>FacilityType</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT_UNSPECIFIED</td>
<td>Use default facility type.</td>
<td>0</td>
</tr>
<tr>
<td>FT_TRUNK_GROUP</td>
<td>Facility is a trunk group.</td>
<td>1</td>
</tr>
<tr>
<td>FT_SKILL_GROUP</td>
<td>Facility is a skill group or split.</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 153: AnsweringMachine Values

<table>
<thead>
<tr>
<th>AnsweringMachine</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM_UNSPECIFIED</td>
<td>Use default behavior.</td>
<td>0</td>
</tr>
<tr>
<td>AM_CONNECT</td>
<td>Connect call to agent when call is answered by an answering machine.</td>
<td>1</td>
</tr>
<tr>
<td>AM_DISCONNECT</td>
<td>Disconnect call when call is answered by an answering machine.</td>
<td>2</td>
</tr>
<tr>
<td>AM_NONE</td>
<td>Do not use answering machine detection.</td>
<td>3</td>
</tr>
</tbody>
</table>
errorcode

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

You must make the MakeConsultCall request via a call whose call status is LCS_CONNECT or it fails. Calling MakeConsultCall successfully results in the same events as a successful MakeCall called on the agent.

The following events are received if this request is successful:

For the call making the MakeConsultCallRequest:

- OnMakeConsultCallConf event
- OnCallHeld event

For the newly created outgoing consult call:

- OnBeginCall event
- OnServiceInitiated event
- OnCallOriginated event
- OnCallDelivered event

For the new connection that is ringing as a result of the consult call:

- OnBeginCall event
- OnCallDelivered event

The following events are received if this request fails:

- OnControlFailureConf event

<table>
<thead>
<tr>
<th>AnsweringMachine</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM_NONE_NO_MODEM</td>
<td>Do not use answering machine detection, but disconnect call if answered by a modem.</td>
<td>4</td>
</tr>
<tr>
<td>AM_CONNECT_NO_MODEM</td>
<td>Connect call when call is answered by an answering machine, disconnect call if answered by a modem.</td>
<td>5</td>
</tr>
</tbody>
</table>
The Reconnect method combines the action of releasing a current call and then retrieving a previously held call at the same device. If there are only two calls at the device, this method can be called via either the talking or the held call.

**Syntax**

**C++**

```cpp
int Reconnect()
int Reconnect(Arguments & reserved_args)
```

**COM**

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

**VB**

```vb
Reconnect([reserved_args As IArguments]) As Long
```

**Java**

```java
int Reconnect(Arguments rArgs)
```

**.NET**

```csharp
CilError Reconnect(Arguments args)
```

**Parameters**

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

- **errorcode**
  
  An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

For switches that allow more than two calls at a device (for example G3), make this request only through the desired held call because of the ambiguity caused by multiple held calls at the device.

You must make the Alternate request via a call whose status is either LCS_CONNECT or LCS_HELD or it fails.

The following events are received if this request is successful:

- For the call making the Reconnect request:
  - OnReconnectCallConf event

- For the originally current call:
  - OnCallConnectionCleared event
• OnCallCleared event
• OnCallEnd event

For the originally held call:
• OnCallRetrieved event

The following events are received by the call making the Alternate request if this request fails:
• OnControlFailureConf event

## Retrieve

The Retrieve method unholds a held call.

### Syntax

#### C++

```cpp
int Retrieve()
int Retrieve(Arguments & reserved_args)
```

#### COM

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

#### VB

```vb
Retrieve([reserved_args As IArguments]) As Long
```

#### Java

```java
int Retrieve(Arguments rArgs)
```

#### .NET

```csharp
CilError Retrieve(Arguments args)
```

### Parameters

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

- **errorcode**
  An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

### Return Values

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

### Remarks

You must make the Retrieve request via a call whose call status is LCS_HELD or it fails.

The following events are received if this request is successful:
• OnRetrieveCallConf event
The following events are received if this request fails:

• OnControlFailureConf event

**SendDTMFSignal**

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

**Syntax**

C++

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

COM

```c
HRESULT SendDTMFSignal /*[in]*/ args *arguments, /*[out, retval]*/ int * errorcode)
```

VB

```vb
SendDTMFSignal (args As CTIOSCLIENTLib.IArguments, errorcode As Long)
```

Java

```java
int SendDTMFSignal(Arguments rArgs)
```

.NET

```c
CilError SendDTMFSignal(Arguments args)
```

**Parameters**

`args`

An input parameter of either a reference or a pointer to an Arguments array containing parameters from following table. You can add any of these parameters included to the Arguments array using the associated key word.

**Table 154: SendDTMFSignal Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTMFString (required)</td>
<td>STRING. maximum length 32</td>
<td>The sequence of tones to be generated.</td>
</tr>
<tr>
<td>ToneDuration (optional)</td>
<td>INT</td>
<td>Specifies the duration in milliseconds of DTMF digit tones. Use 0 to take the default. Can be ignored if the peripheral is unable to alter the DTMF tone timing.</td>
</tr>
<tr>
<td>PauseDuration (optional)</td>
<td>INT</td>
<td>Specifies the duration in milliseconds of DTMF inter-digit spacing. Use 0 to take the default. Can be ignored if the peripheral is unable to alter the DTMF tone timing.</td>
</tr>
</tbody>
</table>
errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks
• The OnSendDTMFSignalConf event is received if this request succeeds.
• The OnControlFailureConf event is received if this request fails.

SetCallData
The SetCallData method sets any or all of a call's CallVariables (1 through 10) and ECC data at one time.

Note
• When writing a custom application, in any language, call variables are not blanked out if they are set to a NULL value. While the application attempts to clear any call variable using a NULL value, the CTI OS server application ignores the NULL value call variables and does not pass them to the CTI Server application. As a result, the call variables set to NULL are not reset.
• To clear the value of a call variable, set its value to a blank character. Setting the call variable to a single space character places a space in the call variable's values for the duration of the call. This space is considered a NULL value by the application.

Syntax
C++
int SetCallData(Arguments & args)

COM
HRESULT SetCallData (/*[in]*/ args *arguments, /*[out]*/ int * errorcode)

VB
SetCallData (args As CTIOSCLIENTLib.IArguments, errorcode As Long)

Java
int SetCallData(Arguments rArgs)

.NET
CilError SetCallData(Arguments args)

Parameters
args
An input parameter of either a reference or a pointer to an Arguments array containing parameters described under Remarks for GetCallData.
errorcode

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values

Default CTIOS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

You must specify the data for all elements in the Arguments array, not just those elements that you want to change. Failure to do so causes the unchanged elements to disappear.

The following events are sent if this request succeeds:

- OnSetCallDataConf
- OnCallDataUpdate

The OnControlFailureConf event is sent if this request fails.

**SingleStepConference**

The SingleStepConference method initiates a one-step conference without the intermediate consultative call so that when the called party answers, they are joined in the current call. This method requires a DialedNumber argument. This method is not supported under all switches.

---

**Note**

The SingleStepConference method is not supported for the Unified CCE.

**Syntax**

C++

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

COM

```cpp
HRESULT SingleStepConference (IArguments *args, int * errorcode)
```

VB

```vb
SingleStepConference (args As CTIOSCLIENTLib.IArguments, errorcode As Long)
```

Java

```java
int SingleStepConference(Arguments rArgs)
```

.NET

```csharp
CilError SingleStepConference(Arguments args)
```

**Parameters**

args

An output parameter of either a reference or a pointer to an Arguments array containing parameters from the following table. You can add any of these parameters included to the Arguments array using the associated keyword.
### Table 155: SingleStepConference Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DialedNumber (required)</td>
<td>STRING, maximum length 40</td>
<td>Dialed number; the number to be dialed to establish the new call.</td>
</tr>
<tr>
<td>CallPlacementType (optional)</td>
<td>STRING, maximum length 40</td>
<td>A value specifying how the call is to be placed identified in Table 149: CallPlacementType Values, on page 408.</td>
</tr>
<tr>
<td>CallMannerType (optional)</td>
<td>INT</td>
<td>A value specifying additional call processing options identified in Table 150: CallMannerType Values, on page 408.</td>
</tr>
<tr>
<td>AlertRings (optional)</td>
<td>INT</td>
<td>The maximum amount of time that the call's destination will remain alerting, specified as an approximate number of rings. A zero value indicates that the peripheral default (typically 10 rings) should be used.</td>
</tr>
<tr>
<td>CallOption (optional)</td>
<td>INT</td>
<td>A value from Table 151: CallOption Values, on page 409 specifying additional peripheral-specific call options.</td>
</tr>
<tr>
<td>FacilityType (optional)</td>
<td>INT</td>
<td>A value from Table 152: FacilityType Values, on page 409 indicating the type of facility to be used.</td>
</tr>
<tr>
<td>AnsweringMachine (optional)</td>
<td>INT</td>
<td>A value from Table 153: AnsweringMachine Values, on page 409 specifying the action to be taken if the call is answered by an answering machine.</td>
</tr>
<tr>
<td>Priority (optional)</td>
<td>BOOL</td>
<td>Set this field to TRUE if the call should receive priority handling.</td>
</tr>
<tr>
<td>PostRoute (optional)</td>
<td>BOOL</td>
<td>When this field is set to TRUE, the Post-Routing capabilities of the Unified ICM determines the new call destination.</td>
</tr>
<tr>
<td>UserToUserInfo (optional)</td>
<td>STRING, maximum length 40</td>
<td>The ISDN user-to-user information.</td>
</tr>
<tr>
<td>CallVariable1 (optional)</td>
<td>STRING, maximum length 40</td>
<td>Set call variable data in the new call in place of the corresponding data in the current call.</td>
</tr>
</tbody>
</table>
### Parameter, Type, Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>CallVariable10 (optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECC</td>
<td>ARGUMENTS</td>
<td>Set ECC data in the new call in place of the corresponding data in the current call.</td>
</tr>
<tr>
<td>FacilityCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>A trunk access code, split extension, or other data needed to access the chosen facility.</td>
</tr>
<tr>
<td>AuthorizationCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>An authorization code needed to access the resources required to initiate the call.</td>
</tr>
<tr>
<td>AccountCode (optional)</td>
<td>STRING, maximum length 40</td>
<td>A cost-accounting or client number used by the peripheral for charge-back purposes.</td>
</tr>
</tbody>
</table>

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

The DialedNumber is the only required member necessary in the Arguments parameter. A SingleStepConference request fails if the call's status is not LCS_CONNECT.

The following events are received if this request is successful:

- OnAgentStateChange event (Hold)
- OnCallHeld event
- OnAgentStateChange event (Talking)
- OnBeginCall event
- OnServiceInitiated event
- OnCallOriginated event
- OnCallDelivered event
- OnCallConferenced event
- OnCallEnd event
- ConferenceCallConf event
The OnControlFailureConf event is received if this request fails.

**SingleStepTransfer**

The SingleStepTransfer method initiates a one-step transfer without the intermediate consultative call. When the called party answers the call, the called party is talking to the party to be transferred and the transferring party drops out of the call. The method requires a DialedNumber argument.

**Syntax**

**C++**

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

**COM**

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

**VB**

```vb
SingleStepTransfer (args As CTIOSCLIENTLib.IArguments, errorcode As Long)
```

**Java**

```java
int SingleStepTransfer(Arguments rASrgs)
```

**.NET**

```c
CilError SingleStepTransfer(Arguments args)
```

**Parameters**

- **args**
  An output parameter of either a reference or a pointer to an Arguments array containing parameters from Table 155: SingleStepConference Parameters, on page 416. You can add any of these parameters included to the Arguments array using the associated keyword.

- **errorcode**
  An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Snapshot**

The Snapshot method issues a server request to retrieve the current call information. If values are passed in the optional args parameter, the snapshot request returns the server's current call values for only the requested arguments. Otherwise all call information is returned, including the fields described under GetCallContext and GetCallData. For more information about OnCallDataUpdate, see OnCallDataUpdate in Event Interfaces and Events, on page 167.
StartRecord

The StartRecord method is used to start recording a call.

Syntax

C++

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

COM

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

VB

```vb
StartRecord([optional_args As IArguments]) As Long
```

Java

```java
int StartRecord(Arguments rArgs)
```

.NET

```csharp
CilError Snapshot(Arguments Args)
```

Parameters

optional_args
An input parameter of either a pointer or a reference to an Arguments array.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values

Default CTIO S return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

The current information about the call is received in the OnCallDataUpdate event.

- The OnCallDataUpdate event is received if this request is successful.
- The OnControlFailureConf event is received if this request fails.
VB
StartRecord([reserved_args As IArguments]) As Long

Java
int StartRecord(Arguments rArgs)

.NET
CilError StartRecord(Arguments args)

Parameters
reserved_args
Not currently used, reserved for future use.

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Value
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks
Calling this method causes the CTI Server to forward the request to one or more server applications that have registered the “Cisco:CallRecording” service as described in the CTI Server Message Reference Guide (Protocol Version 14) for Cisco Unified ICM/Contact Center Enterprise & Hosted (Protocol Version 14) for Cisco Unified ICM/Contact Center Enterprise & Hosted. It fails if there is no recording server available to CTIServer.

• The OnStartRecordingConf event is received if this request is successful.

• The OnControlFailureConf event is received if this request fails.

StopRecord
The StopRecord method is used to stop recording a call.

Syntax
C++

int StopRecord()
int StopRecord(Arguments & reserved_args);

COM

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

VB
StopRecord([reserved_args As IArguments]) As Long

Java
int StopRecord(Arguments rArgs)
.NET
    CilError StopRecord(Arguments args)

Parameters
reserved_args
Not currently used, reserved for future use.

errorCode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Value
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks
Calling this method causes the CTIServer to forward the request to the server application with the SessionID received in the OnStartRecordingConf event if non-zero, or if that SessionID is zero, to one or more server applications that have registered the “Cisco:CallRecording” service as described in the CTI Server Message Reference Guide (Protocol Version 14) for Cisco Unified ICM/Contact Center Enterprise & Hosted. It fails if there is no recording server available to CTIServer.

- The OnStopRecordConf event is received if this request is successful.
- The OnControlFailureConf event is received if this request fails.

Transfer
The Transfer method transfers a call to a third party. You can call this method on either the held original call or the current consult call. If the device has only these two calls, the optional parameter is not necessary. At the end of a successful transfer, both of these calls are gone from the device. For more information, see the Conference method.

Syntax
C++
    int Transfer();
    int Transfer(Arguments& optional_args)

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

VB
    Transfer([optional_args As IArguments]) As Long

Java
    int Transfer(Arguments rArgs)

.NET
    CilError Transfer(Arguments args)
**Parameters**

**optional_args**

An optional input parameter containing a member with a string value that is the UniqueObjectID of the call that is participating in the transfer. If this argument is used, add it to the Arguments parameter with the keyword of “CallReferenceObjectID”. This is only necessary in an environment where there are multiple held calls and the request is made through the current call. If the request is made through a specific held call in this scenario, or if there are only two calls at the device, this parameter is unnecessary.

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**Remarks**

Before making this request, the original call must be in the held state and the consult call in the talking state or the request fails. Therefore, if the calls are alternated (for more information, see Alternate), they must be alternated again to return the two calls into their appropriate states.

If there are only two calls at the device, call this method using either the current or held call. For switches that allow more than two calls at a device (for example G3), make this request only through the desired held call to avoid the ambiguity caused by multiple held calls at the device. Otherwise, indicate the desired held call by using the optional parameter.

You must make the Transfer request via a call whose call status is LCS_CONNECT or LCS_HELD or it fails.

The following events are received by the transfer initiator if this request is successful:

- OnCallTransferred event
- OnCallEnd event
- OnCallEnd event
- OnAgentStateChange event
- OnTransferCallConf event

The OnControlFailureConf event is received if this request fails.
SkillGroup Object

The SkillGroup object provides developers using the CTI OS Client Interface Library with an interface to Skill Group properties and data. The SkillGroup is mainly a representation used for accessing statistics, which you can enable or disable via method calls to the SkillGroup object. The SkillGroups are accessible directly from the Session object or the Agent object.

The SkillGroup object methods can be accessed as follows:

- Via the Agent object inside the Session in Agent mode
- Via the Agent object inside the Session in Monitor mode
- In C++, Java, and .NET, via the session object inside the session in Monitor mode when the special SkillGroupStats filter is set. For more information about code examples related to the special SkillGroupStats filter, see Skill Group Statistics, on page 341 in Chapter 8.

Properties

The following table lists the available SkillGroup properties.

| Note | The data type listed for each keyword is the standardized data type discussed in the section CTI OS CIL Data Types, on page 19 in CIL Coding Conventions, on page 19. For more information about the appropriate language specific types for these keywords, see Table 5: CTI OS CIL Data Type, on page 20. |
### Table 156: SkillGroup Properties

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SkillGroupNumber</td>
<td>INT</td>
<td>The optional, user-defined number of the SkillGroup from the Peripheral.</td>
</tr>
<tr>
<td>SkillGroupID</td>
<td>STRING</td>
<td>The system-assigned identifier of the SkillGroup, if available.</td>
</tr>
<tr>
<td>SkillGroupName</td>
<td>STRING</td>
<td>The Unified ICM SkillGroupName of the SkillGroup, if available.</td>
</tr>
<tr>
<td>SkillGroupState</td>
<td>INT</td>
<td>Values representing the current state of the associated agent with respect to the indicated Agent SkillGroup.</td>
</tr>
<tr>
<td>ClassIdentifier</td>
<td>INT</td>
<td>Value represents SkillGroup class.</td>
</tr>
</tbody>
</table>

To access statistics, first use GetValue on the SkillGroup object to obtain the Statistics Arguments array, then use GetValue to obtain the desired value.

---

**Note**

Not all the statistics values listed in the above table are present in every system configuration. Whether a particular statistic value is available depends on both the protocol version of CTI Server with which CTI OS connects and on the peripheral on which the agent resides. The statistics listed in Table 157: SkillGroup Statistics, on page 424 are available in Protocol Version 8 of CTI Server.

One very important real-time skillgroup statistic is the number of calls currently in queue. Previously, this value was provided in CallsQNow. Now the number of calls currently in queue is stored in RouterCallsQNow.

### Statistics

The following table lists the available SkillGroup statistics.

### Table 157: SkillGroup Statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentsLoggedOn</td>
<td>Number of agents that are currently logged on to the SkillGroup.</td>
</tr>
<tr>
<td>AgentsAvail</td>
<td>Number of agents for the SkillGroup in Available state ready to take calls.</td>
</tr>
<tr>
<td>AgentsNotReady</td>
<td>Number of agents in the Not Ready state for the SkillGroup.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AgentsReady</td>
<td>Number of agents that are in work state (TALKING, HELD, WORK_READY, AVAILABLE, or RESERVED). This statistic is used by the router to determine the number of working agents in the SkillGroup when estimating the expected delay. It is the difference between AgentsLoggedOn and AgentsNotReady. Reference AgentsAvail to get the number of agents that are available to take calls right now.</td>
</tr>
<tr>
<td>AgentsTalkingIn</td>
<td>Number of agents in the SkillGroup currently talking on inbound calls.</td>
</tr>
<tr>
<td>AgentsTalkingOut</td>
<td>Number of agents in the SkillGroup currently talking on outbound calls.</td>
</tr>
<tr>
<td>AgentsTalkingOther</td>
<td>Number of agents in the SkillGroup currently talking on internal (not inbound or outbound) calls.</td>
</tr>
<tr>
<td>AgentsWorkNot Ready</td>
<td>Number of agents in the SkillGroup in the Work Not Ready state.</td>
</tr>
<tr>
<td>AgentsWorkReady</td>
<td>Number of agents in the SkillGroup in the Work Ready state.</td>
</tr>
<tr>
<td>AgentsBusyOther</td>
<td>Number of agents currently busy with calls assigned to other SkillGroups.</td>
</tr>
<tr>
<td>AgentsReserved</td>
<td>Number of agents for the SkillGroup currently in the Reserved state.</td>
</tr>
<tr>
<td>AgentsHold</td>
<td>Number of calls to the SkillGroup currently on hold.</td>
</tr>
<tr>
<td>AgentsICM Available</td>
<td>Number of agents in the SkillGroup currently in the ICMAvailable state.</td>
</tr>
<tr>
<td>AgentsApplication Available</td>
<td>Number of agents in the SkillGroup currently in the Application Available state.</td>
</tr>
<tr>
<td>AgentsTalkingAutoOut</td>
<td>Number of calls to the SkillGroup currently talking on AutoOut (predictive) calls.</td>
</tr>
<tr>
<td>AgentsTalking Preview</td>
<td>Number of calls to the SkillGroup currently talking on outbound Preview calls.</td>
</tr>
<tr>
<td>AgentsTalking Reservation</td>
<td>Number of calls to the SkillGroup currently talking on agent reservation calls.</td>
</tr>
<tr>
<td>RouterCallsQNow**</td>
<td>The number of calls currently queued by the CallRouter for this SkillGroup. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LongestRouterCallQNow**</td>
<td>The queue time, in seconds, of the currently Unified ICM call router queued call that has been queued to the SkillGroup the longest. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>CallsQNow*</td>
<td>The number of calls currently queued to the SkillGroup. 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 SkillGroup. 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 SkillGroup 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 SkillGroup were in the Available state.</td>
</tr>
<tr>
<td>LoggedOnTimeTo5</td>
<td>Total time, in seconds, agents in the SkillGroup were logged in.</td>
</tr>
<tr>
<td>NotReadyTimeTo5</td>
<td>Total seconds agents in the SkillGroup were in the Not Ready state.</td>
</tr>
<tr>
<td>AgentOutCallsTo5</td>
<td>Total number of completed outbound ACD calls made by agents in the SkillGroup.</td>
</tr>
<tr>
<td>AgentOutCallsTalkTimeTo5</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AgentOutCallsTimeTo5</td>
<td>Total handle time, in seconds, for completed outbound ACD calls handled by agents in the SkillGroup. 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 SkillGroup have placed on hold at least once.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeTo5</td>
<td>Total number of seconds outbound ACD calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>HandledCallsTo5</td>
<td>The number of inbound ACD calls handled by agents in the SkillGroup.</td>
</tr>
<tr>
<td>HandledCallsTalk TimeTo5</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by agents in the SkillGroup. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>HandledCallsAfter CallTimeTo5</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the SkillGroup.</td>
</tr>
<tr>
<td>HandledCallsTime To5</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the SkillGroup. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>IncomingCallsHeldTo5</td>
<td>The total number of completed inbound ACD calls agents in the SkillGroup placed on hold at least once.</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeTo5</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>InternalCallsRcvdTo5</td>
<td>Number of internal calls received by agents in the SkillGroup.</td>
</tr>
<tr>
<td>InternalCallsRcvd TimeTo5</td>
<td>Number of seconds spent on internal calls received by agents in the SkillGroup.</td>
</tr>
<tr>
<td>InternalCallsHeldTo5</td>
<td>The total number of internal calls agents in the SkillGroup placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCallsHeld TimeTo5</td>
<td>Total number of seconds completed internal calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>AutoOutCallsTo5</td>
<td>Total number of AutoOut (predictive) calls completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>AutoOutCallsTalk TimeTo5</td>
<td>Total talk time, in seconds, for completed AutoOut (predictive) calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AutoOutCallsTime To5</td>
<td>Total handle time, in seconds, for completed AutoOut (predictive) calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCallsHeld To5</td>
<td>The total number of completed AutoOut (predictive) calls that agents in the SkillGroup have placed on hold at least once.</td>
</tr>
<tr>
<td>AutoOutCallsHeld TimeTo5</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>PreviewCallsTo5</td>
<td>Total number of outbound Preview calls completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>PreviewCallsTalk TimeTo5</td>
<td>Total talk time, in seconds, for completed outbound Preview calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCallsTime To5</td>
<td>Total handle time, in seconds, for completed outbound Preview calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCallsHeld To5</td>
<td>The total number of completed outbound Preview calls that agents in the SkillGroup have placed on hold at least once.</td>
</tr>
<tr>
<td>PreviewCallsHeld TimeTo5</td>
<td>Total number of seconds outbound Preview calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>ReservationCallsTo5</td>
<td>Total number of agent reservation calls completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>ReservationCalls TalkTimeTo5</td>
<td>Total talk time, in seconds, for completed agent reservation calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ReservationCalls TimeTo5</td>
<td>Total handle time, in seconds, for completed agent reservation calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>ReservationCalls HeldTo5</td>
<td>The total number of agent reservation calls that agents in the SkillGroup have placed on hold at least once.</td>
</tr>
<tr>
<td>ReservationCalls HeldTimeTo5</td>
<td>Total number of seconds agent reservation calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>BargeInCallsTo5</td>
<td>Total number of supervisor call barge-ins completed in the SkillGroup.</td>
</tr>
<tr>
<td>InterceptCallsTo5</td>
<td>Total number of supervisor call intercepts completed in the SkillGroup.</td>
</tr>
<tr>
<td>MonitorCallsTo5</td>
<td>Total number of supervisor call monitors completed in the SkillGroup.</td>
</tr>
<tr>
<td>WhisperCallsTo5</td>
<td>Total number of supervisor call whispers completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>EmergencyCallsTo5</td>
<td>Total number of emergency calls completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>CallsQ5*</td>
<td>The number of calls queued to the SkillGroup during the current five-minute. 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 SkillGroup during the current five-minute. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>LongestCallQ5*</td>
<td>The longest queue time, in seconds, of all calls queued to the SkillGroup during the current five-minute. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>AvailTimeToHalf</td>
<td>Total seconds agents in the SkillGroup were in the Available state.</td>
</tr>
<tr>
<td>LoggedOnTimeToHalf</td>
<td>Total time, in seconds, agents in the SkillGroup were logged in.</td>
</tr>
<tr>
<td>NotReadyTimeToHalf</td>
<td>Total seconds agents in the SkillGroup were in the Not Ready state.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AgentOutCallsToHalf</td>
<td>Total number of completed outbound ACD calls made by agents in the SkillGroup.</td>
</tr>
<tr>
<td>AgentOutCallsTalk TimeToHalf</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by agents in the SkillGroup. 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 SkillGroup. 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 SkillGroup have placed on hold at least once.</td>
</tr>
<tr>
<td>AgentOutCallsHeldTimeToHalf</td>
<td>Total number of seconds outbound ACD calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>HandleCallsToHalf</td>
<td>The number of inbound ACD calls handled by agents in the SkillGroup.</td>
</tr>
<tr>
<td>HandleCallsTalk TimeToHalf</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by agents in the SkillGroup. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>HandleCallsAfter CallTimeToHalf</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the SkillGroup.</td>
</tr>
<tr>
<td>HandleCallsTime ToHalf</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the SkillGroup. 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 SkillGroup placed on hold at least once.</td>
</tr>
<tr>
<td>IncomingCallsHeldTimeToHalf</td>
<td>Total number of seconds completed inbound ACD calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>InternalCallsRcvdToHalf</td>
<td>Number of internal calls received by agents in the SkillGroup.</td>
</tr>
<tr>
<td>InternalCallsRcvd TimeToHalf</td>
<td>Number of seconds spent on internal calls received by agents in the SkillGroup.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>InternalCallsHeldToHalf</td>
<td>The total number of internal calls agents in the SkillGroup placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCallsToHalf</td>
<td>Total number of seconds completed internal calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>AutoOutCallsToHalf</td>
<td>Total number of AutoOut (predictive) calls completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>AutoOutCallsTalkToHalf</td>
<td>Total talk time, in seconds, for completed AutoOut (predictive) calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCallsTimeToHalf</td>
<td>Total handle time, in seconds, for completed AutoOut (predictive) calls handled by agents in the SkillGroup. 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>The total number of completed AutoOut (predictive) calls that agents in the SkillGroup have placed on hold at least once.</td>
</tr>
<tr>
<td>AutoOutCallsHeldTimeToHalf</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>PreviewCallsToHalf</td>
<td>Total number of outbound Preview calls completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>PreviewCallsTalkToHalf</td>
<td>Total talk time, in seconds, for completed outbound Preview calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCallsTimeToHalf</td>
<td>Total handle time, in seconds, for completed outbound Preview calls handled by agents in the SkillGroup. 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>The total number of completed outbound Preview calls that agents in the SkillGroup have placed on hold at least once.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>PreviewCallsHeld TimeToHalf</td>
<td>Total number of seconds outbound Preview calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>ReservationCallsToHalf</td>
<td>Total number of agent reservation calls completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>ReservationCalls TalkTimeToHalf</td>
<td>Total talk time, in seconds, for completed agent reservation calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>ReservationCalls TimeToHalf</td>
<td>Total handle time, in seconds, for completed agent reservation calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>ReservationCalls HeldToHalf</td>
<td>The total number of agent reservation calls that agents in the SkillGroup have placed on hold at least once.</td>
</tr>
<tr>
<td>ReservationCalls HeldTimeToHalf</td>
<td>Total number of seconds agent reservation calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>BargeInCallsToHalf</td>
<td>Total number of supervisor call barge-ins completed in the SkillGroup.</td>
</tr>
<tr>
<td>InterceptCallsTo Half</td>
<td>Total number of supervisor call intercepts completed in the SkillGroup.</td>
</tr>
<tr>
<td>MonitorCallsToHalf</td>
<td>Total number of supervisor call monitors completed in the SkillGroup.</td>
</tr>
<tr>
<td>WhisperCallsToHalf</td>
<td>Total number of supervisor call whispers completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>EmergencyCalls ToHalf</td>
<td>Total number of emergency calls completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>CallsQHalf*</td>
<td>The number of calls queued to the SkillGroup 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 SkillGroup during the current half hour. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LongestCallQHalf*</td>
<td>The longest queue time, in seconds, of all calls queued to the SkillGroup 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 SkillGroup were in the Available state.</td>
</tr>
<tr>
<td>LoggedOnTime Today</td>
<td>Total time, in seconds, agents in the SkillGroup were logged in.</td>
</tr>
<tr>
<td>NotReadyTime Today</td>
<td>Total seconds agents in the SkillGroup were in the Not Ready state.</td>
</tr>
<tr>
<td>AgentOutCalls Today</td>
<td>Total number of completed outbound ACD calls made by agents in the SkillGroup.</td>
</tr>
<tr>
<td>AgentOutCallsTalk TimeToday</td>
<td>Total talk time, in seconds, for completed outbound ACD calls handled by agents in the SkillGroup. 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 SkillGroup. 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 SkillGroup 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 SkillGroup.</td>
</tr>
<tr>
<td>HandledCallsToday</td>
<td>The number of inbound ACD calls handled by agents in the SkillGroup.</td>
</tr>
<tr>
<td>HandledCallsTalk TimeToday</td>
<td>Total talk time in seconds for Inbound ACD calls counted as handled by agents in the SkillGroup. Includes hold time associated with the call.</td>
</tr>
<tr>
<td>HandledCallsAfter CallTimeToday</td>
<td>Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the SkillGroup.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HandledCallsTime Today</td>
<td>Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the SkillGroup. 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 SkillGroup 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 SkillGroup.</td>
</tr>
<tr>
<td>InternalCallsRcvd Today</td>
<td>Number of internal calls received by agents in the SkillGroup.</td>
</tr>
<tr>
<td>InternalCallsRcvd TimeToday</td>
<td>Number of seconds spent on internal calls received by agents in the SkillGroup.</td>
</tr>
<tr>
<td>InternalCallsHeld Today</td>
<td>The total number of internal calls agents in the SkillGroup placed on hold at least once.</td>
</tr>
<tr>
<td>InternalCallsHeld TimeToday</td>
<td>Total number of seconds completed internal calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>AutoOutCallsToday</td>
<td>Total number of AutoOut (predictive) calls completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>AutoOutCallsTalk TimeToday</td>
<td>Total talk time, in seconds, for completed AutoOut (predictive) calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCallsTime Today</td>
<td>Total handle time, in seconds, for completed AutoOut (predictive) calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>AutoOutCallsHeld Today</td>
<td>The total number of completed AutoOut (predictive) calls that agents in the SkillGroup have placed on hold at least once.</td>
</tr>
<tr>
<td>AutoOutCallsHeld TimeToday</td>
<td>Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>PreviewCallsToday</td>
<td>Total number of outbound Preview calls completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>Statistic</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PreviewCallsTalk Time Today</td>
<td>Total talk time, in seconds, for completed outbound Preview calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCallsTime Today</td>
<td>Total handle time, in seconds, for completed outbound Preview calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>PreviewCallsHeld Today</td>
<td>The total number of completed outbound Preview calls that agents in the SkillGroup have placed on hold at least once.</td>
</tr>
<tr>
<td>PreviewCallsHeld Time Today</td>
<td>Total number of seconds outbound Preview calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>ReservationCalls Today</td>
<td>Total number of agent reservation calls completed by agents in the SkillGroup.</td>
</tr>
<tr>
<td>ReservationCalls TalkTime Today</td>
<td>Total talk time, in seconds, for completed agent reservation calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>ReservationCalls Time Today</td>
<td>Total handle time, in seconds, for completed agent reservation calls handled by agents in the SkillGroup. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.</td>
</tr>
<tr>
<td>ReservationCalls Held Today</td>
<td>The total number of agent reservation calls that agents in the SkillGroup have placed on hold at least once.</td>
</tr>
<tr>
<td>ReservationCalls Held Time Today</td>
<td>Total number of seconds agent reservation calls were placed on hold by agents in the SkillGroup.</td>
</tr>
<tr>
<td>BargeInCallsToday</td>
<td>Total number of supervisor call barge-ins completed in the SkillGroup.</td>
</tr>
<tr>
<td>InterceptCallsToday</td>
<td>Total number of supervisor call intercepts completed in the SkillGroup.</td>
</tr>
</tbody>
</table>
**Statistic** | **Definition**
---|---
MonitorCallsToday | Total number of supervisor call monitors completed in the SkillGroup.
WhisperCallsToday | Total number of supervisor call whispers completed by agents in the SkillGroup.
EmergencyCallsToday | Total number of emergency calls completed by agents in the SkillGroup.
CallsQToday* | The number of calls queued to the skill. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
CallsQTimeToday* | The total queue time, in seconds, of calls queued to the SkillGroup. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
LongestCallQToday* | The longest queue time, in seconds, of all calls queued to the SkillGroup. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.

* This statistic is available for TDM switches only. It is not valid for Unified CCE.
** This statistic is available for Unified CCE only or Network Queuing.

### Methods

The following table lists the SkillGroup object methods.

**Table 158: SkillGroup Object Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupStatistics</td>
<td>Disables SkillGroup statistic messages.</td>
</tr>
<tr>
<td>DumpProperties</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>EnableSkillGroupStatistics</td>
<td>Enables SkillGroup statistic messages.</td>
</tr>
<tr>
<td>GetElement</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetNumProperties</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetPropertyName</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetValue</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetValueInt (C++)</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetValueIntObj (Java)</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
<tr>
<td>GetValueString</td>
<td>For more information, see CtiOs Object, on page 291</td>
</tr>
</tbody>
</table>
DisableSkillGroupStatistics

The DisableSkillGroupStatistics method requests that real-time statistics stop being sent to the SkillGroup object.

Syntax

C++

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

COM

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

VB

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

Java

```java
int DisableSkillGroupStatistics(Arguments args)
```

.NET

```csharp
CilError DisableSkillGroupStatistics(Arguments args)
```

Parameters

args

If this method is called in C++, Java, or .NET via the session object in monitor mode with the special SkillGroupStats filter, the args parameter has two required values for PeripheralId and SkillGroupNumber. For more information about a code example, see the Remarks section. Otherwise, this parameter is not used.

errorCode

An output parameter (return parameter in VB) that contains an error code, if any.

Return Value

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

The CTI OS server sends SkillGroup statistics in an OnSkillGroupStatisticsUpdated event. If this request is successful, the OnNewSkillGroupStatistics event is no longer received.

The following is a C++ code example where the args parameter contains values for PeripheralID and SkillGroupNumber.

```cpp
Arguments & argsStatBroadcast = Arguments::CreateInstance();
argsStatBroadcast.AddItem(CTIOS_SkillGroupNUMBER, intSG);
```
DumpProperties

For more information about the DumpProperties method, see CtiOs Object, on page 291.

EnableSkillGroupStatistics

The EnableSkillGroupStatistics method requests that real-time statistics be sent to the SkillGroup object. In an agent mode application, this request is usually made through the Agent object (see Call Object, on page 389). If the argument array is empty, then statistics for all SkillGroups are enabled. This is useful when a monitoring application needs to view all statistics without having to enumerate and loop over each statistic to enable it.

Syntax

C++

int EnableSkillGroupStatistics(Arguments & args)

COM

HRESULT EnableSkillGroupStatistics (IArguments * args, int * errorCode)

VB

EnableSkillGroupStatistics (args As CTIOSCLIENTLib.IArguments, errorCode As Long)

Java

int EnableSkillGroupStatistics(Arguments args)

.NET

CilError EnableSkillGroupStatistics(Arguments args)

Parameters

args

If this method is called via the session object in monitor mode with the special SkillGroupStats filter, the args parameter has two required values for PeripheralId and SkillGroupNumber. For more information about a code example, see the Remarks section. Otherwise, this parameter is not used.

terCode

An output parameter (return parameter in VB) that contains an error code, if any.

Return Value

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

Remarks

CTI OS Server sends SkillGroup statistics in an OnSkillGroupStatisticsUpdated event.

The following is a C++ code example where the args parameter contains values for PeripheralID and SkillGroupNumber.
Arguments & argsStatBroadcast = Arguments::CreateInstance();
argsStatBroadcast.AddItem(CTIOS_SkillGroupNUMBER, intSG);
argsStatBroadcast.AddItem(CTIOS_PERIPHERALID, m_periphID);
m_pSkGrStatSession->EnableSkillGroupStatistics ( argsStatBroadcast );
argsStatBroadcast.Release();

**GetElement**

For more information about the GetElement method, see *CtiOs Object, on page 291.*

**GetValue Methods**

For more information about the GetValue, GetValueInt, GetValueList, and GetValueString methods, see *CtiOs Object, on page 291.*

**IsValid**

For more information about the IsValid method, see *CtiOs Object, on page 291.*

**SetValue**

For more information about the SetValue method, see *CtiOs Object, on page 291.*
Helper Classes

The CTI OS Client Interface Library uses several custom data structures. This chapter describes the CTI OS Helper Classes (data structures). The following helper classes are distributed with the Client Interface Library:

- **Arg.** The Arg structure is the basic data type the CIL uses for any parameter included in methods or events. Objects of this type allow the CIL to be fully extensible and reusable. Arg supports many useful types including string, integer, Boolean, and Arguments array. Arg is the base class for the Arguments class. In most programming scenarios, programmers do not use Arg directly, but indirectly through the Arguments class.

- **Arguments.** You use the Arguments structure to maintain and send a set of key-value pairs between the CIL and CTI OS Server for events and requests. The Arguments array elements must all be of type Arg. The Arguments structure enables future growth of the CTI OS feature set, without requiring changes to the method call signature.

- **CILRefArg.** The CILRefArg class is a specialized subclass of Arg. It stores a reference to an object derived from CCTiOsObject (C++ only). For instance, it can hold reference to a CAgent, CCall, CSkillGroup, CCTiOsSession, or CWaitObject.

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

- **CWaitObject.** CWaitObject is a CIL object that derives from CCTiOsObject. It is a utility class (available in all CILs except COM) that enables a thread to wait for one or more CTI events. The user can provide a list of events along with a wait timeout. You create wait objects with the CreateWaitObject Session Object method and destroy them with the DestroyWaitObject Session Object method.
The Logger class creates a Logger object and a LogManager object, if one does not already exist. Any object that needs to perform logging must instantiate the Logger class. The Logger class communicates with the singleton LogManager object, which acts as the centralized logging facility. The Logger class also defines tracing constants.

The LogWrapper class provides a default Logging mechanism. By default, the LogWrapper traces to the console. If you create the LogWrapper with a filename, then it traces to that file.

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 160: enumArgTypes, on page 446.

Arg class methods do conversion between types whenever possible. For example, you can do a SetValue(25) and then do a GetValueString(), which returns the string “25”. You can also do a SetValue(“25”) and then do a GetValueIntObj, which returns an Integer object containing the numeric value 25. However, if you call SetValue “abc” and try to retrieve it as an int, it fails.

The following table lists the available Arg class methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the data item.</td>
</tr>
<tr>
<td>Clone</td>
<td>Creates an exact copy of the Arg object.</td>
</tr>
<tr>
<td>CreateInstance</td>
<td>Creates an Arg object.</td>
</tr>
<tr>
<td>DumpArg</td>
<td>Builds a string containing the value stored in the Arg.</td>
</tr>
<tr>
<td>GetType</td>
<td>Returns the type of the data stored in the argument (one of the values in Table 160: enumArgTypes, on page 446).</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>GetValueUIntObj</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>SetValue</td>
<td>Sets the data in the Arg object.</td>
</tr>
</tbody>
</table>

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

AddRef

The AddRef method increments the reference count for the data item. It is necessary to call this if you are storing a pointer to the item for some time (for example, if you plan to store and use Arguments received in
an event handler after the event thread has returned to the calling code). When you are finished with the item, you must call the Release method or a memory leak occurs.

**Syntax**

**C++**

```cpp
unsigned long AddRef()
```

**COM**

```cpp
HRESULT AddRef()
```

**VB, Java, .NET**

Not used

**Parameters**

None.

**Return Values**

**COM:** Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.

**C++:** The current reference count after the AddRef() call.

### Clone

The Clone method allocates a new Arg in memory and copies its key, value, and type to the new instance. When using the C++ or COM CILs, it is important to release the object when it is no longer needed.

**Syntax**

**C++**

```cpp
Arg & Clone()
```

**COM**

```cpp
HRESULT Clone(/*[out, retval]*/ IArg** arg);
```

**VB**

```vb
Clone() as CTIOSCLIENTLib.IArg
```

**Java**

```java
Arg Clone()
```

**.NET**

```csharp
Object Clone()
```

**Output Parameters**

arg

Pointer to an IArg instance that is a copy of the object.

**Return Values**

**COM:** Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.
Others: If successful, returns a reference to a new Arg object. If unsuccessful in C++ or VB, it throws a
CCtiosException with iCode set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED. If unsuccessful in
Java, it returns null but does not throw an exception.

CreateInstance

The CreateInstance method creates an object of type Arg class and sets the reference count of the object to 1.
It is important to release the object when it is no longer in use in the program.

Syntax

C++

static Arg& CreateInstance();
    // static creation mechanism.
static Arg& CreateInstance(Arg& arg);
    // static creation mechanism.
static bool CreateInstance(Arg ** arg);
    // static creation mechanism,
    // alternate version

COM

Wrapped by CoCreateInstance

VB

Wrapped by New

Java, .NET

Not available

Parameters

arg
(output) Pointer to the newly created Arg.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.
Others: Either a reference to the newly created Arg or a boolean indicating method success. If the methods
not returning bool are unsuccessful, they raise a CCtiosException with iCode set to
E_CTIOS_ARGUMENT_ALLOCATION_FAILED.

Remarks

This method increments the Arg's reference count, so do not call AddRef(). However, you must call Release() after you are finished with the Arg.

DumpArg

The DumpArg method builds a string containing the value stored in the Arg. This involves doing any type
conversion required to display the data as a string. For example, it automatically converts an INTEGER type
to a string that can be logged for debugging. In the event that a Arg object is actually an Arguments object,
the string returned is the one built by Arguments.DumpArg, and thus enabled printing of nested Arguments structures.

**Syntax**

**C++**

```cpp
string DumpArg()
```

**COM**

```c
HRESULT DumpArg([out,retval] BSTR* arg_string);
```

**VB**

```vb
DumpArg() as String
```

**Java, .NET**

Not available. Use the ToString method.

**Parameters**

arg_string

The pointer to the string where the contents of the Arg object are written.

**Return Values**

COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.

Others: A string containing the contents of the structure.

---

**GetArgType (.NET Only)**

The GetArgType method returns the type of the contained value. This returned value is one of the following:

- ARG_NOTSET
- ARG_BOOL
- ARG_SHORT
- ARG_USHORT
- ARG_INT
- ARG_UINT

For more information about valid types, see Table 160: enumArgTypes, on page 446.

**Syntax**

**C++, COM, Java**

Use GetType.

**.NET**

```csharp
ArgDataType GetArgType()
```
Parameters
None.

Returns
int code for the type of value contained in this Arg.

GetType

The GetType method returns the type of the data stored by the Arg. For more information about possible types, see the following table.

Syntax

C++

    enumArgTypes GetType()

COM

    HRESULT GetType(/*[out, retval]*/ int* type);

VB

    GetType () as Integer

Java

    int GetType;

.NET

    Use the GetArgType method.

Output Parameters

type
Integer that receives the enumerated constant that identifies data type stored in IArg.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.

Others: Returns the enumerated value that identifies the data type stored in the Arg (for more information, see the following table).

Table 160: 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>
</tbody>
</table>
GetValue Methods

The GetValue method returns the value stored in the object. To extract a specific type of data you invoke the method designated for it. For more information about GetValueArray, GetValueInt, and GetValueString, see the corresponding methods described in CtiOs Object, on page 291

Syntax

C++

```cpp
int GetValueInt();
unsigned int GetValueUInt();
unsigned short GetValueUShort();
short GetValueShort();
string& GetValueString();
bool GetValueBool();
bool GetValueInt(int * value);
bool GetValueUInt(unsigned int * value);
bool GetValueUShort(unsigned short * value);
bool GetValueShort( short * psValue);
bool GetValueBool( bool * value);
bool GetValueString(string* value);
```

COM

```cpp
HRESULT GetValue(/*[out, retval]*/ VARIANT* value);
```

VB

```vbnet
GetValue() as Variant
GetValue (key as String, value as Variant) as Boolean
```

Java

```java
Integer GetValueIntObj()
Long GetValueUIntObj()
Short GetValueShortObj()
Integer GetValueUShortObj()
Boolean GetValueBoolObj()
String GetValueString()
```

.NET

```csharp
System.Boolean GetValueInt(out System.Int32 nValue)
System.Boolean GetValueUInt(out System.Int64 nValue)
System.Boolean GetValueShort(out System.Int16 nValue)
System.Boolean GetValueUShort(out System.Int32 nValue)
System.Boolean GetValueBool(out System.Boolean bValue)
System.Boolean GetValueString(out System.String strValue)
```
**Parameters**

**Value**

Output parameter of the specified type containing the value of the Arg.

For COM, this value is of type VARIANT * whose type is one of the types listed in the following table.

*Table 161: Variant Types Supported by GetValue (COM)*

<table>
<thead>
<tr>
<th>Variant Type</th>
<th>Standard C++ Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT_INT</td>
<td>Int</td>
</tr>
<tr>
<td>VT_UINT</td>
<td>Unsigned int</td>
</tr>
<tr>
<td>VT_I2</td>
<td>Short</td>
</tr>
<tr>
<td>VT_U12</td>
<td>Unsigned short</td>
</tr>
<tr>
<td>VT_BOOL</td>
<td>Bool</td>
</tr>
<tr>
<td>VT_BSTR</td>
<td>string, const string and char *</td>
</tr>
</tbody>
</table>

**Return Values**

**C++**

Methods taking no parameters, if successful, return the value in the object; otherwise, they raise a CCtiosException with iCode set to E_CTIOS_INVALID_ARGUMENT.

The methods taking a pointer to the variable receiving the result return true, if the method was able to get the value, otherwise, false.

**Java**

Returns null if method failed.

**.NET**

Returns false if method failed.

**COM**

If the method was able to set the variant type of the value to any of the types listed in the above table, it returns the value in the appropriate field of the variant. Otherwise it returns VT_EMPTY.

**Release**

The Release method decrements the reference count for the data item. It is necessary to call Release when you are finished with a data item that had its reference count incremented via CreateInstance or AddRef; otherwise, a memory leak occurs.

**Syntax**

**C++**

```cpp
unsigned long Release()
```
COM
HRESULT Release()

VB, Java, .NET
Not used

Parameters
None.

Return Values
COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.
C++: The current reference count after the Release() call.

SetValue

The SetValue method sets the value in the Arg object.

Syntax

C++

    bool SetValue( int value );
    bool SetValue( unsigned int value );
    bool SetValue( unsigned short value );
    bool SetValue( short value );
    bool SetValue( bool value );
    bool SetValue( char * value );
    bool SetValue( string& value);
    bool SetValue( const string& value);
    bool SetValue( Arg & value);

COM

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

VB

    SetValue(value as Variant) as Boolean

Java

    boolean SetValue(Arg rArg)
    boolean SetValue(int iVal)
    boolean SetValue(short nValue)
    boolean SetValue(String sValue)
    boolean SetValueUInt(long lValue)
    boolean SetValueUShort(int iValue)

.NET

    System.Boolean SetValue(System.Int32 iValue)
    System.Boolean SetValueUInt(System.Int64 lValue)
    System.Boolean SetValueUShort(System.Int32 iValue)
    System.Boolean SetValue(System.Int16 nValue)
    System.Boolean SetValue(System.Boolean bValue)
    System.Boolean SetValue(System.String sValue)
    System.Boolean SetValue(Arg rArg)
Parameters

value

The value of the specified type to assign to the Arg.

For COM, this value is of type VARIANT * whose type is one of the types listed in the following table.

Table 162: Supported Variant Types

<table>
<thead>
<tr>
<th>Variant Type</th>
<th>Standard C++ Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT_INT</td>
<td>Int</td>
</tr>
<tr>
<td>VT_UINT</td>
<td>Unsigned int</td>
</tr>
<tr>
<td>VT_I2</td>
<td>Short</td>
</tr>
<tr>
<td>VT_UI2</td>
<td>Unsigned short</td>
</tr>
<tr>
<td>VT_BOOL</td>
<td>Bool</td>
</tr>
<tr>
<td>VT_BSTR</td>
<td>string, const string and char *</td>
</tr>
<tr>
<td>VT_DISPATCH</td>
<td>Pointer to an IArg interface</td>
</tr>
</tbody>
</table>

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values

C++

If the method was able to set the value it returns true, otherwise it returns false.

COM, VB

If the method was able to set the value it returns VARIANT_TRUE. Otherwise, it returns VARIANT_FALSE.

Java, .NET

This method returns true if the method succeeds, otherwise false.

Arguments Class

The Arguments structure (class) provides key/value support to form a collection of values. Each value stored in an Arguments structure is associated with a key. To add an item, use the AddItem or SetValue method and pass a key and a value. The key must be a string or an enumerated value, and the value can be almost any type (i.e. all types supported by Arg). To retrieve the item, use the appropriate GetValue method with a key, and the value is returned. Keys are not case sensitive, and leading and trailing spaces are always removed from the key.
Arguments also supports access by index. The index is useful for retrieving items sequentially, but may not be as fast as retrieval by key. The Arguments structure's index is 1-based, to provide easier support for Visual Basic programmers. Internally, the Arguments structure uses a binary tree and other techniques to provide fast access to any item. Arguments can support a virtually unlimited number of key-value pairs, and supports nested Arguments structure as well.

The following table lists the Arguments class methods.

**Table 163: Arguments Class Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddItem</td>
<td>Adds an item to an Arguments array.</td>
</tr>
<tr>
<td>AddRef</td>
<td>Increments the reference count for the data item.</td>
</tr>
<tr>
<td>Clear</td>
<td>Deletes all elements from an Arguments array.</td>
</tr>
<tr>
<td>Clone</td>
<td>Creates a copy of an Arguments array.</td>
</tr>
<tr>
<td>CreateInstance</td>
<td>Creates an Arguments array.</td>
</tr>
<tr>
<td>DumpArgs</td>
<td>Returns Arguments object as a string</td>
</tr>
<tr>
<td>GetElement (also GetElementInt, GetElementUInt, GetElementUShort, GetElementShort, GetElementBool, GetElementString, GetElementArg, GetElementKey GetElementArgType)</td>
<td>Returns the value stored under a specified index.</td>
</tr>
<tr>
<td>GetValue (also GetValueInt, GetValueUShort, GetValueShort, GetValueBool, GetValueUInt, GetValueString, GetValueArray, GetValueArg)</td>
<td>Returns the value stored under a specified key.</td>
</tr>
<tr>
<td>IsValid</td>
<td>Tests if a key is present in the current Arguments array.</td>
</tr>
<tr>
<td>NumElements</td>
<td>Returns the number of arguments in the current Arguments array.</td>
</tr>
<tr>
<td>Release</td>
<td>Decrements the reference count for the data item.</td>
</tr>
<tr>
<td>RemoveItem</td>
<td>Removes an item from an Arguments array.</td>
</tr>
<tr>
<td>SetElement</td>
<td>Sets the value of an index.</td>
</tr>
<tr>
<td>SetValue</td>
<td>Sets the value of a key.</td>
</tr>
</tbody>
</table>

**Usage Notes**

When writing an application using the CTI OS SDK, the following sequence of steps in the program can produce a problem:

- Programmer passes an Arguments array into a CTI OS SDK method (methodA)
- MethodA returns
• Programmer modifies the same Arguments array

• Programmer passes the modified Arguments array into another CTI OS SDK method (methodB)

When running the application, the call to methodA can behave as if it was passed the modified Arguments array. This is because many CTI OS methods simply place a pointer to the Arguments array on a queue of items to send to CTI OS server. When the same Arguments array is later modified, as in the preceding example, the pointer on the queue now points to the modified array and the modified array is sent to CTI OS server. A problem can occur depending on timing, as there are multiple threads involved: the thread pulling items off the queue and the thread modifying the Arguments array. If the queued message is sent to CTI OS before the Arguments array is modified, the problem does not occur.

To avoid this problem, call the Clone method on the initial Arguments array and modify the copy rather than modifying the original. For example, the preceding example would change as follows:

• Programmer passes an Arguments array (initialArray) into a CTI OS SDK method (methodA)

• MethodA returns

• modifiedArray = initialArray.Clone()

• Programmer modifies modifiedArray

• Programmer passes the modifiedArray into another CTI OS SDK method (methodB)

AddItem (C++ COM VB Only)

The AddItem method expects a key-value pair. The key value can be a string or an integer. The value can be a string, an integer, or an object reference. If there is an entry with the same key, it is replaced with this entry, otherwise the new key-value pair is added to the Arguments array. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Syntax

C++

```cpp
bool AddItem( std::string & key, int value );
bool AddItem( std::string & key, unsigned int value );
bool AddItem( std::string & key, unsigned short value );
bool AddItem( std::string & key, short value );
bool AddItem( std::string & key, bool value );
bool AddItem( std::string & key, char * pchar );
bool AddItem( char * key, int value );
bool AddItem( char * key, unsigned int value );
bool AddItem( char * key, unsigned short value );
bool AddItem( char * key, short value );
bool AddItem( char * key, bool value );
bool AddItem( char * key, char * value );
bool AddItem( char * key, std::string & value );
bool AddItem( char * key, Arg & cArg );
```
bool AddItem( char * key, const Arg& value );
bool AddItem( char * key, Arguments& value );
bool AddItem( char * key, const Arguments& value );

bool AddItem( enum_Keywords key, int value );
bool AddItem( enum_Keywords key, unsigned int value );
bool AddItem( enum_Keywords key, unsigned short value );
bool AddItem( enum_Keywords key, short value );
bool AddItem( enum_Keywords key, bool value );
bool AddItem( enum_Keywords key, char * value );
bool AddItem( enum_Keywords key, std::string& value );
bool AddItem( enum_Keywords key, Args cArg );
bool AddItem( enum_Keywords key, const Args& value );
bool AddItem( enum_Keywords key, const Arguments& value );

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

VB
AddItem( Key as Variant, Value as Variant)

Java, .NET
Not Applicable. Use the SetValue method.

Parameters
key
Key name for the item to be added.
value
Value of the item to be added.
success
An output parameter (return parameter in C++ and VB) that contains a boolean indicating success or lack thereof.

Return Value
C++: Returns True in if the entry was successfully added, otherwise False.
COM and VB: Standard return values are valid; For more information, see CIL Coding Conventions, on page 19.

AddRef (C++ and COM Only)
The AddRef method increments the reference count for the data item. You must call this if you are storing a pointer to the item for some time. When you are finished with the item, you must call the Release method or a memory leak occurs.

Syntax
C++
unsigned long AddRef()
COM

HRESULT AddRef()

VB, Java, .NET

Not used

Parameters

None.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.

C++: Current reference count.

Others: None.

Clear

The Clear method deletes all the elements from Arguments object.

Syntax

C++

void Clear()

COM

HRESULT Clear()

VB

Clear()

Java, .NET

void Clear()

Parameters

None.

Return Value

None.

Clone

The Clone method creates a copy of the Arguments structure. Because in C++ this method is implemented in the base class (Arg), it returns a reference to an Arg, but this is actually a reference to an Arguments array. Therefore, it is necessary to cast the return value of this method. The following C++ code sample shows this casting:

```cpp
Arguments & argsCopy = (Arguments&) argsOrig.Clone();
```

To cast in VB, do the following:
Dim Args As CTIOSCLIENTLib.IArgumentsSet Args = Orig.Clone()

Syntax

C++
    Arg & Clone()

COM
    HRESULT Clone(/*[out, retval]*/ IArguments ** args);

VB
    Clone() as CTIOSCLIENTLib.IArguments

Java
    Arg Clone()

.NET
    object Clone()

Parameters

args
    An output parameter containing a pointer to an Arguments array that is a copy of the object.

Return Value

COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.
Others: A reference to the Arg structure that is a copy of the object.

CreateInstance (C++ and COM Only)

The CreateInstance method creates an object of type Arguments class and sets the reference count of the object to 1. It is important to release the object when it is no longer in use in the program.

Syntax

C++
    static Arguments & CreateInstance()
    static bool CreateInstance(Arguments ** args)

COM
    Not exposed, called by CoCreateInstance.

VB
    Not exposed, called by New.

Java, .NET
    Not implemented.

Parameters

args
    A pointer to the newly created Arguments structure.
Return Value

COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.
Others: Either a reference to the newly created Arguments structure or a boolean indicating method success.

Remarks

C++, COM: Internally this method increments the Arg's reference count, so do not call AddRef(). However, you must call Release() when you are finished with the Arg.

DumpArgs

The DumpArgs method builds a string showing all of the members of the Arguments structure in the form “key1=value1; key2=value2;...”. It is primarily used for debugging.

Note

The trace mask must be set in order to run the DumpArgs method on the Arguments array of call variables. For information on setting the trace level see, Set Trace Levels (COM and C++), on page 504.

Syntax

C++

    string DumpArgs()

COM

    HRESULT DumpArgs([out,retval] BSTR* arg_string);

VB

    DumpArgs() as String

Java, .NET

    string DumpArgs()

Parameters

arg_string

The pointer to the string containing the contents of the Arguments array listing all of the key/value pairs in the format of “key1=value1; key2=value2;...”.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.
Others: A string containing the contents of the Arguments array listing all key/value pairs.

GetElement Methods

The GetElement method is similar to GetValue, except that it uses an index value instead of a key. The index value is not related to the order in which items are added or removed. The order of items in Arguments is never guaranteed. This method is useful for sequentially iterating over all items in Arguments. The Index is
I-based. The Index should never be less than one or greater than NumElements. see also NumElements method. The GetElementKey returns the key of a given index.

**Syntax**

**C++**

```cpp
Arg& GetElement( int index );
bool GetElement( int index, Arg ** value);
int GetElementInt( int index );
bool GetElementInt( int index, int * value );
bool GetElementUInt( int index, unsigned int * value );
unsigned short GetElementUShort( int index );
bool GetElementUShort( int index, unsigned short * value);
short GetElementShort( int index );
bool GetElementShort( int index, short * value);
bool GetElementBool( int index );
bool GetElementBool( int index, bool * value);
std::string GetElementString( int index );
bool GetElementString( int index, std::string * value );
Arguments & GetElementArg( int index );
std::string GetElementKey( int index );
bool GetElementKey( int nIndex, std::string * key);
bool GetElementKey( int nIndex, int * key);
```

**COM**

```csharp
HRESULT GetElementKey( /*[in]*/ int index, /*[out]*/ BSTR * key);
HRESULT GetElement( /*[in]*/ int index, /*[out]*/ VARIANT * value );
```

**VB**

```vb
GetElement (Integer index, Variant value)
GetElement (Integer index, String key)
```

**Java**

```java
Arg GetElement(int iIndex)
Arguments GetElementArguments(int iIndex)
Integer GetElementIntObj(int iIndex)
Long GetElementUIntObj(int iIndex)
Short GetElementShortObj(int iIndex)
Integer GetElementUShortObj(int iIndex)
Boolean GetElementBoolObj(int iIndex)
String GetElementString(int iIndex)
String GetElementKey(int iIndex)
```

**.NET**

```csharp
Boolean GetElement(System.Int32 iIndex, out Arg obArg)
Boolean GetElementInt(System.Int32 iIndex, out System.Int32 iValue)
Boolean GetElementUInt(System.Int32 iIndex, out System.Int64 nValue)
Boolean GetElementUShort(System.Int32 iIndex, out System.Int32 nValue)
Boolean GetElementShort(System.Int32 iIndex, out System.Int16 nValue)
Boolean GetElementBool(System.Int32 iIndex, out System.Boolean bValue)
Boolean GetElementString(System.Int32 iIndex, out System.String
```
GetValue Methods

The GetValue method returns the value stored under a key. You can test the existence of a key using IsValid. Keys are not case sensitive. Leading and trailing spaces are always removed from the key. For more information about GetValueArray, GetValueInt, and GetValueString, see the corresponding methods described in CtiOs Object, on page 291.

Syntax

C++

```cpp
Arg& GetValue( enum_Keywords eKey );
bool GetValue( enum_Keywords key, Arg ** value );
Arg& GetValue( std::string& key );
bool GetValue( std::string& key, Arg ** value );
Arg& GetValueArg( std::string& key );
bool GetValueArg( std::string& key, Arg ** value );
int GetValueInt( enum_Keywords key ); /*throws exception*/
bool GetValueInt( enum_Keywords key, int * value );
unsigned int GetValueUInt( enum_Keywords key );
bool GetValueUInt( enum_Keywords key, unsigned int * value );
unsigned short GetValueUShort( enum_Keywords key );
bool GetValueUShort( enum_Keywords key, unsigned short * value );
short GetValueShort( enum_Keywords key );
bool GetValueShort( enum_Keywords key, short * value );
bool GetValueBool( enum_Keywords key );
bool GetValueBool( enum_Keywords key, bool * value );
std::string GetValueString( enum_Keywords key );
bool GetValueString( enum_Keywords key, std::string * value );
int GetValueInt( std::string& key ); /*throws exception*/
bool GetValueInt( std::string& key, int * value );
unsigned int GetValueUInt( std::string& key );
bool GetValueUInt( std::string& key, unsigned int * value );
unsigned short GetValueUShort( std::string& key );
bool GetValueUShort( std::string& key, unsigned short * value );
short GetValueShort( std::string& key );
bool GetValueShort( std::string& key, short * value );
bool GetValueBool( std::string& key );
bool GetValueBool( std::string& key, bool * value );
std::string GetValueString( std::string& key );
bool GetValueString( std::string& key, std::string * value );
int GetValueInt( std::string& key, std::string * value );
```
GetValue Methods

bool GetValueUShort( std::string& key , unsigned short * value);
short GetValueShort( std::string& key , short * value);
bool GetValueBool( std::string& key );
bool GetValueBool( std::string& key , bool * value);
std::string GetValueString( std::string& key );
bool GetValueString( std::string& key , std::string * value);
Arguments& GetValueArray( std::string& key );
bool GetValueArray( std::string& key , Arguments ** value);
Arguments& GetValueArray( enum_Keywords key );
bool GetValueArray( enum_Keywords key , Arguments ** value);
bool GetValue( char * key );
bool GetValue( char * key , Arg ** value);
Arguments& GetValueArray( char * key );
bool GetValueArray( char * key , Arguments ** value);
int GetValueInt( char * key );
bool GetValueInt( char * key , int * value);
unsigned int GetValueUInt( char * key );
bool GetValueUInt( char * key , unsigned int * value);
unsigned short GetValueUShort( char * key );
short GetValueShort( char * key , short * value);
bool GetValueBool( char * key );
bool GetValueBool( char * key , bool * value);
std::string GetValueString( char * key );
bool GetValueString( char * key , std::string * value);
Arg& GetValueArg( char * key );
bool GetValueArg( char * key , Arg ** value);
Arguments& GetValueArray( char * key );
bool GetValueArray( char * key , Arguments ** value);
int GetValueInt( char * key );
bool GetValueInt( char * key , int * value);
unsigned int GetValueUInt( char * key );
bool GetValueUInt( char * key , unsigned int * value);
unsigned short GetValueUShort( char * key );
short GetValueShort( char * key , short * value);
bool GetValueBool( char * key );
bool GetValueBool( char * key , bool * value);
std::string GetValueString( char * key );
bool GetValueString( char * key , std::string * value);
Arg& GetValueArg( char * key );
bool GetValueArg( char * key , Arg ** value);

COM

HRESULT GetValue(/*[in]*/ BSTR key, /*[out, retval]*/ VARIANT * pVvalue);
HRESULT GetValueInt(/*[in]*/ VARIANT *key, /*[out, retval]*/ int *value);
HRESULT GetValueString(/*[in]*/ VARIANT *key, /*[out, retval]*/ BSTR *value);
HRESULT GetValueArray(/*[in]*/ VARIANT *key, /*[out, retval]*/ IArguments **pArguments);
HRESULT GetValueBool(/*[in]*/ VARIANT *key, /*[out, retval]*/ VARIANT_BOOL * value);

VB

GetValue (Key as String) as Variant
GetValue(key As Variant) As Arg
GetValueArray(key As Variant) As Arguments
GetValueBool(key As Variant) As Boolean
GetValueInt(key As Variant) As Long
GetValueString(key As Variant) As String

Java

Arg GetValue(int iKey)
Arg GetValue(String sKey)
Arguments GetValueArray(int iKey)
Arguments GetValueArray(String sKey)
Integer GetValueIntObj(int iKey)
Integer GetValueIntObj(String sKey)
Long GetValueUIntObj(int iKey)
Long GetValueUIntObj(String sKey)
Short GetValueShortObj(int iKey)
Short GetValueShortObj(String sKey)
Integer GetValueUShortObj(int iKey)
Integer GetValueUShortObj(String sKey)
Boolean GetValueBoolObj(int iKey)
Boolean GetValueBoolObj(String sKey)
String GetValueString(int iKey)
String GetValueString(String sKey)
.NET

Boolean GetValue(System.String sKey, out Arg obArg)
Boolean GetValueInt(System.String sKey, out System.Int32 nValue)
Boolean GetValueShort(System.String sKey, out System.Int16 nValue)
Boolean GetValueUShort(System.String sKey, out System.Int32 nValue)
Boolean GetValueBool(System.String sKey, out System.Boolean bValue)
Boolean GetValueString(System.String sKey, out System.String strValue)
Boolean GetValueArray(System.String sKey, out Arguments arArguments)
Boolean GetValue(Enum_CtiOs eKey, out Arg obArg)
Boolean GetValueInt(Enum_CtiOs eKey, out System.Int32 nValue)
Boolean GetValueShort(Enum_CtiOs eKey, out System.Int16 nValue)
Boolean GetValueUShort(Enum_CtiOs eKey, out System.Int32 nValue)
Boolean GetValueBool(Enum_CtiOs eKey, out System.Boolean bValue)
Boolean GetValueString(Enum_CtiOs eKey, out System.String strValue)
Boolean GetValueArray(Enum_CtiOs eKey, out Arguments arArguments)

Parameters

An enumerated keyword (for more information, see CTI OS Keywords and Enumerated Types, on page 499) or a string specifying the keyword of the value to retrieve.

Return Values

In C++, the two-parameter version returns a boolean indicating success or failure. The one-parameter version returns the value and throws an exception upon failure.

COM returns an HRESULT. For more information, see CIL Coding Conventions, on page 19.

Java methods return a null object if the method fails.

Remarks

Visual Basic's Integer type is a 16-bit integer. However, the GetValueInt method returns a 32-bit integer. Thus, in Visual Basic the return type for GetValueInt is actually a Visual Basic type Long. Visual Basic Programmers can use the GetValueInt method and receive the return value as an Integer, and Visual Basic will perform an implicit cast. However, if the value retrieved is a 32-bit integer, an overflow error occurs. To resolve this error, use a 32-bit integer (Long).

Those methods that do not return a bool indicating success or failure throw a CtiiosException if the method fails. The most common reasons for failure are NULL key or element with specified key not found.

IsValid

The IsValid method returns True if the specified key exists in the current Arguments array, otherwise it returns False.

Syntax

C++

bool IsValid( std::string& key );
bool IsValid( char * key );
bool IsValid( Arg& arg );
bool IsValid( enum_Keywords key );

COM

HRESULT IsValid( /*[in]*/ VARIANT* key, /*[out, retval]*/ VARIANT_BOOL* bIsvalid);
 VB
  IsValid (key as string) as Boolean

 Java, .NET
  boolean IsValid(int key)
  boolean IsValid(String key)
  boolean  IsValid(Arg rArg)

 Parameters
 key/arg
 Either the key of the desired Arguments member or an Arg containing a string key.
 C++ and COM allow you to specify the key as a string or enumerated (for more information, see CTI OS
 Keywords and Enumerated Types, on page 499); all others expect the key as a string.

 Return Values
 COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.
 Others: True if key exists in the current Arguments array, otherwise False.

 NumElements
 The NumElements method returns the number of elements stored in the current Arguments array. This method
 is useful in combination with GetElement to implement a “for” loop to iterate over all values of an Arguments
 array without knowing the keywords (you can retrieve those at the same time using GetElementKey).

 Syntax
 C++
  int NumElements();

 COM
  HRESULT NumElements(/*[out, retval]*/ int * num_elements);

 VB
  NumElements as Integer

 Java
  int NumElements()

 .NET
  int NumElements()

 Parameters
 num_elements
 Pointer to an integer value containing the number of elements in the Arguments array.

 Return Value
 COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.
 Others: Number of elements in Arguments array.
Release (C++ and COM Only)

The Release method decrements the reference count for the data item. You must call Release when you are finished with a data item that has had its reference count incremented via CreateInstance or AddRef; otherwise, a memory leak occurs.

Syntax

C++
unsigned long Release()

COM
HRESULT Release()

VB, Java, .NET
Not used

Parameters

None.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.
C++: The current reference count after the Release() call.

RemoveItem

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

Syntax

C++
bool RemoveItem( std::string& key );
bool RemoveItem( char * key );
bool RemoveItem( enum_Keywords key );

COM
HRESULT RemoveItem(/*[in]*/ VARIANT* key, /*[out, retval]*/ VARIANT_BOOL* bSuccess);

VB
RemoveItem ( key as Variant) as Boolean

Java
boolean RemoveItem(int key)
boolean RemoveItem(String key)

Parameters

key

The key to use to locate and remove the item in the Arguments array. Leading and trailing spaces are always removed from the key.
Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.

Others: Returns true if the entry was located and removed.

SetElement (C++ COM and VB Only)

The SetElement method is identical to SetValue (which is similar to AddItem), except that it uses an index value instead of a key.

Syntax

C++

bool SetElement( int index, int value );
bool SetElement( int index, unsigned int value );
bool SetElement( int index, unsigned short value );
bool SetElement( int index, short value );
bool SetElement( int index, bool value );
bool SetElement( int index, std::string & value );
bool SetElement( int index, char * pchar );
bool SetElement( int index, Arg & value );
bool SetElement( int index, Arguments & value );

COM

HRESULT SetElement(/*[in]*/ int index, /*[in]*/ VARIANT * value, /*[out,retval]*/ success);

VB

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

Java

Not available.

.NET

Not available.

Parameters

index

The index at which the value is to be set. This index value is not related to the order in which items are added or removed. The order of items in Arguments is never guaranteed. This method is useful for sequentially iterating over all items in Arguments. Index is 1-based. Index should never be less than 1 or greater than NumElements (see above). C++ implements several overloaded methods for different value types, while COM and VB use Variants.

value

The associated value to be set in the element at the designated index.

success

Output parameter (return parameter in C++ and VB) containing a boolean indicating success or failure.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.
Others: A boolean indicating success or failure.

## SetValue

The `SetValue` method sets a value for a key. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

### Syntax

#### C++

```cpp
bool SetValue( std::string& key, int value );
bool SetValue( std::string& key, unsigned int value );
bool SetValue( std::string& key, unsigned short value );
bool SetValue( std::string& key, short value );
bool SetValue( std::string& key, bool value );
bool SetValue( std::string& key, std::string& value );
bool SetValue( std::string& key, char * pchar );
bool SetValue( std::string& key, Arg& value );
bool SetValue( std::string& key, Arguments& value );
bool SetValue( std::string& key, const Arguments& value);
bool SetValue( char * key, int value );
bool SetValue( char * key, unsigned int value );
bool SetValue( char * key, unsigned short value );
bool SetValue( char * key, short value );
bool SetValue( char * key, bool value );
bool SetValue( char * key, std::string& value );
bool SetValue( char * key, char * value );
bool.SetValue( char * key, Arg& value );
bool SetValue( char * key, Arguments& value );
bool SetValue( char * key, const Arguments& value);
bool SetValue( enum_Keywords key, int value );
bool SetValue( enum_Keywords key, unsigned int value );
boolSetValue( enum_Keywords key, unsigned short value );
bool SetValue( enum_Keywords key, short value );
bool SetValue( enum_Keywords key, bool value );
bool SetValue( enum_Keywords key, std::string& value );
bool SetValue( enum_Keywords key, char* key, char* value );
```

#### COM

```com
HRESULT SetValue(/*[in]*/ VARIANT* key, /*[in]*/ VARIANT* value,/*[out, retval]*/ VARIANT_BOOL* success);
```

#### VB

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

#### Java

```java
boolean SetValue(Arguments rArguments)
boolean SetValue(int iKey, Arg rArg)
boolean SetValue(String sKey, Arg rArg)
boolean SetValue(int iKey, int iVal)
boolean SetValue(String sKey, int iVal)
boolean SetValue(int iKey, short nValue)
boolean SetValue(String sKey, short nValue)
boolean SetValue(int iKey, String sValue)
boolean SetValue(String sKey, String sValue)
```
boolean SetValueUInt(int iKey, long lValue)
boolean SetValueUInt(String sKey, long lValue)
boolean SetValueUShort(int iKey, int iValue)
boolean SetValueUShort(String sKey, int iValue)
boolean SetValue(int iKey, Arg rArg)

.NET

System.Boolean SetValueArguments(Arguments rArguments)
System.Boolean SetValue(System.String sKey, System.Int32 iValue)
System.Boolean SetValueUInt(System.String sKey, System.Int64 lValue)
System.Boolean SetValue(System.String sKey, System.Int16 nValue)
System.Boolean SetValueUShort(System.String sKey, System.Int32 iValue)
System.Boolean SetValue(System.String sKey, System.String sValue)
System.Boolean SetValue(System.String sKey, System.String sValue)
System.Boolean SetValue(System.String sKey, Arg rArg)
System.Boolean SetValue(Enum_CtiOs eKey, System.Int32 iValue)
System.Boolean SetValueUInt(Enum_CtiOs eKey, System.Int64 lValue)
System.Boolean SetValue(Enum_CtiOs eKey, System.Int16 nValue)
System.Boolean SetValueUShort(Enum_CtiOs eKey, System.Int32 iValue)
System.Boolean SetValue(Enum_CtiOs eKey, System.Boolean bValue)
System.Boolean SetValue(Enum_CtiOs eKey, System.String sValue)
System.Boolean SetValue(Enum_CtiOs eKey, Arg rArg)

Parameters

key
The key whose value is to be set.

value
The value to use in setting the element with the designated key.

success
Output parameter (return parameter in C++ and VB) containing a boolean indicating success or failure.

Return Values

COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.
Others: A boolean indicating success or failure.

Remarks

The C++ methods overload several implementations for different value types and allow you to specify a key via enumerated keywords (for more information, see CTI OS Keywords and Enumerated Types, on page 499) as well as string. COM and VB use String keywords and Variants as values.

CILRefArg Class (C++ Java and .NET Only)

The CILRefArg class is a subclass of the Arg class. Its main responsibility is to store a reference of a CCtiOsObject object (for more information, see CtiOs Object, on page 291). This class includes object references
in argument structure. The object types you can use are any of the following: CAgent, CCall, CSkillGroup, CWaitObject, or CCtiOsSession.

In addition to the methods inherited from the Arg class, the CILRefArg class contains the methods listed in the following table.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetType</td>
<td>Returns the ARG_REFERENCE.</td>
</tr>
<tr>
<td>GetUniqueObjectID</td>
<td>Returns the UID of the contained CtiOsObject.</td>
</tr>
<tr>
<td>GetValue</td>
<td>Returns the encapsulated pointer in the object.</td>
</tr>
<tr>
<td>SetValue</td>
<td>Encapsulates the pointer to CTI OS object into the CILRefArg object.</td>
</tr>
</tbody>
</table>

**GetType**

The GetType method returns the type of the data stored by the Arg. For a CilRefArg, this is always ARG_REFERENCE.

**Syntax**

**C++**

```cpp
enumArgTypes GetType()
```

**COM**

```c
HRESULT GetType(/*[out, retval]*/ int* type);
```

**VB**

```vb
GetType () as Integer
```

**Java**

```java
int GetType()
```

**.NET**

Use the GetArgType method.

**Output Parameters**

`type`

Integer that receives the enumerated constant that identifies the data type stored in Arg. In this case, that data type is ARG_REFERENCE.

**Return Values**

COM: Default HRESULT return values. For more information, see CIL Coding Conventions, on page 19.

Others: Returns the enumerated value that identifies the data type stored in the Arg (for more information, see Table 160: enumArgTypes, on page 446). For CilRefArg, this is always ARG_REFERENCE.
GetUniqueObjectID (Java and .NET Only)

The GetUniqueObjectID method returns the unique objectID of the contained CtiOsObject.

Syntax

```java
String GetUniqueObjectID()
```

Parameters
None.

Return Values
If successful, it returns the unique objectID of the contained CtiOsObject. If no object is contained in the CilRefArg, it returns null.

Remarks
To obtain a unique object ID in C++, use bool GetValueString(string* pString).

GetValue

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

Syntax

C++

```cpp
CCtiOsObject * GetValue()
```

Java

```java
CtiOsObject GetValue();
```

.NET

```csharp
System.Boolean GetValue(out CtiOsObject sValue)
```

Output Parameters

.NET:sValue
Reference to the contained CtiOsObject derived class.

Return Values

C++: Returns NULL on failure.

.NET: Returns false if the method fails.

Java: Returns a null reference if the method fails.

SetValue

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

```cpp
bool SetValue(CCTiOsObject * pObject);
```

**Input Parameters**

- **pObject**
  
  A pointer to a CtiOsObject to encapsulate (e.g. CCall, CAgent, etc.).

**Return Values**

If the method was able to set the reference it returns true. Otherwise, it returns false.

---

**CCTiOsException Class (C++ Java and .NET Only)**

The CCTiOsException class is normally used within the Arguments class. It provides access to additional information when exceptions are thrown, such as what parameter is in error, memory allocation failed, and so on.

The following table lists the available CCTiOsException class methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCTiOsException</td>
<td>Class constructor.</td>
</tr>
<tr>
<td>getCode</td>
<td>Returns the error code that generated the exception.</td>
</tr>
<tr>
<td>getStatus</td>
<td>Returns the error status that generated the exception.</td>
</tr>
<tr>
<td>getString</td>
<td>Returns a text string containing the description of the exception.</td>
</tr>
<tr>
<td>what</td>
<td>Returns a text string containing the description of the exception, the code of an error and the status.</td>
</tr>
</tbody>
</table>

---

**CCTiOsException Constructor**

The CCTiOsException constructor initializes an object of type CCTiOsException.

**Syntax**

C++, Java, .NET

```cpp
CCTiOsException(const char *pMsg, int iCode, int iStatus);
```

C++

```cpp
CCTiOsException(const string& rstrMsg, int iCode, int iStatus);
```

**Parameters**

- **pMsg**
  
  Pointer to string that holds a description of an error.
iCode

Number that identifies an error.

iStatus

Status of an error.

rstrMsg

An STL string that holds a description of an error.

Return Values

None.

GetCode

The GetCode method returns the error code that generated the exception.

Syntax

C++,
Java,
.NET

int GetCode();

Parameters

None.

Return Values

Returns an integer error code that generated the exception. The errors are described in the Cilerror.h include file—for more information, see CTI OS Keywords and Enumerated Types, on page 499.

GetStatus

The GetStatus method returns the error status that generated the exception.

Syntax

C++,
Java,
.NET

int GetStatus();

Parameters

None.

Return Values

Returns an integer error status that generated the exception.

GetString

The GetString method returns a text string containing the description of the exception.
Syntax

C++

    const char* GetString();

Java, .NET

    String GetString();

Parameters

None.

Return Values

Returns a text string containing the description of the exception.

What

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

Syntax

    const char* What();

Parameters

None.

Return Values

Returns a text string containing the description of the exception, the code of an error, and the status.

CWaitObject Class

CWaitObject is a CIL object that derives from CtIoSObject. It is a utility class that enables a thread to wait for one or more CTI events. The user can provide a list of events along with a wait timeout. Wait objects are created with the CreateWaitObject Session Object method and destroyed with the DestroyWaitObject Session Object method.

⚠️ Warning

You must not use a WaitObject instance within an event handler. Events are sent to desktop applications by a single thread in the CIL. If that thread is blocked while waiting for a specific event, the thread deadlocks and the event handler does not receive any more events.

Methods

The following table lists the CWaitObject class methods.
Table 164: CWaitObject Class Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateWaitObject</td>
<td>For more information, see Session Object, on page 307.</td>
</tr>
<tr>
<td>DestroyWaitObject</td>
<td>For more information, see Session Object, on page 307.</td>
</tr>
<tr>
<td>DumpEventMask</td>
<td>Returns a printable string listing the events in the CWaitObject's mask.</td>
</tr>
<tr>
<td>GetMask</td>
<td>Sets a user provided pointer to an Arguments object that contains the list of events that the object waits for.</td>
</tr>
<tr>
<td>GetTriggerEvent</td>
<td>Gets the ID of the event that triggered the WaitOnMultipleEvents method to wake.</td>
</tr>
<tr>
<td>InMask</td>
<td>Returns true if the specified event ID is in the list of events that the object waits for.</td>
</tr>
<tr>
<td>SetMask</td>
<td>Set the list of events that the object waits for.</td>
</tr>
<tr>
<td>WaitOnMultipleEvents</td>
<td>Waits for the events in the object's event mask for the specified time period or until one of the events occurs.</td>
</tr>
</tbody>
</table>

**CreateWaitObject**

For more information, see Session Object, on page 307.

**DestroyWaitObject**

For more information, see Session Object, on page 307.

**DumpEventMask**

The DumpEventMask method returns a printable string listing the events in the CWaitObject's mask.

**Syntax**

C++, Java, .NET

```cpp
string DumpEventMask();
```

**Parameters**

None.
Return Values
A printable string object listing the events in the wait mask.

GetMask

The GetMask method gets the list of events that the CWaitObject waits for.

Syntax

C++
bool GetMask(Arguments ** pMask);

Java, .NET
Arguments GetMask();

Parameters

pMask
A pointer to an Arguments object pointer. GetMask sets the value of pMask to a pointer to an Arguments object that contains the event mask.

Return Values
If the method was able to get the mask it returns true; otherwise, it returns false. For Java and .NET, the method returns null upon failure.

GetTriggerEvent

The GetTriggerEvent method returns the ID of the last event in the CWaitObject's mask that triggered the WaitOnMultipleEvents method to wake.

Syntax

C++
EnumCTIOS_EventID GetTriggerEvent();

Java
int GetTriggerEvent();

.NET
EventID GetTriggerEvent();

Parameters
None.

Return Values
The ID of the event or eUnknownEvent if no event triggered a wakeup.
**InMask**

The InMask method checks to see if the specified event is in the mask of events that the CWaitObject waits for.

**Syntax**

**C++**

```cpp
bool InMask(int iEventId);
```

**Java, .NET**

```java
boolean InMask(int iEventId);
```

**Parameters**

iEventId

The enumerated event ID of the event to check for.

**Return Values**

If the event is in the mask it returns true. Otherwise, it returns false.

---

**SetMask**

The SetMask method sets the list of events that the CWaitObject waits for.

**Syntax**

**C++**

```cpp
bool SetMask(Arguments & args);
```

**Java, .NET**

```java
boolean SetMask(Arguments rArgs);
```

**Parameters**

args

A reference to an Arguments object containing the list of events to wait for. The Arguments contains values where the keys are “Event1” through “EventN” and the values are the enumerated event IDs.

**Return Values**

The method returns true if it is able to set the. Otherwise it returns false.

---

**WaitOnMultipleEvents**

The WaitOnMultipleEvents method waits for the events in the CWaitObject's wait mask or returns if one of the events has not occurred after the specified timeout period. This is a “one of” method that returns after one of the specified events occurs.
Syntax

C++

    int WaitOnMultipleEvents(DWORD dwMilliseconds = INFINITE);

Java, .NET

    int WaitOnMultipleEvents(long iMilliseconds);

Parameters

Milliseconds

The maximum length of time in milliseconds to wait before timing out. The default is INFINITE if called without arguments. For Java and .NET, a value of zero causes this method to wait infinitely.

Return Values

The WaitOnMultipleEvents method returns one of the values listed in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>When Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_SIGNED</td>
<td>If one of the events in the mask occurred.</td>
</tr>
<tr>
<td>EVENT_WAIT_TIMEDOUT</td>
<td>If the timeout period elapsed.</td>
</tr>
<tr>
<td>WAIT_FAILED</td>
<td>If unable to wait on the events in the mask.</td>
</tr>
</tbody>
</table>

Logger Class (.NET and Java Only)

The Logger class creates a Logger object and a LogManager object, if one does not already exist. Any object that needs to perform logging must instantiate the Logger class. The Logger class communicates with the singleton LogManager object, which acts as the centralized logging facility. The Logger class also defines tracing constants.

Methods

The following table list the methods in the Logger class.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddLogListener</td>
<td>Registers a listener with the LogManager.</td>
</tr>
<tr>
<td>GetTraceMask</td>
<td>Gets the current trace mask. Trace masks define trace levels, such as TRACE_MASK_CRITICAL, which enables tracing for critical errors. See the LogManager Javadoc for a description of trace masks that define tracing.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Logger (Constructor)</td>
<td>Creates a Logger object and also a LogManager object if one does not already exist. If one is already created, it just gets a reference to the existing singleton LogManager object.</td>
</tr>
<tr>
<td>RemoveLogListener</td>
<td>Unregisters a listener from the LogManager.</td>
</tr>
<tr>
<td>SetTraceMask</td>
<td>Sets the current trace mask.</td>
</tr>
<tr>
<td>Trace</td>
<td>Sends a trace message to the central LogManager with the specified trace mask.</td>
</tr>
</tbody>
</table>

**Logger() Constructor**

The Logger constructor creates a Logger object and also a LogManager object if one does not already exist. If a LogManager exists, the Logger gets a reference to the existing singleton LogManager object.

**Syntax**

```java
void Logger()
```

**Parameters**

None.

**Return Values**

None.

**GetTraceMask**

The GetTraceMask method gets the current trace mask.

**Syntax**

```java
int GetTraceMask()
```

**Parameters**

None.

**Return Values**

An int containing the current trace mask.
SetTraceMask

The SetTraceMask method sets the current trace mask.

Syntax

void SetTraceMask(int iTraceMask)

Parameters

iTraceMask

The binary or combination of trace mask bits.

Note

For more information about trace masks available in the .NET CIL, see Table 181: Trace Masks in .NET CIL, on page 509.

Return Values

None.

AddLogListener

The AddLogListener method registers a listener with the LogManager.

Syntax

Java

void AddLogListener(ILogListener rListener)

.NET

void AddLogListener (LogEventHandler rListener)

Parameters

rListener

Java: Reference to the new listener.
.NET: Reference to a LogManager LogEventHandler delegate.

Return Values

None.

RemoveLogListener

The RemoveLogListener method unregisters a listener from the LogManager.
Syntax

Java
   void RemoveLogListener(IListener rListener)

.NET
   void RemoveLogListener (LogEventHandler rListener)

Parameters

rListener
   Java: Reference to the listener to be removed.
   .NET: Reference to a LogManager LogEventHandler delegate to be removed.

Return Values

None.

Trace

The Trace method sends a trace message to the central LogManager with the specified trace mask. If the trace mask set on the Logger contains all the bits in the trace mask that is passed into this method, the LogManager sends the trace string to all log listeners.

Syntax

int Trace(int iTraceMask, String sMessage)

Parameters

traceMask
   Trace mask for this message.

traceMessage
   String containing trace message.

Return Values

int 0 if traced; -1 if not traced.

LogWrapper Class (.NET and Java Only)

The LogWrapper class instantiates the default logging mechanism. By default, the LogWrapper writes trace messages to System.Console.Out. If you instantiate the LogWrapper by passing it a filename, then the LogWrapper writes trace messages to the specified file.

Methods

The following table lists the LogWrapper class methods.
### LogWrapper Class Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispose</td>
<td>Releases system resources used by the LogWrapper (.NET only).</td>
</tr>
<tr>
<td>GetMaxDaysBeforeExpire (.NET Only)</td>
<td>Obtains the current log file age threshold beyond which the active log file is rolled over into a new file regardless of file size.</td>
</tr>
<tr>
<td>GetMaxFileSize (.NET only)</td>
<td>Obtains the current log file size threshold beyond which a new file is created.</td>
</tr>
<tr>
<td>GetMaxNumberOfFiles (.NET Only)</td>
<td>Obtains the current number of log files threshold beyond which older files are deleted.</td>
</tr>
<tr>
<td>GetTraceMask</td>
<td>Gets the current trace mask.</td>
</tr>
<tr>
<td>LogWrapper (String fileName)Constructor</td>
<td>Creates a new LogWrapper object that traces to the file specified in the fileName parameter.</td>
</tr>
<tr>
<td>LogWrapper (string, int, int, int)</td>
<td>Creates a new LogWrapper object that traces to the file specified in the fileName parameter and sets all the provided tracing properties. If the corresponding parameter value is set to 0 then the default value is used.</td>
</tr>
<tr>
<td>SetMaxDaysBeforeExpire</td>
<td>Changes the current log file age threshold beyond which the active log file is rolled over into a new file regardless of file size.</td>
</tr>
<tr>
<td>SetMaxFileSize</td>
<td>Changes the current log file size threshold beyond which a new file is created.</td>
</tr>
<tr>
<td>SetMaxNumberOfFiles</td>
<td>Changes the current number of log files threshold beyond which older files are deleted.</td>
</tr>
<tr>
<td>SetTraceMask</td>
<td>Sets the current trace mask.</td>
</tr>
<tr>
<td>UpdateTraceSettings</td>
<td>Parses TraceConfig.cfg and imports the settings contained within.</td>
</tr>
<tr>
<td>WriteTraceLine</td>
<td>Prints a string to the active trace file or to System.Console.Out if no active trace file exists.</td>
</tr>
</tbody>
</table>

**LogWrapper() Constructor**

The LogWrapper constructor creates a new LogWrapper object that writes tracing messages to System.Console.Out. This constructor also creates an instance of the LogManager, if one does not already
exist. If you are using the .NET CIL, call the Dispose method to release system resources when you no longer need the LogWrapper.

**Syntax**

```csharp
void LogWrapper()
```

**Parameters**

None.

**Return Values**

None.

---

**LogWrapper(string filename) Constructor**

This constructor creates a new LogWrapper object that traces to the file specified in the `filename` parameter. If you are using the .NET CIL, call the Dispose method to release system resources when you no longer need the LogWrapper.

**Syntax**

```csharp
void LogWrapper (string sFileName)
```

**Parameters**

- `sFileName`
  - Name of the trace file.

**Return Values**

None.

---

**LogWrapper(string int int int) Constructor**

Creates a new LogWrapper object that traces to the file specified in the `filename` parameter and sets all the provided tracing properties. If the corresponding parameter value is set to 0 then the default value is used. If you are using the .NET CIL, call the Dispose method to release system resources when you no longer need the LogWrapper.

**Syntax**

Java

```java
void LogWrapper (string sFileName, long iMaxSize, int iArchives, int iFlushIntervalMs)
```

.NET

```csharp
void LogWrapper (string sFileName, int maxSizeKB, int maxFiles, int daysUntilExpiration)
```
**Parameters**

.NET and Java: `fileName`
Name of the trace file.

.NET: `maxSizeKB`
Maximum size of a single trace file in KB (default is 2048 KB).

Java: `iMaxSize`
Maximum size of a single trace file.

.NET: `maxFiles`
Maximum number of trace files to create before older files are deleted (default is 4).

Java: `iArchives`
Maximum number of trace files stored.

.NET: `daysUntilExpiration`
Maximum age (in days) of the active trace file before it is rolled over to a new file regardless of size (default is 1 day).

Java: `iExpires`
Number of days before the trace file expires.

Java: `iFlushIntervalMs`
Number of milliseconds before data is flushed to the trace file. There is no .NET counterpart for this parameter.

**Return Values**

None.

**Dispose (.NET Only)**

The Dispose method releases system resources used by the LogWrapper.

**Syntax**

```csharp
void Dispose ()
```

**Parameters**

None.

**Return Values**

None.

**GetMaxDaysBeforeExpire (.NET Only)**

The GetMaxDaysBeforeExpire method gets the current log file age threshold beyond which the active log file is rolled over into a new file regardless of file size.
**Syntax**

```csharp
int GetMaxDaysBeforeExpire ()
```

**Parameters**

None.

**Return Values**

Current age threshold.

---

**SetMaxNumberFiles**

The `SetMaxNumberFiles` method changes the current number of log files threshold beyond which older files are deleted. If the provided value is not greater than zero, the default value of 4 is used.

**Syntax**

**Java**

```java
void SetMaxNumberFiles (int iArchives)
```

**.NET**

```csharp
void SetMaxNumberFiles (int maxFiles)
```

**Parameters**

maxTraceFiles

New number of files threshold. If 0 is specified, the default value is used.

**Return Values**

None.

---

**GetMaxNumberFiles (.NET Only)**

The `GetMaxNumberFiles` method gets the current number of log files threshold beyond which older files are deleted.

**Syntax**

```csharp
int GetMaxNumberFiles ()
```

**Parameters**

None.

**Return Values**

Current number of files threshold.
SetMaxDaysBeforeExpire

The SetMaxDaysBeforeExpire method changes the current log file age threshold beyond which the active log file is rolled over into a new file regardless of file size.

**Syntax**

**Java**

```java
void SetMaxDaysBeforeExpire (int iExpires)
```

**.NET**

```csharp
void SetMaxDaysBeforeExpire (int maxDaysUntilExpiration)
```

**Parameters**

maxDaysUntilExpiration

New age threshold. If value is not greater than zero, the default value of 1 is used.

**Return Values**

None.

ProcessConfigFile

The ProcessConfigFile method opens the default config file (TraceConfig.cfg) in the parent directory and updates LogWrapper trace settings with data from the config file.

**Syntax**

```java
boolean ProcessConfigFile()
```

**Parameters**

None.

**Return Values**

Returns true if operation succeeded and false if unable to open the TraceConfig.cfg file.
SilentMonitorManager Object

The SilentMonitorManager object provides developers with an interface to silently monitor behavior. The SilentMonitorManager object performs all silent monitor tasks, such as starting, stopping, and managing silent monitor sessions. The SilentMonitorManager object stores specific silent monitor session information as properties.

You can use the SilentMonitorManager object in two different modes:

- In Monitoring Mode, an application that wants to silently monitor conversation without being noticed by the calling parties must create a SilentMonitorManager object and set the mode to eSMMonitoringMode using the StartSMMonitoringMode method.
- In Monitored Mode, an application accepts requests to initiate silent monitor sessions to forward the voice conversations to the remote monitoring application. The application creates a SilentMonitorManager object and sets the mode to eSMMonitoredMode using the StartSMMonitoredMode method.

For more information about these modes see Silent Monitor Session, on page 89 in Building Your Custom CTI Application, on page 33

---

**Note**

SilentMonitorManager Object methods and properties are not available in the Java or .NET CILs.

**Note**

SilentMonitorManager Object methods and properties are supported for use with Unified CCE only.

**Note**

SilentMonitorManager Object methods and properties are only supported for CTI OS based silent monitoring.
Properties

The following table lists the SilentMonitorManager object properties.

### Table 168: SilentMonitorManager Object Properties

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Heartbeat interval for the silent monitor session.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Timeout for no activity.</td>
</tr>
<tr>
<td>IPPhoneInformation</td>
<td>ARGUMENTS</td>
<td>This property is only accessible via the GetIPPhoneInfo method. It contains all the information related to the IP Phone used by the application.</td>
</tr>
<tr>
<td>MediaTerminationPort</td>
<td>INT</td>
<td>TCP/IP port where monitored conversation is sent for playback on system sound card.</td>
</tr>
<tr>
<td>SessionInformation</td>
<td>ARGUMENTS</td>
<td>This property is only accessible via the GetSessionInfo method. It contains all the information related to the current active silent monitor session.</td>
</tr>
<tr>
<td>SMManagerMode</td>
<td>SHORT</td>
<td>Mode in which the manager object operates (for more information, see the table below).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If SetIPPhoneInfo is used, SMManagerMode attempts to use the information provided.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> Only applies to CTI OS based Silent Monitor.</td>
</tr>
</tbody>
</table>

### Table 169: SMManagerMode Values

<table>
<thead>
<tr>
<th>enum Value</th>
<th>Description</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eSMModeNotSet</td>
<td>Mode not set.</td>
<td>-1</td>
</tr>
<tr>
<td>eSMMonitoredMode</td>
<td>The manager accepts request for silent monitor sessions and forwards voice to the monitoring application.</td>
<td>0</td>
</tr>
</tbody>
</table>
### Methods

The following table lists the SilentMonitorManager object methods.

**Table 170: SilentMonitorManager Object Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcceptSilentMonitoring</td>
<td>Establishes a silent monitor session and immediately starts sending audio.</td>
</tr>
<tr>
<td>GetIPPhoneInfo1</td>
<td>Retrieves the information of the IP Phone used by the client application.</td>
</tr>
<tr>
<td></td>
<td>Gets its information from the RTP events that occur when RTP streams are created and modified.</td>
</tr>
<tr>
<td>GetSessionInfo</td>
<td>Retrieves the information related to the current silent monitor session.</td>
</tr>
<tr>
<td>GetSMSessionList</td>
<td>Retrieves a list of all active silent monitor sessions.</td>
</tr>
<tr>
<td>IsMonitoredTarget</td>
<td>Determines if the device/agent is a target being monitored.</td>
</tr>
<tr>
<td>SetIPPhoneInfo⁴</td>
<td>Saves the information of the IP Phone used by the client application.</td>
</tr>
<tr>
<td>StartSilentMonitorRequest</td>
<td>Sends a silent monitor session start request to a targeted client.</td>
</tr>
<tr>
<td>StartSMMonitoredException</td>
<td>Puts the SilentMonitorManager in Monitored mode.</td>
</tr>
<tr>
<td>StartSMMonitoringMode</td>
<td>Puts the SilentMonitorManager in Monitoring mode.</td>
</tr>
<tr>
<td>StopSilentMonitorMode</td>
<td>Sets the SilentMonitorManager mode to eSMMModeNotSet. If a silent monitor session is active at this time, the session is terminated.</td>
</tr>
<tr>
<td>StartSilentMonitorRequest</td>
<td>The StartSilentMonitorRequest () method is used to initiate a CTI OS based silent monitor session. When this method is called and Cisco Unified Communications Manager based silent monitor is configured, it returns E_CTIOS_INVALID_SILENT_MONITOR_MODE.</td>
</tr>
</tbody>
</table>
Argument Parameter Rules

The following rules apply to the optional_args and reserved_args parameters in SilentMonitorManager object methods:

- In VB, you can ignore these parameters. For example, you can treat the line:

  ```vba
  StartSMMonitoringMode([reserved_args As IArguments]) As Long
  ```

  as follows:

  ```vba
  StartSMMonitoringMode()
  ```

  To ignore these parameters in COM you must send a NULL, as shown:

  ```vba
  StartSMMonitoringMode(NULL)
  ```

AcceptSilentMonitoring

The AcceptSilentMonitoring method establishes the silent monitor session requested by the OnSilentMonitorRequestedEvent and immediately starts sending audio to the monitoring client. You should only use this method if the parameter DoDefaultMessageHandling was set to False when the subscriber handled the OnSilentMonitorRequestedEvent event.

**Syntax**

**C++**

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

**COM**

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

**VB**

```vba
AcceptSilentMonitoring (ByVal args as CTIOSCLIENTLIB.IArguments) As Long
```

**Parameters**

`args`

---

5 GetIPPhoneInfo and SetIPPhone Info are used by SilentMonitorManager in the following manner. The RTPStartedEvent arrives and SilentMonitorManager uses GetIPPhoneInfo to store the IP address and port carried in the RTPStartedEvent. The SilentMonitorStartRequestedEvent arrives and SilentMonitorManager uses GetIPPhoneInfo to retrieve the stored IP address and port to build the packet filter. The SilentMonitorManager uses SetIPPhoneInfo internally to populate IP phone information carried in RTPStartedEvents.
Arguments array that contains the parameters listed in the following table:

**Table 171: AcceptSilentMonitoring arguments array parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the object being monitored.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the monitoring application.</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port of the monitoring application.</td>
</tr>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique identifier for the Silent Monitor Session.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Heartbeat interval for the silent monitor session.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Timeout for no activity.</td>
</tr>
<tr>
<td>OriginatingServerID</td>
<td>STRING</td>
<td>TCP/IP Address:Port of the CTI OS server from which the request originated.</td>
</tr>
<tr>
<td>OriginatingClientID</td>
<td>STRING</td>
<td>Client Identification of the monitoring application.</td>
</tr>
<tr>
<td>DoDefaultMessage Handling</td>
<td>BOOLEAN</td>
<td>When this parameter is set to True, it instructs the SilentMonitorManager to immediately start sending audio and establish the silent monitor session. If this value is set to False, it instructs the SilentMonitorManager not to send voice and not to establish the silent monitor session. It is then the responsibility of the subscriber to report this status accordingly.</td>
</tr>
</tbody>
</table>

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.
GetIPPhoneInfo

The GetIPPhoneInfo method gets the information about the client application IP Phone.

You do not have to use this method. You can use the defaults to figure out the information to sniff packets from.

Syntax

C++

```cpp
C++: Arguments * GetIPPhoneInfo(void);
```

COM

```cpp
HRESULT GetIPPhoneInfo ( /*[out,retval]*/ IArguments ** IPPhoneInfo);
```

VB

```cpp
GetIPPhoneInfo () as CTIOSCLIENTLIB.IArguments
```

Parameters

None.

Return Value

This method returns an Arguments array that contain the parameters listed in the following table.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientAddress</td>
<td>STRING</td>
<td>IP Address of the IP Phone to be used by the client application.</td>
</tr>
<tr>
<td>BitRate</td>
<td>INT</td>
<td>Audio transmission bit rate.</td>
</tr>
<tr>
<td>PacketSize</td>
<td>INT</td>
<td>Number of milliseconds of audio stored in a packet.</td>
</tr>
<tr>
<td>Direction</td>
<td>SHORT</td>
<td>One of the following values that indicates the direction of voice flow between the calling party and the called party:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: Bidirectional</td>
</tr>
</tbody>
</table>
### Keyword

<table>
<thead>
<tr>
<th><strong>Keyword</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>RTPTypea</td>
<td>SHORT</td>
<td>One of the following values that indicate the type of RTP messages between the calling party and the called party: 0: audio 1: video 2: data</td>
</tr>
<tr>
<td>EchoCancelation</td>
<td>SHORT</td>
<td>One of the following values that indicates whether the echo cancellation feature is enabled on this IP Phone: 0: Off 1: On</td>
</tr>
<tr>
<td>PayLoadType</td>
<td>SHORT</td>
<td>Audio codec type.</td>
</tr>
</tbody>
</table>

## GetSessionInfo

The GetSessionInfo method retrieves the information related to the current silent monitor session.

### Syntax

**C++**

```c++
Arguments * GetSessionInfo(Arguments & args) ;
```

**COM**

```c
HRESULT GetSessionInfo ( /*[in]*/ IArguments * args, /*[out,retval]*/ IArguments * SMSessionInfo );
```

**VB**

```vb
GetSessionInfo (ByVal args as CTIOSCLIENTLIB.IArguments) As CTIOSCLIENTLIB.IArguments
```

### Parameters

**args**

Arguments array that contains one of the parameters listed in the following table:

#### Table 173: GetSessionInfo Arguments Array Parameters

<table>
<thead>
<tr>
<th><strong>Keyword</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique silent monitor session Object ID of the target object that is being monitored.</td>
</tr>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the target object that is being monitored.</td>
</tr>
</tbody>
</table>
Return Values

This method returns an Arguments array containing the key/value pairs listed in the following table:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique silent monitor session Object ID of the target object that is being monitored.</td>
</tr>
<tr>
<td>SMSessionStatus</td>
<td>SHORT</td>
<td>One of the ISilentMonitorEvent status codes in Table 120: ISilentMonitorEvent Status Codes, on page 276.</td>
</tr>
<tr>
<td>AudioMode</td>
<td>INT</td>
<td>Reserved. Specifies the audio mode bitmask.</td>
</tr>
<tr>
<td>AgentID/DeviceID</td>
<td>STRING</td>
<td>Agent ID or DeviceID of the target being monitored.</td>
</tr>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the target object being monitored.</td>
</tr>
<tr>
<td>MonitoredDeviceIPAddress</td>
<td>STRING</td>
<td>TCP/IP Address of the monitored IP Phone.</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>ID of the peripheral associated with the agent and IP phone.</td>
</tr>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP Address of the system receiving audio.</td>
</tr>
<tr>
<td>MonitoringIPPort</td>
<td>INT</td>
<td>TCP/IP port on which receiving system is listening for audio.</td>
</tr>
</tbody>
</table>

GetSMSessionList

The GetSMSessionList method returns an Arguments array that contains the parameters listed in Table 177: StartSilentMonitorRequest Arguments Array Parameters, on page 493. All parameters are required.

Syntax

C++
```
Arguments * CIL_API GetSMSessionList(void)
```

COM
```
HRESULT GetSMSessionList([out,retval] IArguments **pIArguments );
```

VB
```
GetSMSessionList () as CTIOSCLIENTLIB.IArguments
```
Parameters

None.

Return Values

Arguments array that contains a list of all Silent Monitor sessions. The current version only allows one active session, so the main use for this function is to use the NumElements method on the returned Arguments array to determine if the current SilentMonitorManager is in an active Silent Monitor session.

**IsMonitoredTarget**

The IsMonitoredTarget method determines if the specified device or agent is a target that is being monitored.

Syntax

**C++**

```cpp
bool IsMonitoredTarget (Arguments & args);
```

**COM**

```cpp
HRESULT IsMonitoredTarget ( /*[in]*/ IArguments * args, /*[out,retval]*/ VARIANT_BOOL * bMonitored );
```

**VB**

```vb
IsMonitoredTarget () As Boolean
```

Parameters

*args*

Arguments array that contains the parameter listed in the following table:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoredUniqueObjectID</td>
<td>STRING</td>
<td>Unique Object ID of the target object being monitored.</td>
</tr>
</tbody>
</table>

Return Value

True if the specified MonitoredUniqueObjectID corresponds to the monitored agent or device; False otherwise.

**SetIPPhoneInfo**

The SetIPPhoneInfo method saves the information of the IP Phone used by the client application.

You use the SetIPPhoneInfo() function to set the specific IP address/port to sniff on for RTP packets on the agent system. If you call StartSMMonitoredMode() and have not called SetIPPhoneInfo(), then the silent monitor library sniffs on all IP interfaces on the agent system and figures out the correct interface. If you set a specific ip address/port to sniff with SetIPPhoneInfo(), then the silent monitor library sniffs for RTP packets on the agent system only on that specific address and specific port. SetIPPhoneInfo() is used externally by the Agent control to set a specific address for silent monitor sniffing.
Syntax

C++

    int SetIPPhoneInfo (Arguments & args);

COM

    HRESULT SetIPPhoneInfo ( /*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode);

VB

    SetIPPhoneInfo (ByVal args as CTIOSCLIENTLIB.IArguments ) As Long

Parameters

args

Arguments array that can contain the parameters listed in the following table:

Table 176: SetIPPhoneInfo arguments array parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientAddress (required)</td>
<td>STRING</td>
<td>IP Address of the IP Phone to be used by the client application.</td>
</tr>
<tr>
<td>BitRate (optional)</td>
<td>INT</td>
<td>Audio transmission bit rate.</td>
</tr>
<tr>
<td>PacketSize (optional)</td>
<td>INT</td>
<td>Number of milliseconds of audio stored in a packet.</td>
</tr>
<tr>
<td>Direction (optional)</td>
<td>SHORT</td>
<td>One of the following values that indicates the direction of voice flow between the calling party and the called party:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: Bidirectional</td>
</tr>
<tr>
<td>RTPType (optional)</td>
<td>SHORT</td>
<td>One of the following values that indicates the type of RTP messages between the calling party and the called party:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: audio</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: video</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: data</td>
</tr>
</tbody>
</table>
One of the following values that indicates whether the echo cancellation feature is enabled on this IP Phone:

- 0: Off
- 1: On

**PayLoadType (optional)**

Audio codec type.

**StartSilentMonitorRequest**

The `StartSilentMonitorRequest` method sends a silent monitor session start request to a targeted client.

**Syntax**

**C++**

```cpp
int StartSilentMonitorRequest (Arguments & args, unsigned short * SMSessionKey );
```

**COM**

```cpp
HRESULT StartSilentMonitorRequest ( /*[in]*/ IArguments * args, /*/[out]*/ unsigned short * SMSessionKey, /*/[out,retval]*/ int * errorcode );
```

**VB**

```vb
StartSilentMonitorRequestInt(ByVal args as CTIOSCLIENTLIB.IArguments, ByRef SMSessionKey AsLong) As Long
```

**Parameters**

- **args**
  
  Arguments array that contains the parameters listed in the following table. All parameters are required.

**Table 177: StartSilentMonitorRequest Arguments Array Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgentID or DeviceID</td>
<td>STRING</td>
<td>AgentID or DeviceID of the target to monitor. Specify either an AgentID or a DeviceID, not both,</td>
</tr>
<tr>
<td>PeripheralID</td>
<td>INT</td>
<td>ID of the peripheral associated with the agent or device.</td>
</tr>
</tbody>
</table>

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Values**

Default CTIOS return values. For more information, see CIL Coding Conventions, on page 19.
### SilentMonitorManager Object

#### StartSMMonitoredMode

The StartSMMonitoredMode method puts the SilentMonitorManager in Monitored mode.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitoringIPAddress</td>
<td>STRING</td>
<td>TCP/IP address of the system receiving audio.</td>
</tr>
<tr>
<td>MonitoringIPPort (Optional)</td>
<td>INT</td>
<td>TCP/IP port where the monitoring application is listening for audio.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>INT</td>
<td>Interval in seconds between heartbeats.</td>
</tr>
<tr>
<td>HeartbeatTimeout</td>
<td>INT</td>
<td>Seconds elapsing before a Silent Monitor session is aborted because of no heartbeats received from the monitored peer.</td>
</tr>
</tbody>
</table>

**SMSessionKey**

An output parameter that contains the unique key to the started silent monitor session. You must use this key to perform any action on the currently active silent monitor session.

**errorcode**

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

### Return Values

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

### Remarks

If you use the DeviceID, there must be an agent associated with the device. The session timeouts if there is no agent logged into the device. An established silent monitor session ends if the associated agent logs out of the device.

E_CTIOS_INVALID_SILENT_MONITOR_MODE is returned when SilentMonitorManager.StartSilentMonitorRequest() is called when Cisco Unified Communications Manager based silent monitor is configured.

If an application using a version of the CIL that is older than 7.2(1) connects to a 7.2(1) CTI OS Server configured for Cisco Unified Communications Manager Based Silent Monitor and calls SilentMonitorManager.StartSilentMonitorRequest(), the application receives an OnSilentMonitorStatusReportEvent carrying a status code of eSMStatusCCMSilentMonitor.
Syntax

C++

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

COM

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

VB

```vb
StartSMMonitoredMode (ByVal args as CTIOSCLIENTLIB.IArguments) As Long
```

Parameters

args

Arguments array that contains the following parameters listed in the following table:

**Table 178: StartSMMonitoredMode Arguments Array Parameters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>ARRAY</td>
<td>An array of IP addresses and/or hostnames for silent monitor services. These silent monitor service should all be members of the same cluster to ensure that the agent's calls can be silently monitored. The CIL randomly chooses one silent monitor service to which to connect. For more information about silent monitor service cluster configuration, see <em>CTI OS System Manager’s Guide for Cisco Unified ICM/Contact Center Enterprise &amp; Hosted</em>.</td>
</tr>
<tr>
<td>SMSAddr</td>
<td>STRING</td>
<td>If Cluster is not present, you can use this parameter to specify the address of a silent monitor service to which to connect.</td>
</tr>
<tr>
<td>SMSListenport</td>
<td>INT</td>
<td>The port on which the silent monitor services listen for connections.</td>
</tr>
<tr>
<td>SMSTOS</td>
<td>INT</td>
<td>The QoS setting for the connection.</td>
</tr>
<tr>
<td>SMSHeartbeats</td>
<td>INT</td>
<td>The interval in milliseconds between heartbeat packets.</td>
</tr>
<tr>
<td>SMSRetries</td>
<td>INT</td>
<td>The number of heartbeats that can be missed before the connection is aborted.</td>
</tr>
</tbody>
</table>
errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values
Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

StartSMMonitoringMode
The StartSMMonitoringMode method puts the SilentMonitorManager in Monitoring mode.

Syntax
C++
int StartSMMonitoringMode (Arguments & args);

COM
HRESULT StartSMMonitoringMode ( /*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode );

VB
StartSMMonitoringMode (ByVal args as CTOISCLIENTLIB.IArguments) As Long

Parameters
Table 179: StartSMMonitoringMode Arguments Array Parameters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMSAddr</td>
<td>STRING</td>
<td>A string that contains the address of the silent monitor service used to decode and play back the agent's phone call.</td>
</tr>
<tr>
<td>SMSListenport</td>
<td>INT</td>
<td>The port on which the silent monitor services listen for connections.</td>
</tr>
<tr>
<td>SMSTOS</td>
<td>INT</td>
<td>The QoS setting for the connection.</td>
</tr>
<tr>
<td>SMSHeartbeats</td>
<td>INT</td>
<td>The interval in milliseconds between heartbeat packets.</td>
</tr>
<tr>
<td>SMSRetries</td>
<td>INT</td>
<td>The number of heartbeats that can be missed before the connection is aborted.</td>
</tr>
</tbody>
</table>

errorcode
An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.
**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**StopSilentMonitorMode**

The StopSilentMonitorMode method sets the SilentMonitorManager mode to eSMModeNotSet. If a silent monitor session is active at the time, the session is terminated.

**Syntax**

C++

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

COM

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

VB

```vbnet
StopSilentMonitorMode (ByVal reserved_args as CTIOSCLIENTLIB.IArguments) As Long
```

**Parameters**

reserved_args

Not currently used, reserved for future use.

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

**Return Values**

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.

**StopSilentMonitorRequest**

The StopSilentMonitorRequest method stops the Active silent monitor session.

**Syntax**

C++

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

COM

```csharp
HRESULT StopSilentMonitorRequest ( /*[in]*/ IArguments * args, /*[out,retval]*/
int * errorcode );
```

VB

```vbnet
StopSilentMonitorRequest (ByVal args as CTIOSCLIENTLIB.IArguments) As Long
```

**Parameters**

args
Arguments array that contains the parameter listed in the following table:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMSessionKey</td>
<td>UNSIGNED SHORT</td>
<td>Unique key of the current active silent monitor session</td>
</tr>
</tbody>
</table>

errorcode

An output parameter (return parameter in VB) that contains an error code from Table 6: CIL Error Codes, on page 21.

Return Values

Default CTI OS return values. For more information, see CIL Coding Conventions, on page 19.
The CTI OS Client Interface Library uses the Arguments structure to pass key-value pairs between the client and the server (for more information about Arguments, see Helper Classes, on page 441). Throughout this document all event and method parameter lists, as well as object properties, are listed with the keywords and the types associated with those keywords.

The expected (required and optional) keywords are referred to in this document by string name. For example, the Agent's property for agent identifier is referred to as AgentID.

In addition to using the string name for a keyword, programmers can take advantage of an enumeration of keywords as well.

The enumeration of keywords is presently only available in the C++ CIL.

For each string keyword, a corresponding enumerated keyword exists. The enumerated keyword is the same name, preceded by the prefix “ekw”. For example, the AgentID string keyword is mapped to the enumerated keyword ekwAgentID.

Usage Example in C++:

```c++
Arguments & args = Arguments::CreateInstance();
args.AddItem(ekwAgentID, "22866");
args.AddItem(ekwAgentInstrument, "23901");
pAgent->Login(args);
args.Release();
```

The complete set of standard keywords used in CTI OS is available in the C++ header file “ctioskeywords.h”, located in the \Distribution\cpp\Include directory on the CTI OS toolkit media.
Java CIL Keywords

For Java CIL, the CtiOs_IKeywordIDs interface contains a list of known Java CIL CTI OS keywords. For more information, see the Java CIL Javadoc file.

.NET CIL Keywords

The Cisco.CtiOs.Util.Keywords.Enum_CtiOs enum contains the list of CTI OS keyword IDs.

Enumerated Types

CTI OS employs enumerated types to provide symbolic names for commonly recurring values:

- In C++, Visual Basic, and COM, these are presented as enumerated types.
- In Java, special interfaces are used to simulate enumerated types. For more information, see Java Interfaces, on page 500.

The complete set of enumerated types and their values are available in the following locations:

- For C++ CIL using static libraries: the complete set of enumerated types is located in the C++ header file “cilmessages.h”, located in the C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\Include directory on the CTI OS toolkit media.
- For COM (Visual Basic and Visual C++): the complete set of enumerated types is located in the CTIOSClient Type Library, which is compiled into the “CTIOSClient.dll” file, located in the C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\COM Servers and Activex Controls directory on the CTI OS toolkit media.

In the Java CIL, the CTIOS_ Enums interface contains the Java CIL enumerated types. For more information, see the Java CIL Javadoc file.

In the .NET CIL, the CtiOs_ Enums class contains the .NET CIL enumerated types:

- For Java: To be supplied with Java package release.

Java Interfaces

The Java CIL handles the C++ CIL enums through the use of interfaces. The custom application can then either implement those interfaces and use the static data members without referencing them with the interface name first, or it can access those members through referencing. By convention, the name of the Java interface is the same as the enum tag but with the “enumCTIOS_” prefix substituted with “CtiOs_I”. So for example, the following C++ CIL enum:

```java
enum enumCTIOS_AgentState{
    eLogin = 0,
    eLogout = 1,
    eNotReady = 2,
    eAvailable = 3,
    eTalking = 4,
    eWorkNotReady = 5,
    eWorkReady = 6,
}
```
is implemented in the Java CIL as follows:

```java
public interface CtiOs_IAgentState{
    public static final int eLogin = 0,
    eLogout = 1,
    eNotReady = 2,
    eAvailable = 3,
    eTalking = 4,
    eWorkNotReady = 5,
    eWorkReady = 6,
    eBusyOther = 7,
    eReserved = 8,
    eUnknown = 9,
    eHold=10;
}
```

A Java CIL application can access those defined values in one of two ways; either by implementing the interface, as shown:

```java
public class MyAgent extends CtiOsObject implements CtiOs_IAgentState {

    ..................................................

    public int MyLogin(Arguments rArguments)
    {
        ............................
        //Access eLogin directly
tArguments.AddItemInt( "agentstate", eLogin );
        ............................
    }
}
```

or by referencing as follows:

```java
public class MyAgent extends CtiOsObject{

    ..................................................

    public int MyLogin(Arguments rArguments)
    {
        ............................
        rArguments.AddItemInt( "agentstate", CtiOs_IAgentState.eLogin );
        ............................
    }
}
```
APPENDIX A

CTI OS Logging

- CTI OS Client Logs (COM and C++), on page 503
- Set Trace Levels (COM and C++), on page 504
- Trace Configuration (COM and C++), on page 504
- Java CIL Logging Utilities, on page 505
- Logging and tracing (Java), on page 507
- Logging and tracing (.NET), on page 508

CTI OS Client Logs (COM and C++)

If you install the tracing mechanism, the COM and C++ CILs automatically create a log file and trace to it. The trace log file name and location for client processes is found under the following Windows registry key:

HKEY_LOCAL_MACHINE\Software\Cisco Systems, Inc.\CTIOS Tracing

The default filename is CtiostClientLog. Log files are created using the convention <TraceFileName>.<Windows username>.mmdd.hhmmss.log. The files are created in the current directory of the executing program, such as the directory into which you install the Agent Desktop. You can provide a fully qualified path for the TraceFileName if you wish to store the files in a different location. For example, setting the following value stores the log files in the directory C:\Temp, using the naming convention CtiostClientLog.<Windows user name>.mmdd.hhmmss.log.

C:\Temp\CtiostClientLog

Client trace files are formatted in ASCII text that you can open them with a conventional text editor such as Notepad.

Install Tracing Mechanism (COM and C++)

To install the tracing mechanism:

Procedure

Step 1 Copy the tracing executable, ctiostracecexec.exe, from the distribution media to the folder in which your application is located.
Step 2 Open a command window and register the tracing mechanism:

ctiostracertext.exe /regserver

Step 3

Set Trace Levels (COM and C++)

You must set the trace level in the registry by creating a TraceMask registry value within the HKEY_LOCAL_MACHINE\Software\Cisco Systems, Inc.:CTIOS Tracing key and setting its value to 0x40000307.

[HKEY_CURRENT_USER\Software\Cisco Systems, Inc.:CTIOS Tracing]\"TraceMask"=dword:40000307

Trace levels for client processes, such as the Agent Desktop phone, are stored under the following registry key:

[HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems, Inc.:CTIOS Tracing]\"TraceFileName"="%HOMEPATH%\CtiOsClientLog" \"TraceMask"=dword:00000000\n\"MaxDaysBeforeExpire"=dword:00000007 \"MaxFiles"=dword:00000005 \"MaxFileSizeKb"=dword:00000800\n\"FlushIntervalSeconds"=dword:0000001e \"TraceServer"="C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\Trace\CTIOSTraceText.exe"

For CTIOS server versions 7.5(10), 8.0(3) and later the default trace level will not print the call variable in CTIOS server logs. This has been done as an enhancement to reduce the log size in these two versions and above. To get the call variable in CTIOS logs you need to set the trace level to 0x400000.

Note

You can configure CTIO Tracing globally for the entire machine (using the TraceMask setting on HKLM) and per user (using the TraceMask setting on HKCU).

Warning

If the TraceMask is not set or if it is set incorrectly, the application's performance can be negatively affected. The preferred setting for normal operation is 0x40000307.

Trace Configuration (COM and C++)

You can set C++ and COM client trace configuration parameters in the Windows registry at the following key. For more information about configuring tracing for the Java CIL, see Java CIL Logging Utilities, on page 505. For more information about configuring tracing for the .NET CIL, see Logging and tracing (.NET), on page 508.

HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems\CTIOS Tracing

These settings are defined as follows:
Table 180: Configuring Tracing Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Optimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlushIntervalSeconds</td>
<td>Maximum number of seconds before the trace mechanism transfers data to the log file.</td>
<td>30</td>
</tr>
<tr>
<td>MaxDaysBeforeExpire</td>
<td>Maximum number of days before a log file is rolled over into a new log file regardless of the size of the file.</td>
<td>7</td>
</tr>
<tr>
<td>MaxFiles</td>
<td>Maximum number of log files that can exist in the log file directory before the logging mechanism starts overwriting old files.</td>
<td>5</td>
</tr>
<tr>
<td>MaxFileSizeKb</td>
<td>Maximum size of a log file in kilobytes. When a log file reaches the maximum size, a new log file is created.</td>
<td>2048</td>
</tr>
<tr>
<td>TraceMask</td>
<td>Bit mask that determines the categories of events that are traced.</td>
<td>0x40000307</td>
</tr>
</tbody>
</table>

Java CIL Logging Utilities

The Java CIL provides a different logging facility than the C++ CIL. This gives the customer application more flexibility in how trace messages are handled. It also limits the number of special privileges the browser would need to give the applet using the CIL; the Java CIL only needs to access the network and not the file system. For that reason, the Java CIL does its tracing through the firing of special events called “LogEvents” that the custom application can trap and handle in however way it sees fit.

The Java CIL provides the following objects for logging as part of the utilities package.

ILogEvents

This interface must be implemented by a class interested in receiving Java CIL LogEvents. It has only one method.

```java
void processLogEvent(LogEvent event)
```

LogEvent

A custom application that is interested in receiving LogEvents receives an object of this type whenever a log message is generated. This class extends the Java “EventObject”, and has one public method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
**Logger**

A custom application that is interested in firing or handling its own LogEvents can create an instance of this class.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logger</td>
<td>Public constructor of the Logger object.</td>
</tr>
<tr>
<td>Trace</td>
<td>Lets the custom app fire a LogEvent.</td>
</tr>
<tr>
<td>GetTraceMask</td>
<td>Gets the trace mask.</td>
</tr>
<tr>
<td>IsTraceMaskEnabled</td>
<td>Determines if a certain trace mask is set.</td>
</tr>
<tr>
<td>addLogListener</td>
<td>Subscribes to receive LogEvents.</td>
</tr>
<tr>
<td>removeLogListener</td>
<td>Unsubscribes from receiving LogEvents.</td>
</tr>
</tbody>
</table>

**Syntax**

```java
Logger()
int Trace(long nMsgTraceMask, String message)
long GetTraceMask()
boolean IsTraceMaskEnabled(long nMsgTraceMask)
void addLogListener(ILogEvents logEvents)
void removeLogListener(ILogEvents logEvents)
```

where logEvents implements the ILogEvents interface.

**LogEventsAdapter**

This is a wrapper class around the Logger facility. A custom application that is interested in tracing but does not want to implement its own ILogEvents interface can create an instance of this class. The adapter class provides two constructors, a default one that automatically logs to the Java console and one that takes in an output filename.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogEventsAdapter</td>
<td>Public constructor.</td>
</tr>
<tr>
<td>startLogging</td>
<td>Starts receiving LogEvents.</td>
</tr>
<tr>
<td>stopLogging</td>
<td>Stops receiving LogEvents.</td>
</tr>
<tr>
<td>processLogEvent</td>
<td>Handles a LogEvent.</td>
</tr>
</tbody>
</table>
Logging and tracing (Java)

The Java CIL tracing mechanism behaves differently from that of the COM and C++ CILs. The Java CIL does not automatically create a log file and trace to it. You must develop the custom application to create and maintain the log file.

The Java CIL provides classes that allow you to write tracing messages from CTI applications. You can create a class that implements ILogListener, register it with the LogManager, and write the trace events to a log file.

The Java CIL also includes the LogWrapper class, which implements the ILogListener interface and provides a default logging mechanism.

The LogWrapper class has three constructors:

- `LogWrapper()` - Creates a new LogWrapper object that writes tracing messages to System.out.
- `LogWrapper(String fileName)` - Creates a new LogWrapper object that writes trace messages to the file specified in fileName.
- `LogWrapper(String fileName, long iMaxSize, int iArchives, int iExpires, int iFlushIntervalMs)` - Creates a new LogWrapper object that traces to the file specified in fileName and sets all the tracing properties provided:
  - The maximum size of a single trace file (the default is 2048 Kb).
  - The maximum number of trace files before LoggerManager deletes the oldest file (the default is 4).

If a developer deploys an application and then wants to debug it in the field, they need a way to change the trace mask from the default level if necessary to provide more information for debugging.

You also need to provide a way to adjust the trace mask at runtime. If you encounter problems, Cisco personnel need to see this log file to assist you with your problem.

For more information about the LogWrapper class and its associated methods, see the Java CIL Javadoc file.
Logging and tracing (.NET)

The .NET CIL tracing mechanism behaves differently from that of the COM and C++ CILs. The .NET CIL does not automatically create a log file and trace to it. You must develop the custom application to create and maintain the log file.

The .NET CIL provides classes that allow you to write tracing messages from CTI applications. Custom applications can either create their own logging mechanism or use the default logging mechanism provided in the .NET CIL.

Default Logging Mechanism

You can use the .NET CIL LogWrapper class to implement logging to the system console or to a file. The LogWrapper class registers itself as an event listener and creates a log file.

Log Trace Events with LogWrapper Class

To log trace events using the LogWrapper class:

Procedure

Step 1

Create an instance of the LogWrapper class, passing the following arguments:

- `logFileName` - Name of file in which to write trace events.
- `fileMaxSize` - The maximum size of the log file.
- `numberArchivesFiles` - Maximum number of log files that can exist in the log file directory before the logging mechanism starts overwriting old files.
- `numberDaysBeforeFileExpired` - Maximum number of days before a log file is rolled over into a new log file regardless of the size of the file.

The following code snippet creates an instance of the LogWrapper class that writes trace events to MyLogFile.txt. When MyLogFile.txt reaches 2048 KB, a new log file is created. The Logger creates a maximum of 20 log files in the log file directory before overwriting existing files. After 10 days, the log file is rolled over into a new log file regardless of its size.

```csharp
// Create a LogWrapper. This will create a file and start // listening for log
// events to write to the file.
String logFileName = "MyLogFile.txt";
int fileMaxSize = 2048;
int numberArchivesFiles = 20;
int numberDaysBeforeFileExpired = 10;
m_logWrapper = new LogWrapper(logFileName, fileMaxSize, numberArchivesFiles,
numberDaysBeforeFileExpired);
```

Step 2

In your application, write trace events. The following example traces a message at the given trace level for the given method. Set the trace level to the desired trace mask. Trace masks are defined in the Logger class. For more information about available trace mask values, see the following table.
The CTI Toolkit Combo Desktop .NET sample application included with the CTI OS toolkit shows how to use the CIL's LogWrapper class in the context of a complex softphone application.

The following table lists the trace masks available in the .NET CIL.

Table 181: Trace Masks in .NET CIL

<table>
<thead>
<tr>
<th>TraceMask Bit</th>
<th>Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE_LEVEL_MAJOR</td>
<td>0x000000ff</td>
<td>Mask for major events.</td>
</tr>
<tr>
<td>TRACE_LEVEL_EVENT_REQ</td>
<td>0x0000ff00</td>
<td>Mask for general events and requests.</td>
</tr>
<tr>
<td>TRACE_LEVEL_METHOD</td>
<td>0x00ff0000</td>
<td>Mask for method entry and exit.</td>
</tr>
<tr>
<td>TRACE_LEVEL_MEMORY</td>
<td>0xff000000</td>
<td>Mask for very low level operations.</td>
</tr>
</tbody>
</table>

Individual Trace Mask

Lowest Order Byte Mask: Events

<table>
<thead>
<tr>
<th>TraceMask Bit</th>
<th>Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE_MASK_ALWAYS</td>
<td>0x00</td>
<td>Always print.</td>
</tr>
<tr>
<td>TRACE_MASK_CRITICAL</td>
<td>0x01</td>
<td>Critical error.</td>
</tr>
<tr>
<td>TRACE_MASK_WARNING</td>
<td>0x02</td>
<td>Warning.</td>
</tr>
<tr>
<td>TRACE_MASK_EVT_REQ_HIGH</td>
<td>0x04</td>
<td>High important events/requests.</td>
</tr>
<tr>
<td>TRACE_MASK_EVT_REQ_HIGH_PARM</td>
<td>0x08</td>
<td>High important events/requests.</td>
</tr>
<tr>
<td>TRACE_MASK_EVT_REQ_AVG</td>
<td>0x10</td>
<td>Average important events/requests.</td>
</tr>
<tr>
<td>TRACE_MASK_EVT_REQ_AVG_PARM</td>
<td>0x20</td>
<td>Average important events/requests.</td>
</tr>
<tr>
<td>TRACE_MASK_EVT_REQ_LOW</td>
<td>0x40</td>
<td>Low important events/requests.</td>
</tr>
<tr>
<td>TRACE_MASK_EVT_REQ_LOW_PARM</td>
<td>0x80</td>
<td>Low important events/requests.</td>
</tr>
</tbody>
</table>

Second Lowest Order Byte: Method Tracing

<table>
<thead>
<tr>
<th>TraceMask Bit</th>
<th>Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE_MASK_METHOD_HIGH</td>
<td>0x0100</td>
<td>High visibility method entry/exit trace.</td>
</tr>
<tr>
<td>TRACE_MASK_METHOD_HIGH_LOGIC</td>
<td>0x0200</td>
<td>High visibility method logic trace.</td>
</tr>
<tr>
<td>TraceMask Bit</td>
<td>Value</td>
<td>Purpose</td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>TRACE_MASK_METHOD_HIGH_LOGIC</td>
<td>0x0400</td>
<td>Internal visibility method entry/exit trace.</td>
</tr>
<tr>
<td>TRACE_MASK_METHOD_AVG_LOGIC</td>
<td>0x0800</td>
<td>Internal visibility method logic trace.</td>
</tr>
<tr>
<td>TRACE_MASK_METHOD_LOW</td>
<td>0x1000</td>
<td>Helper object visibility method entry/exit trace.</td>
</tr>
<tr>
<td>TRACE_MASK_METHOD_LOW_LOGIC</td>
<td>0x2000</td>
<td>Helper object visibility method logic trace.</td>
</tr>
<tr>
<td>TRACE_MASK_METHOD_MAP</td>
<td>0x4000</td>
<td>Map access.</td>
</tr>
</tbody>
</table>

**Highest Order Byte: Communications and Processing**

<table>
<thead>
<tr>
<th>TraceMask Bit</th>
<th>Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE_MASK_ARGS_METHODS</td>
<td>0x01000000</td>
<td>Method entry/exit for Arguments objects.</td>
</tr>
<tr>
<td>TRACE_MASK_ARGS_LOGIC</td>
<td>0x02000000</td>
<td>Logic trace for Arguments objects.</td>
</tr>
<tr>
<td>TRACE_MASK_PACKETS_METHODS</td>
<td>0x04000000</td>
<td>Method entry/exit for packets objects.</td>
</tr>
<tr>
<td>TRACE_MASK_PACKETS_LOGIC</td>
<td>0x08000000</td>
<td>Logic trace for packets objects.</td>
</tr>
<tr>
<td>TRACE_MASK_SERIALIZE_DUMP</td>
<td>0x10000000</td>
<td>Memory dump of serialize buffer.</td>
</tr>
<tr>
<td>TRACE_MASK_SOCKETS_DUMP</td>
<td>0x20000000</td>
<td>Memory dump of sockets buffer.</td>
</tr>
<tr>
<td>TRACE_MASK_THREADING</td>
<td>0x40000000</td>
<td>Threading tracing on or off.</td>
</tr>
<tr>
<td>TRACE_MASK_CONNECTION</td>
<td>0x80000000</td>
<td>Connection tracing on or off.</td>
</tr>
</tbody>
</table>

**Custom Logging Mechanism**

The LogManager class within the .NET CIL implements all CIL logging functions. This singleton class has only one instance of LogManager, which provides a global point of access. The LogManager object defines a LogEventHandler delegate that custom applications must implement:

```csharp
public delegate void LogEventHandler(object eventSender, LogEventArgs args);
```

**Log Trace Events with Logger Class**

To log trace events from a custom application to a file, perform the following steps:
Procedure

Step 1
Create a Logger object. For example:

```csharp
m_log = new Logger();
```

Step 2
Write a method to handle log events. This method can trace the log events to a file, if desired. For example:

```csharp
public virtual void ProcessLogEvent(Object eventSender, LogEventArgs Evt)
{
    String traceLine = Evt.Description;
    // Check that tracing is enabled for this tracelevel
    if ( m_logger.IsTraceMaskEnabled(traceLevel) )
    {
        WriteTraceLineToFile(traceLine);
    }
}
```

Step 3
Create a log listener to handle trace events. In the following example, the AddLogListener method registers the LogEventHandler delegate as a listener for trace events. The LogManager sends trace events to the method that you pass to the LogEventHandler.

In the following example, the LogManager sends trace events to the ProcessLogEvent method created in Step 2.

```csharp
m_log.AddLogListener(new LogManager.LogEventHandler(ProcessLogEvent));
```

Note
The LogManager only calls the method passed as a parameter to the LogEventHandler for a particular trace if the trace level for that trace is enabled. You can use the IsTraceMaskEnabled method in the Logger class to determine whether or not a trace level is enabled.

Trace Configuration (Java and .NET)

For the Java and .NET CILs, you can configure tracing either programatically by using the LogWrapper class or by editing the TraceConfig.cfg file. Settings in TraceConfig.cfg do not take effect until LogWrapper.ProcessConfigFile is called. Your application must call ProcessConfigFile if you have edited the configuration settings in the TraceConfig.cfg file.

The All Agents Sample .NET code in the .NET CIL includes a sample TraceConfig.cfg file and shows you how to process that file.

Log file configuration settings are defined as follows:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Optimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumberDaysBeforeFileExpired</td>
<td>Maximum number of days before a log file is rolled over into a new log file regardless of the size of the file.</td>
<td>1</td>
</tr>
<tr>
<td>NumberArchivesFiles</td>
<td>Maximum number of log files that may exist in the log file directory before the logging mechanism starts overwriting old files.</td>
<td>5</td>
</tr>
<tr>
<td>FileMaxSize</td>
<td>Maximum size of a log file in kilobytes. When a log file reaches the maximum size, a new log file is created.</td>
<td>2048</td>
</tr>
<tr>
<td>TraceMask</td>
<td>Bit mask that determines the categories of events that are traced.</td>
<td>0x40000307</td>
</tr>
</tbody>
</table>