CTI OS Developer's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted

Release 8.0(1)

February 2010

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

CONTENTS

About This Guide

Purpose

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.

Organization

The manual is divided into the following chapters.

Chapter	Description
Chapter 1, "Introduction"	Provides an overview of CTI and the CTI OS Client Interface.
Chapter 2, "CTI OS Client Interface Library Architecture"	Discusses CTI OS architecture.
Chapter 3, "CIL Coding Conventions"	Explains how to build an application using the CTI OS libraries.
Chapter 4, "Building Your Application"	Discusses how to build your custom CTI application to use the CTI OS Client Interface Library
Chapter 5, "CTI OS ActiveX Controls"	Describes the CTI OS softphone controls and explains how to use them in a VB or COM container.
Chapter 6, "Event Interfaces and Events"	Describes the CTI OS event interfaces.

Chapter	Description
Chapter 7, "CtiOs Object"	Discusses features common to all CTI OS objects derived from CtiOsObject.
Chapter 8, "Session Object"	Describes the methods associated with the CTI OS Session object.
Chapter 9, "Agent Object"	Describes the methods associated with the CTI OS Agent object.
Chapter 10, "Call Object"	Describes the methods associated with the CTI OS Call object.
Chapter 11, "SkillGroup Object"	Describes the methods associated with the CTI OS SkillGroup object.
Chapter 12, "Helper Classes"	Describes the methods associated with the CTI OS Arguments classes.
Chapter 13, "SilentMonitorManager Object"	Describes the methods associated with the CTI OS Silent Monitor Manager object.
Appendix A, "CTI OS Keywords and Enumerated Types"	Discusses CTI OS keywords and enumerated types.
Appendix B, "CTI OS Logging"	Discusses a few issues related to CTI OS logging.
Appendix C, "Migrating From CTI OS 7.1(x) or 7.2(x)"	Describes the migration steps and the new parameters required to build the application.

Related Documentation

Documentation for Cisco Unified ICM/Unified Contact Center Enterprise & Hosted (Unifed ICM/CCE/CCH), as well as related documentation, is accessible from Cisco.com at http://www.cisco.com/cisco/web/psa/default.html?mode=prod.

- Related documentation includes the documentation sets for Cisco CTI Object Server (CTI OS), Cisco Agent Desktop (CAD), Cisco Agent Desktop - Browser Edition (CAD-BE), Cisco Unified Contact Center Management Portal, Cisco Unified Customer Voice Portal (Unified CVP), Cisco Unified IP IVR, Cisco Support Tools, and Cisco Remote Monitoring Suite (RMS).
- For documentation for these Cisco Unified Contact Center Products, go to http://www.cisco.com/cisco/web/psa/default.html?mode=prod click on **Voice and Unified Communications**, then click on **Cisco Unified Contact Center Products** or **Cisco Unified Voice Self-Service Products**, then click on the product/option you are interested in.
- For troubleshooting tips for these Cisco Unified Contact Center Products, go to http://docwiki.cisco.com/wiki/category:Troubleshooting, then click the product/option you are interested in.
- Also related is the documentation for Cisco Unified Communications Manager, which can also be accessed fromhttp://www.cisco.com/cisco/web/psa/default.html?mode=prod

- Technical Support documentation and tools can be accessed from http://www.cisco.com/en/US/support/index.html
- The Product Alert tool can be accessed through (login required) http://www.cisco.com/cgi-bin/Support/FieldNoticeTool/field-notice

Conventions

This manual uses the following conventions.

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

Obtaining Documentation and Submitting a Service Request

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

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

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

Documentation Feedback

You can provide comments about this document by sending email to the following address:

mailto:ccbu_docfeedback@cisco.com

We appreciate your comments.

CHAPTER 1

Introduction

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

Introduction to CTI

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

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

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

What is a CTI-Enabled Application?

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

Screen Pop

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

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

Agent State Control

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

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

Third-Party Call Control

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

For example, a financial services application might perform the transfer of a phone call to a speed-dial number designated by the application itself. In this scenario, the agent could 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.

Leveraging CTI Application Event Flow

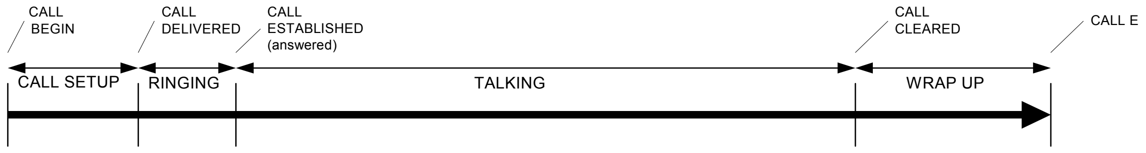
The first step to developing a CTI-enabled application is to understand the events and requests that are at play within the CTI environment. Asynchronous events are messages sent to applications that indicate an event to which the application can respond (for example, CallBeginEvent). Requests are the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

Asynchronous Events

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

Figure 1-1 depicts the stages of a typical inbound telephone call and its associated events:

Figure 1-1 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 Figure 1-1, the following are typical call events used for CTI applications:

- 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.

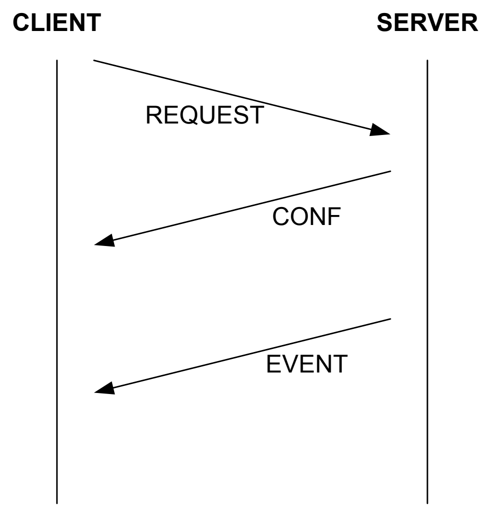
The foregoing is only a brief sample of the events available via CTI OS. The complete set of events available for CTI developers is detailed in later chapters in this guide.

Request-Response Paradigm

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

The typical request-response flow for CTI uses the model shown in Figure 1-2:

Figure 1-2 Sample Request-Response Message Flow.

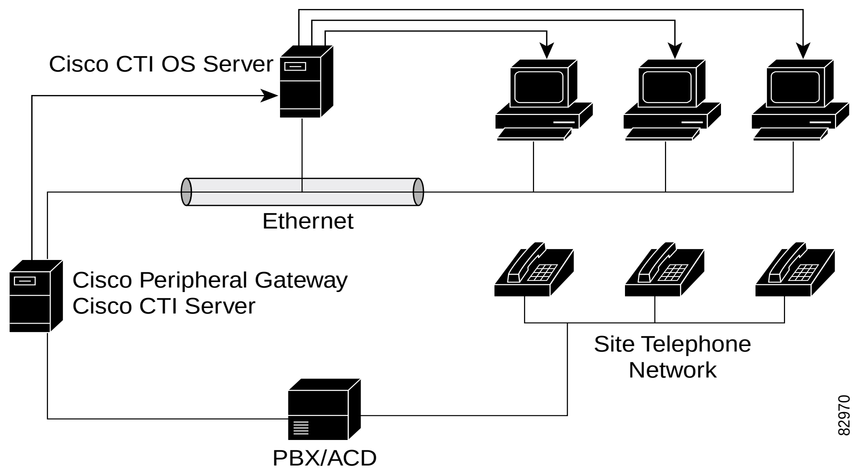


A request generated by the CTI-enabled application (CLIENT) is sent to the CTI service (SERVER), and a response message (CONF) is generated to indicate that the request has been received. In most cases if the request is successful, a follow-on event will be received indicating that the desired behavior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

Overview of CTI OS

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

Figure 1-3 CTI OS Three-Tiered Architecture Topology



The CTI OS application architecture employs three tiers:

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

Advantages of CTI OS as 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. CTI OS enables developers to create a screen pop application in as little as five minutes. The only custom-development effort required is within the homegrown application to which CTI is being added.
- **Complex solutions made simple**. CTI OS enables complex server-to-server integrations and multiple agent monitoring-type applications. The CIL provides a single object-oriented interface that can be used in two modes: agent mode and monitor mode. See Chapter 2, "CTI OS Client Interface Library Architecture" for an explanation of these two modes.
- **Fault tolerant.** CTI OS is built upon the 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 required to rapidly develop high-quality CTI enabled applications, taking advantage of the rich features of the CTI OS server. Every feature of CTI OS was designed with ease of integration in mind, to remove the traditional barriers to entry for CTI integrations.

- **Object-oriented interactions.** CTI OS provides an object-oriented CTI interface by defining objects for all call center interactions. Programmers interface directly with Session, Agent, SkillGroup, and Call objects to perform all functions. CIL objects are thin proxies for the server-side objects, where all the 'heavy-lifting' is done. The Session object manages all objects within the CIL. A UniqueObjectID identifies each object. Programmers can access an object by its UniqueObjectID or by iterating through the object collections.
- **Connection and session management.** The CTI OS CIL provides out-of-the-box connection and session management with the CTI OS Server, hiding all of the details of the TCP/IP sockets connection. The CIL also provides an out-of-the-box failover recovery: upon recovery from a failure, the CIL will automatically reconnect to another CTI OS (or reconnect to the same CTI OS after restart), re-establish the session, and recover all objects for that session.
- **All parameters are key-value pairs.** The CTI OS CIL provides helper classes to treat all event and request parameters as simply a set of key-value pairs. All properties on the CTI OS objects are accessible by name via a simple Value = GetValue("key") mechanism. Client programmers can add values of any type to the CTI OS Arguments structure, using the enumerated CTI OS keywords, or their own string keywords (for example, AddItem("DialedNumber", "1234")). This provides for future enhancement of the interface without requiring any changes to the method signatures.
- **Simple event subscription model.** The CTI OS CIL implements a publisher-subscriber design pattern to enable easy subscription to event interfaces. Programmers can subscribe to the appropriate event interface that suits their needs, or use the IAllInOne interface to subscribe for all events. C++ and COM contain subclassable event adapter classes. These classes enable programmers to subscribe to event interfaces; they only add minimal custom code for the events they use and no code at all for events they do not use.

Illustrative Code Fragments

Throughout this manual, illustrative code fragments are provided both to clarify usage and as examples. These fragments are written in several languages, including C++ and Visual Basic (VB). Though .NET (and therefore VB .NET) is supported, note that the VB code fragments are written using **VB 6 syntax.**

CHAPTER 2

CTI OS Client Interface Library Architecture

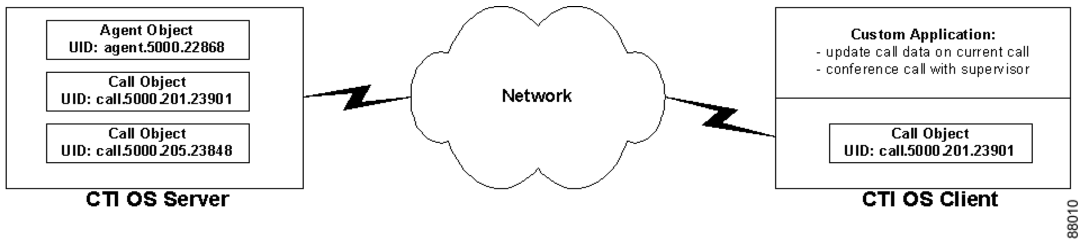
This chapter describes the architecture of the CTI OS Client Interface Library (CIL). The CIL is the programmer's interface into the CTI OS system.

Object Server Architecture

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

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

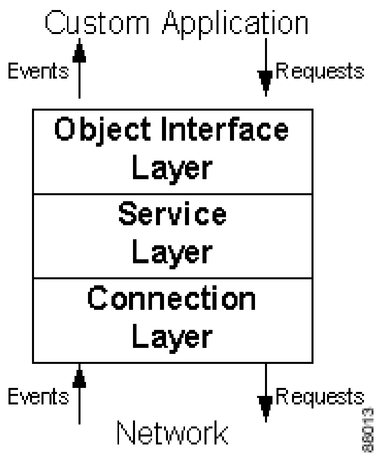
Figure 2-1 CTI OS Object Server and Client Object Sharing



Client Interface Library Architecture

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

Figure 2-2 Client Interface Library Three-Tiered Architecture



Connection Layer

The Connection layer provides basic communication and connection recovery facilities to the CIL. It creates the foundation, or bottom tier of the CIL's layered architecture, and decouples the higher-level event and message architecture from the low-level communication link (TCP/IP sockets). The Connection layer sends and receives 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 only when the connection has been silent for some period of time to detect network-level failures.

Service Layer

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

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

Object Interface Layer

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

Custom Application

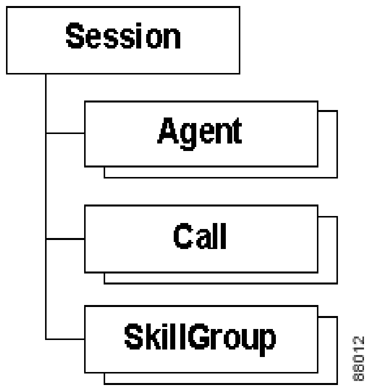
The custom application is the business application that is developed to integrate with the CTI OS Client Interface Library. The custom application makes use of the CIL in two ways. One, the CIL provides the object-based interface for interacting with CTI OS, to send requests for agent and call control.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

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

CIL Object Model

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

Figure 2-3 The CIL Object Model Object Interfaces



Session Object

The Session object is the main object in the CIL. It controls the logical session between the client application and the 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 of the other objects in the CIL.

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

Session Modes

A Session object can be set to work in one of two modes: Agent Mode or Monitor Mode. 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. Once agent mode has been set, the CIL receives the events for the specified agent, as well as all call events for that agent's calls. If the agent is also configured as a Supervisor in 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 details of setting up a monitor mode connection, refer to the section in Chapter 4, "How to Select Monitor Mode".

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

Agent Object

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

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

Call Object

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

For the complete interface specification of the Call object, see Chapter 10, "Call Object."

SkillGroup Object

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

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

Object Creation and Lifetime

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

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

An Agent, Call or SkillGroup object is created (by the Session) when the Session receives an event for an object (identified by its UniqueObjectID) that is not yet present at the CIL. This ensures that the CIL will always have the appropriate collection of proxy objects, one for each object on the 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 Chapter 8, "Session Object."

Reference Counting

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

An application or object that will hold a reference to a CIL object must add to its reference count using the AddRef method. When the reference is no longer required, the application or object holding that reference must decrement the reference count using the Release() method. Reference counting is discussed further in Chapter 7, "CtiOs Object."

Note Reference counting must be done explicitly in C++ applications (COM or non-COM). Visual Basic, Java, and the .NET frameworkwill 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 will be created in response to the OnCallBegin event, but in certain failover recovery scenarios a Call object could be created in response to an OnSnapshotCallConf event. Any call data available for the call is passed in the event, and is used to set up the Call object's initial state and properties.

The Call object will remain valid at the CIL until the receipt of the OnCallEnd event. When the OnCallEnd event is received, the Session object will publish the event to any subscribers to the event interfaces. Applications and objects must release any remaining references to the Call object within their event handler for OnCallEnd to allow the Call object to be properly deleted. When the Call object's OnEvent method returns after handling OnCallEnd, the Session will check the reference count for zero; if any references remain, the call object will be removed from the call object collection but will not be 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.

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

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

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

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

SkillGroup Object Lifetime

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

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

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 2-1 SessionLib (C++)

Object Name	Method Name	Explanation
CAgent	GetSkillGroups(), GetMonitoredCall()	The client application must call Release() on the returned object when the object is no longer needed.
CILRefArg	CreateInstance(), GetValue()	The client application must call Release() on the returned object when the object is no longer needed.
CILRefArg	SetValue(), operator=	These methods increment the reference count on the passed in object. When the CilRefArg is deleted the reference count of the enclosed object will be decremented.

Table 2-1 SessionLib (C++)

Object Name	Method Name	Explanation
CCtiOsSession	SetCurrentCall()	This method increments the reference count on the passed in object. The previous "current" 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.
CCtiOsSession	DestroyWaitObject()	This method call decrements the reference count on the passed in object.
CCtiOsSession	CreateWaitObject()	The client application must call DestroyWaitObject() on the returned object when the object is no longer needed.
CCtiOsSession	DestroySilentMonitor Manager()	This method decrements the reference count of the passed in object.
CCtiOsSession	CreateSilentMonitor Manager()	The client application must call DestroySilentMonitorManager () on the returned object when it is no longer needed.
CCtiOsSession	SetCurrentSilent Monitor()	This method increments the reference count on the passed in object. The previous "current" silent monitor's reference count is decremented.
CCtiOsSession	GetCurrentCall(), GetCurrentSilent MonitorManager(), GetAllCalls(), GetAllSkillGroups(), GetAllAgents(), GetCurrentAgent(), GetValue(), GetObjectFromObject ID()	The client application must call Release() on the returned object when it is no longer needed.

Table 2-1 SessionLib (C++)

Object Name	Method Name	Explanation
CCtiOsSession	SetAgent()	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.
CSilentMonitor Manager	GetSessionInfo(), GetIPPhoneInfo(), GetSMSessionList()	The client application must call Release() on the returned object when it is no longer needed.

Table 2-2 CtiosClient.dll (COM)

Object Name	Method Name	Explanation
IAgent	GetSkillGroups()	This method increments the reference count for every SkillGroup object, adds them to a safe array and then returns the safe array.
IAgent	GetMonitoredAgent(), GetMonitoredCall()	The client application must call Release() on the returned object when it is no longer needed.
IAgent	GetValue(), GetValueArray(), GetElement()	The client application must call Release() on the returned object (second argument) when it is no longer needed.
IAgent	GetAllProperties()	The client application must call Release() on the returned object (first argument) when it is no longer needed.
ISkillGroup	GetValue(), GetValueArray(), GetElement()	The client application must call Release() on the returned object (second argument) when it is no longer needed.
ISkillGroup	GetAllProperties()	The client application must call Release() on the returned object (first argument) when it is no longer needed.
ICall	GetCallContext(), GetCallData()	The client application must call Release() on the returned object when it is no longer needed.

Table 2-2 CtiosClient.dll (COM)

Object Name	Method Name	Explanation
ICall	GetValue(), GetValueArray(), GetElement()	The client application must call Release() on the returned object (second argument) when it is no longer needed.
ICall	GetAllProperties()	The client application must call Release() on the returned object (first argument) when it is no longer needed.
ISilentMonitorManager	SetMonitor()	This method increments the reference count of the passed in object and decrements the reference count of the previous monitor.
ISilent MonitorManager	GetMonitor()	The client application must call Release() on the returned object when it is no longer needed.
ISilent MonitorManager	GetSessionInfo(), GetIPPhoneInfo(), GetSMSessionList(), GetValue(), GetValueArray(), GetElement()	The client application must call Release() on the returned object (second argument) when it is no longer needed.
ISilentMonitorManager	GetAllProperties()	The client application must call Release() on the returned object (first argument) when it is no longer needed.
ISession	SetAgent()	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.
ISession	GetCurrentAgent(), GetCurrentCall()	The client application must call Release() on the returned object when it is no longer needed.
ISession	GetAllCalls()	This method increments the reference count for every Call object, adds them to a safe array and then returns the safe array.
ISession	GetAllAgents()	This method increments the reference count for every Agent object, adds them to a safe array and then returns the safe array.

Table 2-2 CtiosClient.dll (COM)

Object Name	Method Name	Explanation
ISession	GetAllSkillGroups()	This method increments the reference count for every SkillGroup object, adds them to a safe array and then returns the safe array.

ISession	GetValue()
	GetValueArray(),
	GetElement()
		The client application must call Release() on the returned object (second argument) when it is no longer needed.

ISession	GetAllProperties()	The client application must call Release() on the returned object (first argument) when it is no longer needed.
ISession	GetObjectFromObject ID()	The client application must call Release() on the returned object (second argument) when it is no longer needed.
ISession	CreateSilentMonitor Manager()	The client application must call DestroySilentMonitorManager() on the returned object when it is no longer needed.
ISession	DestroySilentMonitor Manager()	This method call decrements the reference count on the passed in object.
ISession	GetCurrentSilent MonitorManager()	The client application must call Release() on the returned object when it is no longer needed.

Table 2-3 CtiosComArguments.dll (COM)

Object Name	Method Name	Explanation
IArg	Clone()	The client application must call Release() on the returned object when it is no longer needed.
IArg	GetValueArray()	The client application must call Release() on the returned object when it is no longer needed.
IArg	GetValue()	If ARG_TYPE = ARG_ARRAY, the client application must call Release() on the returned object when it is no longer needed.

Table 2-3 CtiosComArguments.dll (COM)

Object Name	Method Name	Explanation
IArguments	GetValueArray(), GetValue(), GetElement()	The client application must call Release() on the returned object (second argument) when it is no longer needed.
IArguments	Clone()	The client application must call Release() on the returned object when it is no longer needed.

Table 2-4 ArgumentsLib (C++)

Object Name	Method Name	Explanation
Arg	CreateInstance(), GetValueArray(), operator=	The client application must call Release() on the returned object when it is no longer needed.
Arguments	CreateInstance(), Clone(), GetValue(), GetValueArg, GetValueArray(), GetElement(), GetElementArg()	The client application must call Release() on the returned object when it is no longer needed.
Arguments	SetValue()	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.
Arguments	SetElement()	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.

Where To Go From Here

Subsequent chapters in this manual contain the following information:

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

CHAPTER 3

CIL Coding Conventions

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

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

This chapter discusses the following topics:

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

CTI OS CIL Data Types

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

- **STRING**: 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 will 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.

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

Table 3-1 CTI OS CIL Data Type

Documented Data Type	STRING	INT	UNSIGNED INT	SHORT	UNSIGNED SHORT	BOOL	ARGUMENTS	ARG
C++ Type	std::string or const char	long or int	unsigned int	short	unsigned short	bool	Arguments	Arg
Visual Basic 6.0 Type	String	Long	None	Integer	Integer	Boolean	Arguments	Arg
COM Type	BSTR	long or int	unsigned int	short	unsigned short	VARIANT _BOOL	IArguments *	IArg*
Java Type	String	int	long	short	int	Boolean	Arguments	Arg
.NET Type	System.String	System .Int32	System.Int64	System.Int16	System.Int32	System. Boolean	Arguments	Arg

Asynchronous Program Execution

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

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

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

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

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

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

A common mistake: developers who have not previously programmed with asynchronous events might mistake the error code returned from a method call for the actual result of the request. The correct semantics are to interpret the error code as being indicative of 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 Table 3-2.

Note The numeric values listed in Table 3-2 are subject to change. It is recommended that you use the error code enumerations to check a given error code, rather than rely on a specific numeric value.

Table 3-2 CIL Error Codes

CIL Error Code	Numeric Value	Description
CIL_OK	1	The method succeeded. The request to silent monitor the call was successfully initiated.
CIL_FAIL	0	The method failed.
E_CTIOS_METHOD_NO_ IMPLEMENTED	-99	There is no implementation available for this method.
E_CTIOS_INVALID_ PROPERTY	-100	One or more properties are invalid.
E_CTIOS_MODE_CONFLICT	-101	A conflict when setting session mode.
E_CTIOS_INVALID_ EVENTID	-102	The Event ID is not valid.
E_CTIOS_INVALID_ ARGUMENT	-103	The Argument is not valid.

CIL Error Code	Numeric Value	Description
E_CTIOS_INVALID_ SESSION	-104	The Session is not valid.
E_CTIOS_UNEXPECTED	-105	An unexpected error has occurred.
E_CTIOS_OBJ_ALLOCATION_ FAILED	-106	There is not enough memory available and the creation of CCtiOsObject failed.
E_CTIOS_ARRAYREF_ ALLOCATION_FAILED	-107	There is not enough memory available and an creation of an array of references to objects of type CCtiOsObject failed.
E_CTIOS_ARGUMENT_ ALLOCATION_FAILED	-108	There is not enough memory available and the creation of an object of type Arguments failed.
E_CTIOS_TARGET_ OBJECT_ NOT_FOUND	-109	There are no CTI OS Objects capable of processing an incoming event.
E_CTIOS_PROP_ ATTRIBUTES_ACCESS_ FAILED	-110	An error occurred while accessing a property's attributes, System may be running out of memory.
E_CTIOS_INVALID_ OBJECT_TYPE	-111	The object type is not one of the following predefined types CAgent, CCall, CSkillGroups, or CWaitObject.
E_CTIOS_INVALID_AGENT	-112	No valid agent.
E_CTIOS_INVALID_CALL	-113	No valid call.
E_CTIOS_IN_FAILOVER	-114	The session is recovering from a connection failure and had started the Fail Over procedure.
E_CTIOS_INVALID_ DESKTOP_TYPE	-115	Indicates that the desktop type specified in the request for DeskSettings download is neither Agent or Supervisor.
E_CTIOS_MISSING_ ARGUMENT	-116	Missing a required argument.
E_CTIOS_CALL_NOT_ON_ HOLD	-117	Call is not on hold.
E_CTIOS_CALL_ALREADY_ ON_HOLD	-118	Call is already on hold.
E_CTIOS_CALL_NOT_ ALERTING	-119	Call is not in alert state, it can not be answered.
E_CTIOS_AGENT_NOT_ LOGIN	-120	Agent is not logged in.
E_CTIOS_INVALID_ METHOD_PARAMETER	-121	The input parameter is invalid.
E_CTIOS_UNKNOWN	-122	The cause of this error is unknown.
E_CTIOS_OUT_OF_ MEMORY	-123	Failed to allocate new memory.
E_CTIOS_PORT_ UNAVAILABLE	-124	The specified port is not available for use.
E_CTIOS_SM_SESSION_ TERMINATED_ ABNORMALLY	-125	The Silent Monitor session was terminated abnormally.
E_CTIOS_SM_REJECTED_ ALREADY_IN_SESSION	-126	The request was rejected because there is an active silent monitor session in progress.

CIL Error Code	Numeric Value	Description
E_CTIOS_SM_PACKET_ SNIFFER_NOT_INSTALLED	-127	The packet sniffer is not present in the system; verify installation.
E_CTIOS_PACKET_ SNIFFER_FAILED	-128	An error occurred in the packet sniffer.
E_CTIOS_SOCKET_CALL_ FAILED	-129	A CTI OS socket call failed.
E_CTIOS_MEDIA_ TERMINATION_NOT_ INSTALLED	-130	EVVBU Media Termination component in the system, verify installation.
E_CTIOS_MT_UNKNOWN_ CODEC	-131	Specified CODEC is not supported.
E_CTIOS_MEDIA_ TERMINATION_FAILED	-132	An error occurred in the Media Termination Packet Decoder.
E_CTIOS_SNIFFER_NO_ PACKETS_RECEIVED	-133	The Sniffer has not received any IP packets.
E_CTIOS_SNIFFER_ FAILED_TO_OPEN_DEVICE	-134	The Sniffer failed to open the networking device.
E_CTIOS_SNIFFER_ FAILED_TO_SET_FILTER	-135	The Sniffer failed when setting the packet filter.
E_CTIOS_ERROR_IN_ PACKET_FILTER	-136	The packet filter expression is incorrect.
E_CTIOS_INVALID_ MONITORED_IP_ADDRESS	-137	The IP Address specified for the monitored device (IP Phone) is not valid.
E_CTIOS_INVALID_ SNIFFER_OBJECT	-138	Invalid Sniffer object.
E_CTIOS_INVALID_ DECODER_OBJECT	-139	Invalid Decoder object.
E_CTIOS_NO_SM_ SESSION_IN_PROGRESS	-140	There are no Silent Monitor Sessions in progress.
E_CTIOS_INVALID_ SILENT_MONITOR_ SESSION	-141	The specified Silent Monitor session does not exist.
E_CTIOS_FAILED_ REMOVING_SILENT_ MONITOR_SESSION	-142	Silent Monitor Session was not removed from the collection.
E_CTIOS_IP_PHONE_ INFORMATION_NOT_ AVAILABLE	-143	There is no information available about the IP Phone.
E_CTIOS_PEER_NOT_ ENABLED_FOR_SILENT_ MONITOR	-144	The peer application is not enabled for Silent Monitor.
E_CTIOS_NOT_ENABLED_ FOR_SILENT_MONITOR	-145	This application is not enabled for Silent Monitor.

CIL Error Code	Numeric Value	Description
E_CTIOS_NO_PENDING_REQUEST	-146	There are no pending requests to be processed.
E_CTIOS_ALREADY_IN_SESSION	-147	There is already an established session.
E_CTIOS_MODE_SET_ALREADY	-148	The session mode has already been set.
E_CTIOS_MODE_NOT_SET	-149	The session mode is not set yet.
E_CTIOS_INVALID_OBJECT_STATE	-150	Indicates that the object is not in the correct state.
E_CTIOS_INVALID_SILENT_MONITOR_MODE	-151	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.
E_CTIOS_COM_OBJ_ ALLOCATION_FAILED	-200	CoCreateInstance failed to create a COM object wrapper for a CIL Object (Session, Agent, Call, Skill, etc.).
E_CTIOS_COM_ CORRUPTED_REGISTRY	-201	A COM component failed to access data from the registry.
E_CTIOS_COM_DIALPAD_ FAIL_TO_LOAD	-202	The Dial Pad common dialog was not created and CoCreateInstance failed.
E_CTIOS_COM_CONV_ COMPTR_TO_CPPPTR_ FAILED	-203	Failed converting COM pointer to C++ pointer.
E_CTIOS_COM_NOT_ INITIALIZED	-204	The MS COM library is not initialized. Invoke CoInitialize(...).
E_CTIOS_SESSION_ DISCONNECT_PENDING	-300	A disconnect is already pending.
E_CTIOS_SESSION_NOT_ CONNECTED	-301	The session is not connected.
E_CTIOS_SESSION_NOT_DISCONNECTED	-351	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 have completed.
E_CTIOS_AGENT_ ALREADY_IN_SESSION	-900	An object for this agent already exists in the session.
E_CTIOS_SET_AGENT_ SESSION_DISCONNECT_ REQUIRED	-901	Session must be disconnected before operation.
E_CTIOS_SERVICE_SEND_MESSAGE_FAILED	-902	Could not send message. Session may not be connected.
E_CTIOS_CALL_ALREADY_CURRENT_IN_SESSION	-903	An object for this call is already set as current in the session.
E_CTIOS_LOGIN_ INCONSISTENT_ ARGUMENTS	-904	The AgentID and/or PeripheralID provided to a Login call do not match the properties set on the Agent object when SetAgent() was called.

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

COM Error Codes

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

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

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

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

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

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

Table 3-3 COM Error Codes

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

Generic Interfaces

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

Arguments

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

Example of using Arguments in a Visual Basic MakeCall request:

```
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:

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

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

Accessing Properties and Parameters with GetValue

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

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

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

//Java
```

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

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

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

Setting Object Properties and Request Parameters with SetValue

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

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

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

'Visual Basic
Dim a As Agent
a.SetValue "AgentID", "22866"

//Java. Note use of the CTIOS_AGENTID version of keywords.
String  sAgentID = "22866";
Args.SetValue("AgentID", sAgentID);
Args.SetValue(CtiOs_IkeywordIDs.CTIOS_AGENTID, sAgentID);  // alternative
Args.SetValue(ekwAgentID, sAgentID);
```

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

UniqueObjectID

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

The UniqueObjectID is a variable-length string which can uniquely identify the object within the current context of the CTI OS Server and the Unified ICME and CTI Interlink Advanced. The UniqueObjectID is composed of an object type (e.g. call, agent, skillgroup, etc.), and two or more additional identifying fields. Table 3-4 explains the composition of the UniqueObjectID.

Table 3-4 UniqueObjectID Components

Object Type	Sample UniqueObjectID	Explanation
Call Object	call.5000.202.23901	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).
Agent Object	agent.5000.22866	The agent object is uniquely identified by its PeripheralID (5000, generated by Unified ICM), and its agent ID.
Device Object (for events only; no CIL object)	device.5000.23901	The device object is uniquely identified by its PeripheralID (5000, generated by Unified ICM), and its instrument number (configured by the PBX/ACD).
SkillGroup Object	skillgroup.5000.77	The skill group object is uniquely identified by its PeripheralID (5000, generated by Unified ICM), and its SkillGroupNumber (configured by the PBX/ACD).
Team Object (for events only; no CIL object)	team.5000.5001	The team object is uniquely identified by its PeripheralID (5000, generated by Unified ICM), and its TeamID (5001, also generated by Unified ICM).

Note The CTI OS UniqueObjectID is not the same as the Unified ICM globally unique 64 bit key used in the Unified ICME historical databases (herein called the ICMEnterpriseUniqueID), which exists only for calls. The ICMEnterpriseUniqueID stays with the call even when the call is transferred between call center sites, whereas the UniqueObjectID for a call is specific to its site (by PeripheralID, ConnectionCallID, and ConnectionDeviceID).

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

“`icm.routercallkeyday.routercallkeycallid`”

where routercallkeyday is the field Day in the 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 seek to match the call data with the arrival of an eCallBeginEvent.

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

- eCallBeginEvent
- eCallDataUpdateEvent
- eSnapshotCallConf
- eCallEndEvent

Obtaining Objects from UniqueObjectIDs

Client applications written to take advantage of the CIL can use the UniqueObjectID to obtain a pointer (in C++ or COM for C++) or a reference (in 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. Since this mechanism is generic, and does not contain specific information about the object type retrieved, the pointer (or reference) returned is a pointer or reference to the base class: a CCtiosObject* in C++, an Object in Visual Basic, an IDispatch* in COM for C++, or CtiOsObject in .NET and Java.

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

C++ example:

```
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:

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

Using Button Enablement Masks

The CTI OS Server provides a rich object-level interface to the CTI interactions of the contact center. One of the features the CTI OS Server provides is to evaluate all of the telephony events, and map them to the features permitted by the 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 time when it is valid to answer a call is when the ENABLE_ANSWER bit in the enablement mask is set to the on position. The following C++ example depicts this case:

```
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:

```
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 OS Server. This allows future new features to be enabled, and software bugs to be corrected in a central location, which is a great benefit for deploying future releases.

Warning **The button enablement feature is intended to be used in agent mode applications and not for monitor mode applications.**

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

CHAPTER 4

Building Your Application

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

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

- **ActiveX Controls**. This section covers using the CTI OS ActiveX controls in a COM container such as Visual Basic.
- **COM CIL in C++.** This section covers the steps required to use the CIL's COM components in a Microsoft Visual C++ application.
- **C++ CIL using static libraries.** This section covers the steps required to reference the CIL's C++ classes in your application, and how to link the C++ static library files into a Microsoft Visual C++ application.
- **Java CIL libraries**. This section covers considerations for installing and using the Java CIL libraries.
- **.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.

Setting Up Your Environment for .NET

Cisco CTI OS Toolkit 8.0(1) introduces support for application development targeting the Microsoft .NET Framework 2.0 and Microsoft Visual Studio .NET 2005. Cisco CTI OS Toolkit 8.0(1) provides a native .NET class library (.NET CIL) and runtime callable wrappers for COM CIL and the CTI OS ActiveX controls. The CTI OS Toolkit 8.0(1) consist of a set of production ready desktops, and five software development kits.

The .NET CIL and the runtime callable wrappers (RCWs) are installed in the Global Assembly Cache (GAC) by the setup program such that all the components are available to any of the sample included in the toolkit and any new application in development. In the recommended environment settings for building .NET applications using the CTI OS toolkit, however, there are additional configuration steps 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 and the CTI OS Toolkit Outbound Option Desktop, 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 Visual Studio.NET 2005 (VS.2005).

Integrating with Microsoft Visual Studio .NET 2005

Visual Studio 2005 offers a wider spectrum of development possibilities and an advanced design experience. In addition, with Service Pack 1 it also provides:

- Windows Vista , native application development

  The CTI OS Toolkit 8.0(1) focuses on satisfying Vista's fundamentals technical pillar as follows:

  – C++/COM/.NET CIL, CTI OS ActiveX Controls, build targets (DLLs, EXEs) include built-in manifest information.
  – C++/COM/.NET CIL, CTI OS ActiveX Controls, build targets (DLLs, EXEs) include Cisco's digital signature & Certificate.
  – Generates *.PDB files for postmortem debugging for any of the following components: C++/COM/.NET CIL, CTI OS ActiveX.
  – C++/COM CIL, CTI OS ActiveX Controls, use of secure CRT and ATL functions to increase code security and reduce memory overrun vulnerabilities.
  – .NET CIL 8.0(1) is implemented as set native .NET Framework 2.0 assemblies.
  – .NET CIL incorporates the new specification in the CLI .2005 and CLR 2.0.
  – New installer program for CTI OS Toolkit 8.0(1) that includes following features:

    - Detection of .NET Framework 2.0 CLR and installation on demand (if required).

    - Secure installation of production ready CTI OS desktops and auto configuration.

    - Set appropriate run-time and access rights to configuration areas used by production ready CTI OS desktops such that only authorized users can launch an application based on its logon authorization level.

    - Silent installation/uninstallation supporting unattended software setup.

    - Use Vista's Side-by-Side (SxS) paradigm to install system library dependencies .
  – Implement a new MR/ES installation mechanism using the Native Patch Manager Framework.
  – NET CIL is registered at the target computer's .NET Framework 2.0 Global Assembly Cache (GAC).

Note Sample Code needs to built with Visual Studio 2005 in Administrative mode.

- Microsoft .NET Framework 2.0 & 3.0 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.

In order to access the .NET CIL and the RCWs directly from Visual Studio. NET 2005 you need to add the following configuration to your environment.

Adding CTI OS Toolkit 8.0(1) Components to the "Add Reference" Dialog Box

In Visual Studio.NET 2005, you have the ability to select class libraries and assemblies from the .NET tab of the "Add Reference Dialog". This facilitates the development process and always allows you to use the correct version of the components.

In order to enable the .NET CIL class libraries to appear on the Add References dialog, follow the steps described in:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbtskaddingremovingreferences.asp

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

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_CURRENT_USER\SOFTWARE\Microsoft\.NETFramework\<version>\AssemblyFoldersEx\MyAsse
mblies]@="<AssemblyLocation>"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\<version>\AssemblyFoldersEx\MyAss
emblies]@="<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>"

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

- Cisco CTI OS RCWs to the Add Reference dialog

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\v2.0.50727\AssemblyFoldersEx\Cisc
oCtiOsRCWs]@="C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32
CIL\.NETInterops"
```

Adding Cisco CTI OS ActiveX 8.0(1) Controls to the Toolbox

The Visual Studio .NET 2005 IDE allows visual editing of Windows Forms based applications toolbox of visual components available. Since Windows Forms applications are native, the visual components are also native. However, it still is possible to use ActiveX controls and they can also be included in the toolbox.

Adding CTI OS ActiveX 8.0(1) controls to the toolbox provides pre-packaged CTI functionality such as Agent Login, Make Call, Transfer Call, Barge In, etc. The ActiveX controls use COM CIL as the API to provide call center and telephony services. These components are use in rapid software development. You can "Drag & Drop" selected components into your project, and immediately gain the selected CTI functionality. These components are used in development environments such as: Microsoft Visual Basic 6.0, Microsoft Visual C++ 6.0, 7.0 & 8.0, Power Builder and others.

In order to use the Cisco CTI OS ActiveX 8.0(1) Controls in Visual Studio .NET 2005, it is necessary to configure the Cisco CTI OS RCWs as described in the previous section add to the Form Editor Toolbox:

1. From Visual Studio's View menu, make sure to select the "Add/Remove Toolbox Items…" command.

2. From the “Customize Toolbox” dialog box, select the “.NET Framework Components” tab.

Warning **Never select the COM Components tab from the “Customize Toolbox” dialog box and never select the CTI OS ActiveX controls. Doing this will cause Microsoft Visual Studio .NET 2005 to automatically generate a set of private RCWs that are not optimized nor approved by Cisco, and may lead your application for unexpected behavior that could result in application failure.**

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 the “AxAgentStateCtl”
4. To add more CTI OS ActiveX controls, repeat steps 1 to 3.

Integrating your Application with CTI OS via the 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.

Planning and Designing Your Integration

Good design depends upon understanding how CTI will fit 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 CTI data can be collected from the caller before the call arrives at an agent.
- **Study the agent’s workflow.** What are the points where CTI will be able to make the workflow easier and faster? Build a business case for the CTI integration.
- **Evaluate what will CTI do for your application.** A good approach is to make a list based on the priority (e.g. screen pop, then call control) and then design and implement features in that order.
- **Design how CTI should work within your application.** What are the interaction points? Get specifications as to which screen will 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 will be server-type integrations that will connect at startup, specify a *monitor-mode event filter*, and stay connected permanently. Agent-mode applications will connect up when a specific agent begins the work shift.
- **Clean up when you’re done.** When and how does the application stop? Some applications will stay up and running permanently, while others will have a defined runtime, such as the agent’s workday or shift. For server-type applications without a specified stopping point, create an object lifetime model and procedure for recovering no-longer-used resources. For applications with a specific stopping point, determine the kind of clean up needs to be done when the application closes (e.g. disconnect from server, release resources).

What Language and Interface to Use

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

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

- **ActiveX Controls.** The CTI OS ActiveX controls are the appropriate choice for creating a rapid "drag-and-drop" integration of CTI and third-party call control with an existing desktop application. The CTI OS ActiveX controls are appropriate choice for developing a CTI integration with any fully ActiveX-compliant container, or any other container that fully supports ActiveX features (e.g. Powerbuilder, Delphi, and many third-party CRM packages). The ActiveX controls will be the easiest to implement in graphical environments, and will help achieve the fastest integrations by providing a complete user interface. All CTI OS ActiveX components are distributed via dynamic link library files (.dll), which only have to be registered once to work on any Microsoft Windows platform. These components are not appropriate for non-Windows environments. The CTI OS ActiveX controls can be used 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, refer to "Using The CTI OS ActiveX Controls" on page 4-8.
- **COM**. The CTI OS Client Interface Library for COM (Microsoft's Component Object Model) is the appropriate choice for developing a CTI integration with any COM-compliant container, or any other container that supports COM features, such as Microsoft Internet Explorer or Visual Basic for Applications scripting languages. The COM CIL will be the easiest to implement in scripting environments, and will help achieve the fastest integrations requiring a custom or non-graphical user interface. All CTI OS components are distributed via dynamic link library files (.dll), which only have to be registered once to work on any Microsoft Windows platform. These components are not appropriate for non-Windows environments. The COM CIL can be used 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, refer to "Adding a Hook for Screenpops" on page 4-9.
- **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.

Testing CTI Applications

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

Developing a Test Plan

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

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

- **Unit Testing.** In a unit test, you ensure that the new code units can execute properly. Each component will operate correctly based on the input, and produce the correct output. An example of a unit test would be to 'stub-in' or hardcode the expected screen-pop data, and ensure that all of the screens come up properly based on this data.
- **Integration Testing.** In an integration test, you ensure that the new components work together properly. The physical connections and data passing between the layers and servers involved in the system are tested. An example of an integration test would be testing your client application with the CTI OS server, to ensure that data can be passed correctly through the components.
- **System Testing.** In a system test, you ensure that the correct application behavior is exhibited. An example of a system test would be to make a phone call to a VRU, collect the appropriate caller information, transfer the call to an agent, and ensure that the screen pop arrives correctly.
- **User Acceptance Testing.** In a user acceptance test, you ensure that your application has met all business requirements set by your analysis and design process. An example of a user acceptance test would be 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 can be used for application development and demo purposes. It has the capability to roughly simulate a Lucent PBX/ACD or a Cisco Unified Contact Center (Unified CC) environment. Documentation on how to configure and use the simulator can be found on the CTI OS CD in the directory Tools\Simulator.

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

Using the Samples

The CTI OS Software Development Toolkit (SDK) is distributed with a rich set of Developer Sample Applications (DSAs) for Cisco Unified Contact Center (Unified CC) customers and similar Production Class Applications for Unified ICM customers on the CD media.

The DSAs are provided as tools for Unified CC 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 will 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, Nortel, Aspect, etc.), some DSAs can be deployed 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 ICM Software Supported Switches (ACDs), at http://www.cisco.com for the current list of supported ACD types.

With Unified CC, 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.

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

Table 4-1 CTI OS Toolkit Sample Programs

Program Name	Location	Description
CTI Toolkit Outbound Desktop	CTIOS Toolkit\Win32 CIL\Samples\CTI Toolkit Outbound Desktop	A softphone application that demonstrates Outbound Option (formerly Blended Agent) functionality.
All Agents Sample .NET	CTIOS Toolkit\dotNet CIL\Samples\All Agents Sample.NET	A Microsoft C# program demonstrating a monitor mode application. This program lists all agents in a grid along with current state updates.
All Calls Sample.NET	CTIOS Toolkit\dotNet CIL\Samples\All Calls Sample.NET	Similar to AllAgents but lists calls instead of agents.
CTI Toolkit Combo Desktop.NET	CTIOS Toolkit\dotNet CIL\Samples\CTI Toolkit Combo Desktop.NET	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, Unified CC Supervisor and Outbound Option features.
CtiOs Data Grid.NET	CTIOS Toolkit\dotNet CIL\Samples\CtiOs Data Grid.NET	Microsoft C# program that implements a Tabular Grid used by the CTI Toolkit Combo Desktop.NET to show calls and statistics.
CTI Toolkit Agent Desktop	CTTIOS Toolkit\Win32 CIL\Samples\CTI Toolkit AgentDesktop	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.
CTI Toolkit Supervisor Desktop	CTTIOS Toolkit\Win32 CIL\Samples\CTI Toolkit IPCC SupervisorDesktop	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.
C++Phone	CTIOS Toolkit\Win32 CIL\Samples\CTI Toolkit C++Phone	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.
AllAgents	CTIOS Toolkit\Java CIL samples	A Java counterpart to the Visual Basic all agents program.

Using The 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.

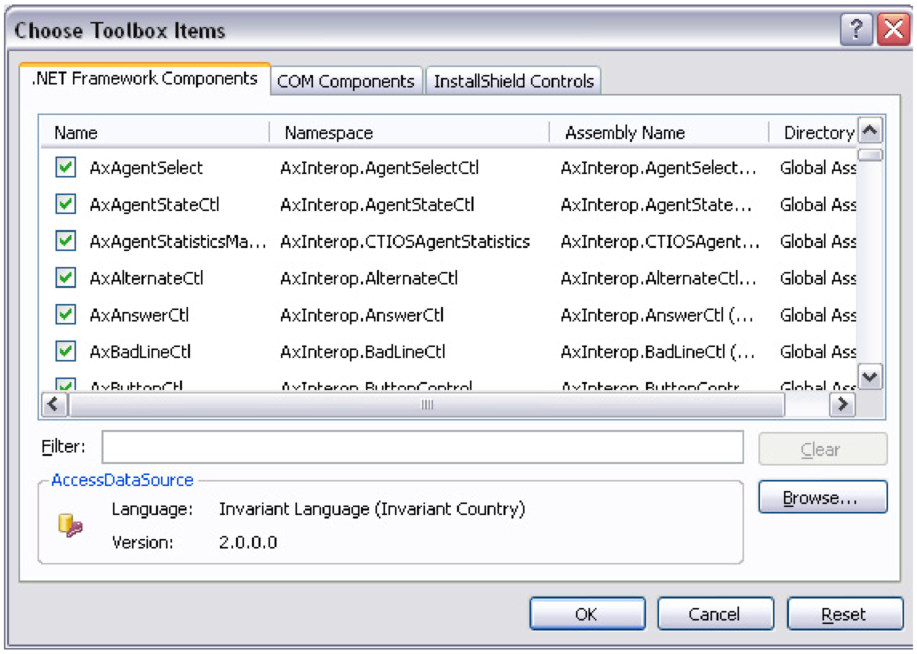
Building a Simple Softphone with ActiveX Controls

To use the CTI OS ActiveX controls, the ActiveX controls need to be copied on the target system and registered with Windows. This is accomplished by 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.

Once Visual Basic .NET is launched, you can use the ActiveX controls by selecting them via the Customized Toolbox dialog (**Tools->Add/Remove Toolbox Items** via the menu.)

Note Note: If the CTI OS ActiveX controls are not listed as shown in Figure 4-1, the files are either not copied on the target system or the controls were not properly registered.

Figure 4-1 Customize Toolbox in Visual Basic .Net Listing CTI OS ActiveX Controls Runtime Callable Wrappers



Choose Toolbox Items

.NET Framework Components | COM Components | InstallShield Controls

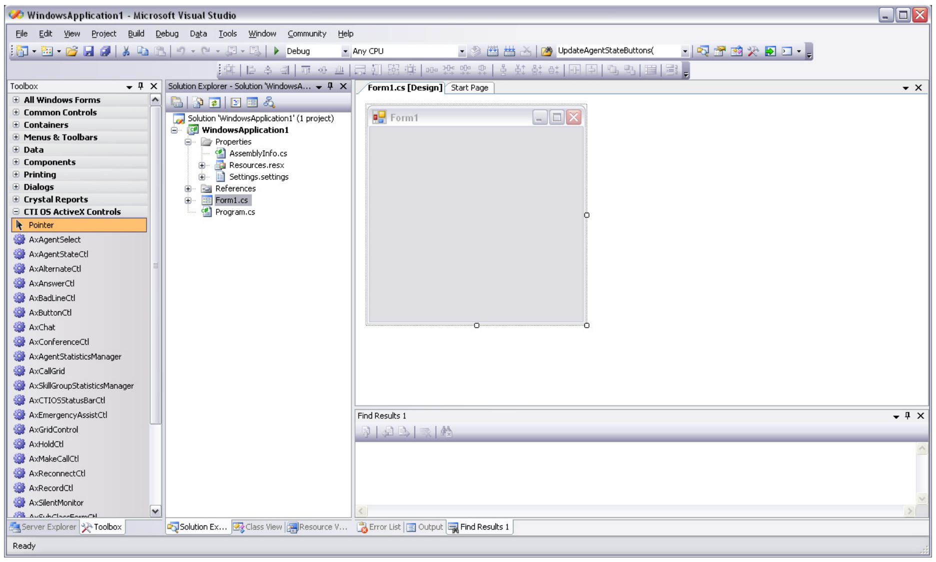
	Name	Namespace	Assembly Name	Directory
✓	AxAgentSelect	AxInterop.AgentSelectCtl	AxInterop.AgentSelect...	Global Ass
✓	AxAgentStateCtl	AxInterop.AgentStateCtl	AxInterop.AgentState...	Global Ass
✓	AxAgentStatisticsMa...	AxInterop.CTIOSAgentStatistics	AxInterop.CTIOSAgent...	Global Ass
✓	AxAlternateCtl	AxInterop.AlternateCtl	AxInterop.AlternateCtl...	Global Ass
✓	AxAnswerCtl	AxInterop.AnswerCtl	AxInterop.AnswerCtl (...	Global Ass
✓	AxBadLineCtl	AxInterop.BadLineCtl	AxInterop.BadLineCtl (...	Global Ass
✓	AxButtonCtl	AxInterop.ButtonControl	AxInterop.ButtonContr...	Global Ass

Filter:

AccessDataSource
Language: Invariant Language (Invariant Country)
Version: 2.0.0.0

Once the CTI OS ActiveX controls have been selected in the .NET Framework Components Tab they should be visible in the Visual Basic .NET ToolBox. The CTI OS ActiveX RCWs components can now be dragged and dropped onto the Windows Form. For a softphone application, it is useful to start with the CallAppearanceCtl (see Figure 4-2).

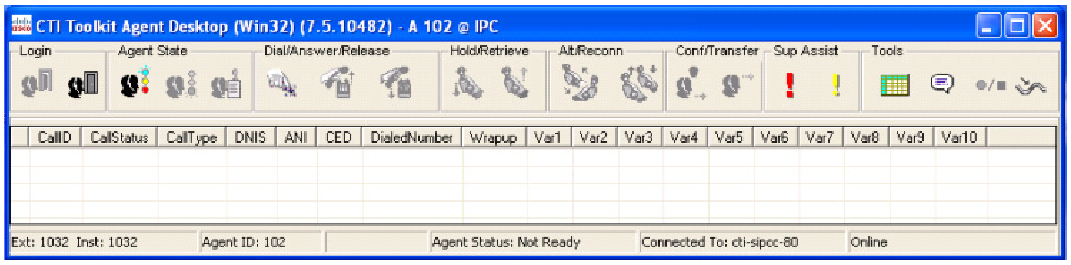
Figure 4-2 *Microsoft Visual Basic .NET Screen with the CTI OS ActiveX controls.*



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 Chapter 5, "CTI OS ActiveX Controls." Figure 4-3 shows the CTI OS Toolkit Agent Desktop application, which is also included as a sample on the CTI OS CD.

Figure 4-3 *CTI OS Toolkit Agent Desktop (see CD) Built with CTI OS ActiveX Controls*



Ext: 1032 Inst: 1032 | Agent ID: 102 | Agent Status: Not Ready | Connected To: cti-sipcc-80 | Online

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>**.

Adding a Hook for Screenpops

This agent desktop application did not require any Visual Basic .NET coding. A user may choose to add some custom code to add a hook for screenpops. For example, a user 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 Chapter 8, "Session Object"). 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 one chooses to write a program not using ActiveX controls, a Session object can be created and used directly (see CTI Toolkit AgentDesktop at the Win32 CIL samples).

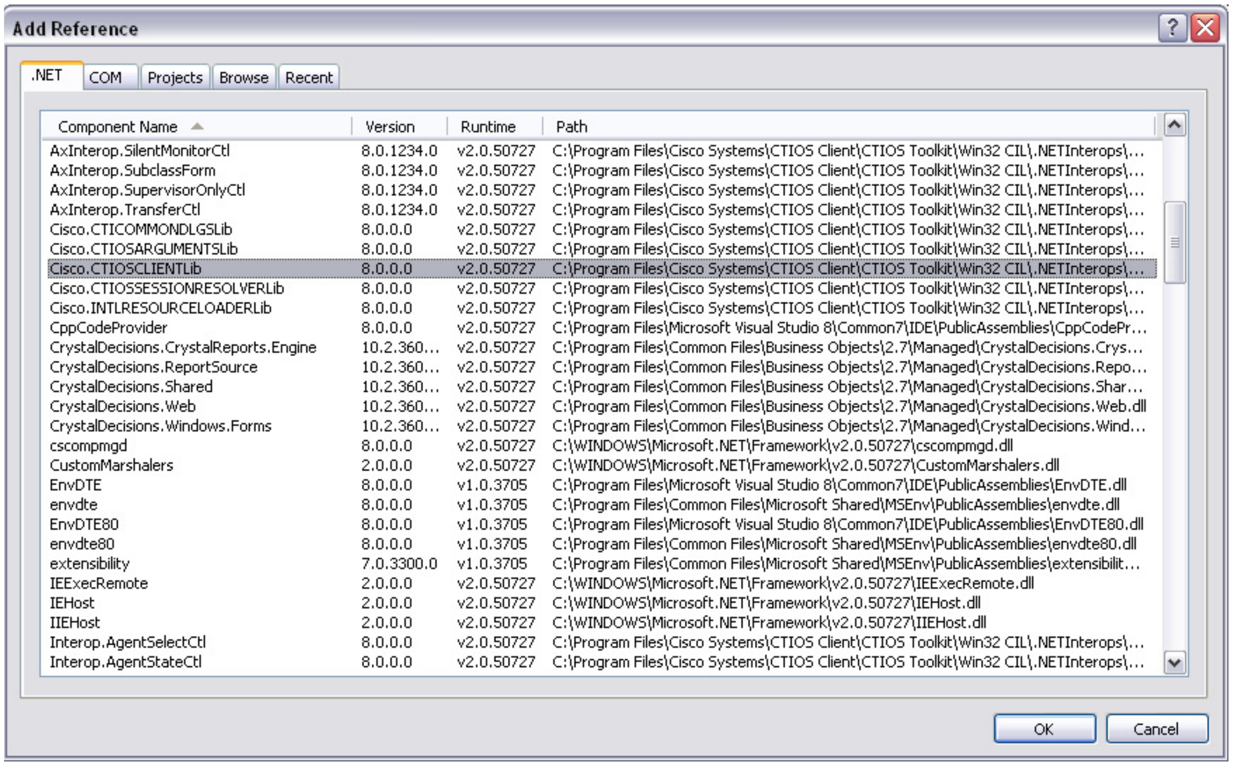
However, in the case of an application built with the ActiveX controls, all ActiveX controls must use the same session object. The ActiveX controls accomplish this by retrieving a pointer to the same session object via the SessionResolver. The program hosting the ActiveX can obtain the Same session object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

Sample VB .NET code to Retrieve CallVariable1

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

This code uses the COM CIL Interfaces and therefore, needs the following references: Cisco.CTIOSCLIENTLib, Cisco.CTIOSARGUMENTSLib, Cisco.CTIOSSESSIONRESOLVERLib. The references are shown in Figure 4-4 (in Visual Basic .NET, select **Project-> Add Reference**...).

Figure 4-4 CTI OS COM CIL RCWs References Needed for Visual Basic .NET COM Programming



Component Name	Version	Runtime	Path
AxInterop.SilentMonitorCtl	8.0.1234.0	v2.0.50727	C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\.NETInterops\...
AxInterop.SubclassForm	8.0.1234.0	v2.0.50727	C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\.NETInterops\...
AxInterop.SupervisorOnlyCtl	8.0.1234.0	v2.0.50727	C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\.NETInterops\...
AxInterop.TransferCtl	8.0.1234.0	v2.0.50727	C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\.NETInterops\...
Cisco.CTICOMMONDLGSLib	8.0.0.0	v2.0.50727	C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\.NETInterops\...
Cisco.CTIOSARGUMENTSLib	8.0.0.0	v2.0.50727	C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\.NETInterops\...
Cisco.CTIOSCLIENTLib	8.0.0.0	v2.0.50727	C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\.NETInterops\...
Cisco.CTIOSSESSIONRESOLVERLib	8.0.0.0	v2.0.50727	C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\.NETInterops\...
Cisco.INTLRESOURCELOADERLib	8.0.0.0	v2.0.50727	C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\.NETInterops\...
CppCodeProvider	8.0.0.0	v2.0.50727	C:\Program Files\Microsoft Visual Studio 8\Common7\IDE\PublicAssemblies\CppCodePr...
CrystalDecisions.CrystalReports.Engine	10.2.360...	v2.0.50727	C:\Program Files\Common Files\Business Objects\2.7\Managed\CrystalDecisions.Crys...
CrystalDecisions.ReportSource	10.2.360...	v2.0.50727	C:\Program Files\Common Files\Business Objects\2.7\Managed\CrystalDecisions.Repo...
CrystalDecisions.Shared	10.2.360...	v2.0.50727	C:\Program Files\Common Files\Business Objects\2.7\Managed\CrystalDecisions.Shar...
CrystalDecisions.Web	10.2.360...	v2.0.50727	C:\Program Files\Common Files\Business Objects\2.7\Managed\CrystalDecisions.Web.dll
CrystalDecisions.Windows.Forms	10.2.360...	v2.0.50727	C:\Program Files\Common Files\Business Objects\2.7\Managed\CrystalDecisions.Wind...
cscompmgd	8.0.0.0	v2.0.50727	C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\cscompmgd.dll
CustomMarshalers	2.0.0.0	v2.0.50727	C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CustomMarshalers.dll
EnvDTE	8.0.0.0	v1.0.3705	C:\Program Files\Microsoft Visual Studio 8\Common7\IDE\PublicAssemblies\EnvDTE.dll
envdte	8.0.0.0	v1.0.3705	C:\Program Files\Common Files\Microsoft Shared\MSEnv\PublicAssemblies\envdte.dll
EnvDTE80	8.0.0.0	v1.0.3705	C:\Program Files\Microsoft Visual Studio 8\Common7\IDE\PublicAssemblies\EnvDTE80.dll
envdte80	8.0.0.0	v1.0.3705	C:\Program Files\Common Files\Microsoft Shared\MSEnv\PublicAssemblies\envdte80.dll
extensibility	7.0.3300.0	v1.0.3705	C:\Program Files\Common Files\Microsoft Shared\MSEnv\PublicAssemblies\extensibilit...
IEExecRemote	2.0.0.0	v2.0.50727	C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\IEExecRemote.dll
IEHost	2.0.0.0	v2.0.50727	C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\IEHost.dll
IIEHost	2.0.0.0	v2.0.50727	C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\IIEHost.dll
Interop.AgentSelectCtl	8.0.0.0	v2.0.50727	C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\.NETInterops\...
Interop.AgentStateCtl	8.0.0.0	v2.0.50727	C:\Program Files\Cisco Systems\CTIOS Client\CTIOS Toolkit\Win32 CIL\.NETInterops\...

```
' 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.

Using the COM CIL in Visual C++ 8.0(1)

COM Client Interface Library (COM CIL.)

This API is used in development environments that support COM/DCOM and OLE Automation. Examples: Microsoft Visual Basic 6.0, Microsoft Visual C++ 6,0,70 & 8.0, Borland Delphi, Power Builder, etc. COM CIL is an adaptor interface that uses C++ CIL as kernel. The API is deployed as group of Dynamic Linked Libraries (DLLs)

Note **All C++ applications using COM CIL 8.0(1) must be built using Visual Studio .NET. Applications using COM CIL 8.0(1) built with Visual C++ 6.0 are not supported.**

Building a custom Win32 (Console or Windows) CTI application in Visual C++ 8.0(1) with COM requires knowledge of creating and using COM components in Microsoft Visual C++ 8.0(1). Client applications of this type tend to be more complex to build, and more powerful and faster in execution, than scripting clients (for example, Visual Basic). All the CIL components for COM are distributed as COM Dynamic Link Libraries (COM DLL).

In order to be accessible to COM containers including Microsoft Visual C++ 8.0(1), COM components must be registered with Windows. The components required for programming in Microsoft Visual C++ 8.0(1) are:

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

Adding COM Support to your Application

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

- Microsoft Foundation Classes (MFC). The following header 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++ 7.1(1) application wizard, these files are included for you 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++ 7.1(1) 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.

Using the CIL Dynamic Link Libraries

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

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

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

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

Creating an Instance of a COM Object

Note 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 will need to use the *CreateInstance()* method call.

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

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

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

Subscribing and Unsubscribing to COM Events in C++

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

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

```
class CEventSink : public CCmdTarget
{
//…
};
```

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

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

```
// Add event sink as event listener for the _IallEvents interface

HRESULT hRes =
AtlAdvise(m_pSession, m_EventSink.GetIDispatch(FALSE),
__uuidof(_IAllEvents), &m_dwEventSinkAdvise);
```

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

```
// Unsubscribe from the Session object for the _IAllEvents interface

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

Next Steps

- For detailed information on the CTI OS client start up and shut down sequence, see section "Disconnecting from CTI OS Server, page 4-29".
- For detailed information on 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) used 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 and is use to build Win 32 or console based applications.
- For a complete sample application that uses the CIL COM interface written in C++, see the Comphone sample application on the CTI OS CD.

Using the C++ CIL and Static Libraries

Note **All C++ applications using C++ CIL 8.0(1) must be built using Visual Studio .NET 2005. Applications using C++ CIL 8.0(1) built with Visual Studio .NET 2003 are not supported.**

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

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

Header Files and Libraries

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

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

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

- **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 OS Server.
- **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; e.g., for ArgumentsLibSpd, the Debug library is ArgumentsLibd.lib.

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

- the standard Microsoft sockets library, Wsock32.lib
- 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:

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

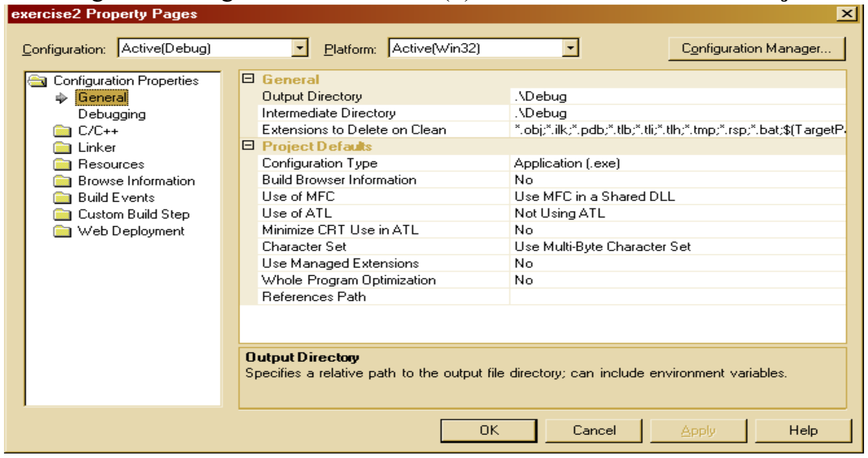
Use the following libraries in linker in addition to the CIL libraries:

- ws2_32.lib
- Winmm.lib
- odbc32.lib
- odbccp32.lib

Project Settings for Compiling and Linking

Setting up your Visual C++ 8.0(1) application requires you to configure some program settings.

The Program Setting in Visual C++ 8.0(1) are accessed under the **Project** > **Properties** menu.



Configuration: Active(Debug) Platform: Active(Win32)

General	
Output Directory	.\Debug
Intermediate Directory	.\Debug
Extensions to Delete on Clean	*.obj;*.ilk;*.pdb;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.bat;$(TargetP
Project Defaults	
Configuration Type	Application (.exe)
Build Browser Information	No
Use of MFC	Use MFC in a Shared DLL
Use of ATL	Not Using ATL
Minimize CRT Use in ATL	No
Character Set	Use Multi-Byte Character Set
Use Managed Extensions	No
Whole Program Optimization	No
References Path	

Output Directory
Specifies a relative path to the output file directory; can include environment variables.

Within the Project Settings dialog, select the C/C++ tab, select "General" and then select “Additional include Directories”. Provide either the absolute or relative path to find the header files (.h) required for your application. This path should point to the CTIOSToolkit\Win32 CIL\Include directory, where the CIL header files are installed.

Within the Property Pages dialog, select the C/C++ folder. Select Code Generation. For a Debug Mode program, the setting for “Runtime Library” should be “Multi-threaded Debug DLL (/MDd)”. For a Release Mode program, the setting should be “Multi-threaded DLL (/MD).”

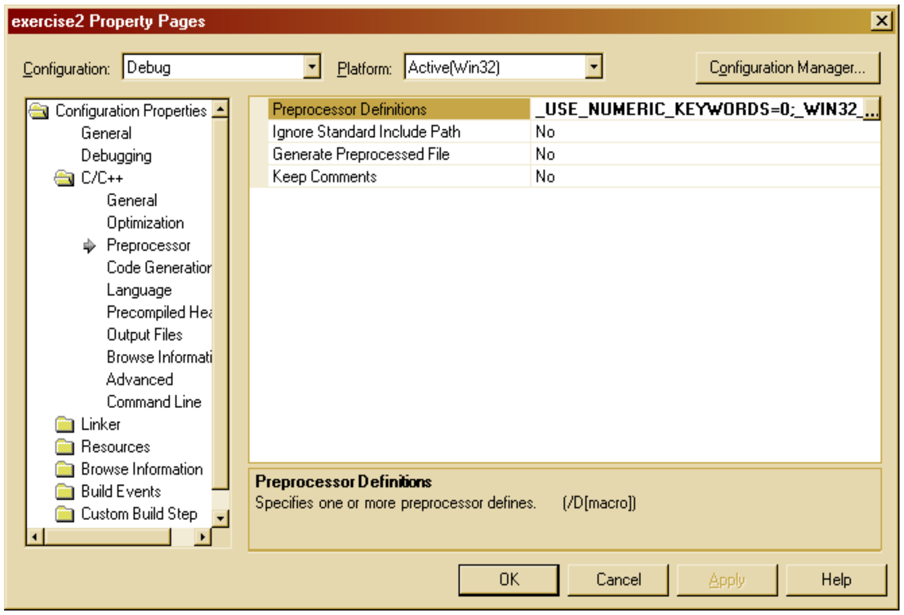
Next, under the “Proprocessor,” you will need to set the "Preprocessor Definitions". You need to provide the compiler with the following define constants _USE_NUMERIC_KEYWORDS=0;_WIN32_WINNT=0x0500; WIN32_LEAN_AND_MEAN in addition to the suggested by default.

The following setings must be set for the C++ compiler:

- Preprocessor Definitions
  - Add the following two macros:

    _CRT_SECURE_CPP_OVERLOAD_STANDARD_NAMES=1

    _CRT_SECURE_CPP_OVERLOAD_STANDARD_NAMES_COUNT=1

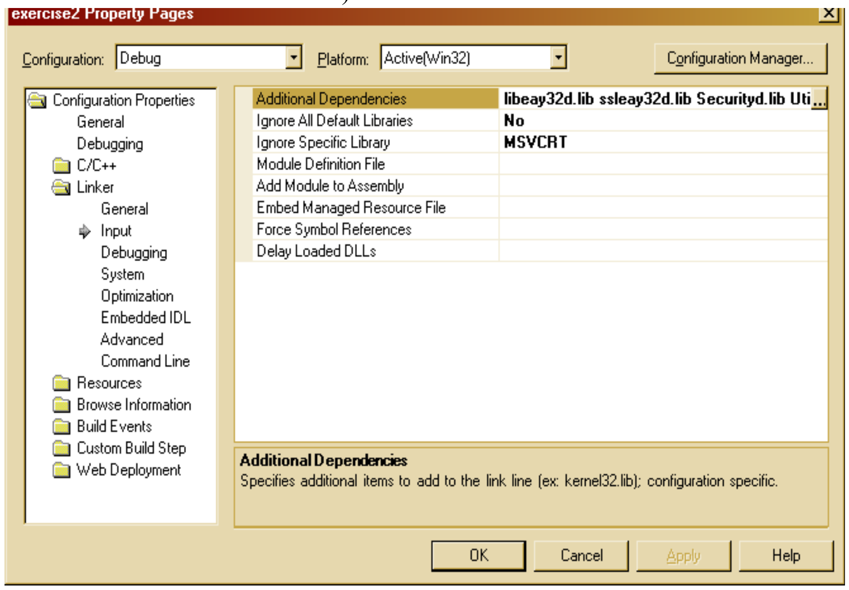


Preprocessor Definitions	_USE_NUMERIC_KEYWORDS=0;_WIN32_...
Ignore Standard Include Path	No
Generate Preprocessed File	No
Keep Comments	No

Preprocessor Definitions
Specifies one or more preprocessor defines. (/D[macro])

- Language
  - Set the parameter "Treat wchar_t as Built-in Type" to **No (/Zc:wchar_t-)**
- Precompile Headers
  - Set to **Not Using Precompile Headers**

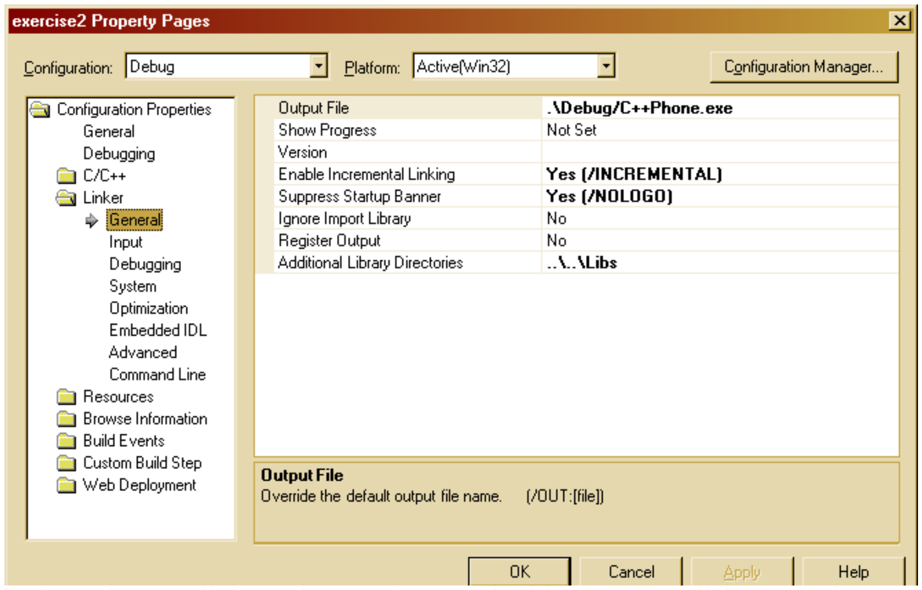
Next, you need to set the link settings for your project, under the Link folder. You must list all the static libraries (in the section "Header Files and Libraries" on page 4-14) for your program to link with the settings described in Project Settings for Compiling and Linking. The libraries required for CIL (in addition to the default libraries) are described in the section "Header Files and Libraries" on page 4-14.



Additional Dependencies	libeay32d.lib ssleay32d.lib Securityd.lib Uti...
Ignore All Default Libraries	No
Ignore Specific Library	MSVCRT
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

Finally, on the Link folder, select "General" to "Input." You will need to set the "Additional Library Directories:" to the location of the CTIOSToolkit\Win32 CIL\Libs directory:



Output File	.\Debug/C++Phone.exe
Show Progress	Not Set
Version	
Enable Incremental Linking	Yes (/INCREMENTAL)
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Additional Library Directories	..\..\Libs

Output File
Override the default output file name. (/OUT:[file])

The foregoing are all the Project Settings required for CTI OS. Click **OK**, and save your project settings.

Subscribing for Events in C++

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

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

```
// 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:

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

STLPort

Version 7.1(1)(0) of the Cisco CTI OS Toolkit no longer uses STLPort. Instead, it uses Microsoft's version of STL, thereby removing any special configuration of the build environment.

Next Steps

- For detailed information on the CTI OS client start up and shut down sequence, see the section "Disconnecting from CTI OS Server, page 4-29".
- For detailed information on the CTI OS Client Interface Library objects, see Chapters 6 through 11.
- For a complete sample application that uses the CIL interface with C++ static libraries, see the C++phone sample application on the CTI OS CD.

Using the Java CIL Libraries

The Java CIL provides a powerful cross-platform library for developing Java CTI applications. This Java API allows the creation of multiplatform client application that can be executed either in MS Windows or Linux. JavaTM CIL is built to supports the 1.4.2 Java Development Kit (JDK) and JREIt is built using a similar architecture to the C++ CIL and the interface is also similar to C++ with some differences. As a result, a developer porting a C++ CIL application to Java or working between a Java and C++ should find it fairly easy to switch between the two.

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

Sun JRE installers are also included on the media as a convenience for developers who need to obtain the correct version of the JRE. CTI OS Java CIL 8.0(1) has been updated to compile and run with the latest version of JDK/JRE.

The Java CIL ships with a GUI TestPhone application which 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 informationSee section "Using the Samples".

The CTI OS Java Test Phone has been updated and compiled with CTI OS Java CIL 8.0(1) using the JDK/JRE 1.6_01 for Linux and has been functionally tested on Red Hat Linux Enterprise 5.0.

Next Steps

- Refer to Event Interfaces and Events, page 6-1 and Keywords, page A-1 for differences between the C++ and Java event publishing.
- Refer to CtiOs Object, page 7-1 through 12 for differences in method calls and syntax for those classes between C++ and Java.
- Refer to Creating CTI OS Client Logs (COM and C++), page B-1 for differences between C++ and Java tracing.

Using the .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 also similar to C++ with some differences. As a result, a developer porting a C++ CIL application to .NET CIL between a .NET and Win32 should find it fairly 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 1.1 (CLR). The API can be use 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's Global Assembly Cache (GAC).

The .NET CIL consists of two class libraries: NetCil.dll and NetUtil.dll that need to be added as references on the build project. See the CTI OS Toolkit Combo Desktop sample.

For deploying the client application, it is recommended that the NetCil.dll and NetUtil.dll class libraries to be installed on the host's Global Assembly Cache (GAC) using the "gacutil" (provided by in Microsoft Visual Studio .NET 2003) or the "Microsoft .NET Framework 1.1" configuration manager. Together with .NET CIL are provided sample programs that teaches the use of the API under a .NET programming environment. For more information, see Chapter 4, "Using the Samples," .

Next Steps

- Refer to Event Interfaces and Events, page 6-1 and Creating CTI OS Client Logs (COM and C++), page B-1 for differences between the C++, and .NET and Java event publishing.
- Refer to CtiOs Object, page 7-1 through 12 for differences in method calls and syntax for those classes between C++ and Java.

Connecting to the CTI OS Server

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

1. Create a session instance, described below.
2. Set the event listener and subscribe to events, described below.
3. Set connection parameters, described below.
4. Call the Connect() method, described on page 4-21.
5. Set the connection mode, described on page 4-23.

This section also describes how to deal with connection failures, on page 4-22.

If your system is a duplexed Unified CCE PG with CSA installed, and you do not have one side of the CTI OS server running, CSA does not respond to login requests on the CTI OS server port. This triggers a timeout (20 second delay) before you attempt to connect to the active CTIOS server, in the CTI OS client machine TCP stack. On start-up or log-in, the CTI OS client randomly chooses a CTI OS server side to connect and it may connect to the server side that is **not** running.

To avoid this delay/timeout, you must:

- Start the inactive CTI OS server side
- Disable CSA (temporarily) and reconfigure the CTI OS desktop for a simplex operation
- Upgrade the version of the CTI OS server to CTI OS 8.0 (The desktop does not appear frozen though the delay persists)

How to Create the Session Instance

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

For example:

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

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

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

For example:

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

How to Set the Event Listener and Subscribe to Events

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

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

```
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.

How to Set Connection Parameters for the Session

To set connection parameters:

Step 1 Create an instance of the Arguments class.

Step 2 Set values for the CTI OS servers, ports, and the heartbeat value.

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

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

```
/* 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).

How to Connect the Session to the 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:**

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

For more information on the possible values and meanings of the int value returned by the Connect() method in the Java CIL, see page 4-22.

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

Dealing with Connection Failures

This section contains the following information:

- Connection Failure Events, page 4-22
- Connection Attempt Error Codes in Java and .NET CIL, page 4-22
- Configuring the Agent to Automatically Log In after Failover, page 4-23
- Stopping the Failover Procedure, page 4-23

Also see How to Deal with Failover In Monitor Mode, page 4-27.

Connection Failure Events

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

- OnConnectionRejected() event indicates that an unsupported version mismatch has been found.
- OnCTIOSFailure() indicates that the CTI OS Server requested in the Connect() method is down. If an OnConnectionFailure() event is generated, the application is in Failover and the CIL continues to attempt to connect until the connection succeeds or until the application calls Disconnect(). The Arguments parameter for the event includes the following keywords:
  - FailureCode
  - SystemEventID
  - 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 may 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 will either fire the OnConnection() event to indicate that the CIL successfully connected or will fire the OnConnectionFailure() event and go into failover mode. If the latter occurs, the CIL will continue to attempt to connect, alternating between hosts CTIOS_CTIOSA and CTIOS_CTIOSB, until the connection succeeds, at which point the CIL will fire the OnConnection() event.
- **E_CTIOS_INVALID_ARGUMENT** - A null Arguments parameter was passed to the Connect() method. The connection failed. No events are fired.
- **E_CTIOS_MISSING_ARGUMENT** - The Arguments parameter did not contain values for both CTIOS_CTIOSA and CTIOS_CTIOSB. At least one of these values must be provided. The connection failed. No events are fired.
- **E_CTIOS_IN_FAILOVER** - A previous connection attempt failed and the CIL is currently in failover and attempting to establish a connection. This continues until a connection is established, at which point the CIL will fire an OnConnection() event indicating that the previous Connect() method has succeeded. To attempt to connect again with different parameters, the application must first use the Disconnect() method.

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

Configuring the 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:

```
rArgs.SetValue(CtiOs_IKeywordIDs.CTIOS_AUTOLOGIN, “1”);
```

For more information on logging in an agent, see page 4-30.

Stopping the Failover Procedure

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

How to Set the Connection Mode

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

- Agent mode
- Monitor mode

Setting the Connection Mode in the OnConnection() Event Handler

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

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

When to Use Agent Mode

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

How to Select Agent Mode

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

Step 1 Set properties for the agent.

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

```
void OnConnection(Arguments rArgs) {

   /* 1. Create and agent and set the required properties. */
   Agent agent = new Agent();
   agent.SetValue(CtiOs_IKeywordIDs.CTIOS_AGENTID, "275");
   agent.SetValue(CtiOs_IKeywordIDs.CTIOS_PERIPHERALID, "5002");

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

}
```

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

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

When to Use Monitor Mode

Applications that need to receive all events that CTI OS Server publishes, or a specified subset of those events should use Monitor Mode. 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.

Caution 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 silent monitor agents

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

Warning **Note that the button enablement feature can only be used in agent mode sessions and are not intended for monitor mode applications.**

Monitor Mode Filters

Overview

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

Filter String Syntax

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

The general syntax for the filter String is as follows:

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

Note The filter String may also contain an asterisk (*), which is used as a wildcard to indicate any possible value. A prefix can be used in addition to * to narrow the results. For example, using 10* will match 1001, 1002, 10003.However, CTI OS ignores any characters that follow the asterisk. For example, using 10*1will match both 1001and 1002.

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

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

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

For example, a filter String could be as follows:

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

This example works as follows:

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

Filter Keys

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

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

For example, in **Java:**

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

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

Filtering for Events for Monitored Calls or Monitored Agents

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

- Create the filter
- Check events to verify that the CTIOS _MONITORED parameter is present and is TRUE
- Ignore events if the CTIOS_MONITORED parameter is missing or FALSE

How to Select Monitor Mode

To select Monitor mode for the connection:

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

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

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

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

Caution A Monitor Mode application that monitors any Call-related events must also monitor the OnCallEnd() event, as described on page 4-43.

The following example, which assumes the Session object has been created, demonstrates this task in 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 CC only, OnQueryAgentStateConf() for each team and each agent logged in
- OnSnapshotDeviceConf() for each device
- OnSnapshotCallConf()
- OnMonitorModeEstablished()

How to Deal with Failover In Monitor Mode

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

To deal with failover in Monitor mode:

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

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

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

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

a. Create a session instance.

b. Set the event listener.

c. Set connection parameters.

d. Call the Connect() method.

e. Set the connection mode in the OnConnection() event handler.

Settings Download

One of the many useful features of CTI OS is the ability to configure Agent Desktop settings for once on the server and have them be available to all agent desktops via the `RequestDesktopSettings`() method. Any changes can be made once on the server instead of changing each and every desktop. Settings download can be considered as part of the process of setting up a connection that the client application will use.

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 every time a client application connects ensures that all the desktops are based on the same settings.

Downloading settings from CTI OS Server can be done 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. These properties can be accessed via the `GetValue()` methods. Refer to Chapter 9 for a list of all the properties of the Session object.

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

```
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 they can be accessed 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:

```
Private Sub m_session_OnGlobalSettingsDownloadConf(ByVal pDispParam As Object)

Dim SoundArgs As CTIOSARGUMENTSLib.Arguments
' 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
```

Disconnecting from CTI OS Server

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

Note Disconnect is a higher priority method than all others. Before calling Disconnect, ensure that all prior requests have completed lest the call to Disconnect may abort these requests. For example, calling Disconnect immediately after calling Logout may 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 since 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 CCtiOsSsession::SetAgent(...) or CCtiOsSsession::SetMessageFilter(...), disconnect must be called with an arguments array containing the CTIOS_FORCEDDISCONNECT set to True.

```
m_session.Disconnect
// Perform disconnect
       if(m_ctiSession->GetValueInt(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;
          bAllocOk = Arguments::CreateInstance(&pDisconnectArgs);

          if ((false==bAllocOk) || (pDisconnectArgs == NULL))
          {
              CDialog::OnClose();
              argsWaitParams.Release();
              return;
          }

          pDisconnectArgs->AddItem(CTIOS_FORCEDDISCONNECT,true);
          m_ctiSession->Disconnect(*pDisconnectArgs);
          pDisconnectArgs->Release();
       }
       else
       {
         m_ctiSession->Disconnect();
       }
```

Logging In and Logging Out an Agent

How to 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 would typically request global configuration data, in order to correctly handle a duplicate log in attempt. For more information, see section "How to Get Registry Configuration Values to Your Desktop Application".

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

Step 1 Create an instance of the Arguments class.

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

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

Step 3 Log in the agent.

The following example, which assumes the Agent object has been created, demonstrates this task in 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 the these events must check both the ENABLE_LOGOUT and ENABLE_LOGOUT_WITH_REASON bitmasks. For more information, see What to do in the OnButtonEnablementChange() Event, page 4-40.

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

How to Handle Duplicate Log In Attempts

Overview

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

You can handle duplicate log in attempts in three ways:

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

You control how duplicate log in attempts are handled in two ways:

- By configuring how duplicate log in attempts are to be handled on a global basis by creating custom values in the CTI OS Server Registry. By using custom values in the CTI OS Server registry to control how duplicate log in attempts are handled, and downloading these settings to your desktop application as described on page 4-35, you can enable flexibility without having to modify your desktop application code.
- By implementing code in your desktop application to detect then to handle the duplicate log in attempt error according to the custom values in the CTI OS Server Registry. You can write code to handle duplicate log in attempts in each of the three ways listed above. Then when you must change how such attempts are handled, you would simply change the registry settings; you would then not need to change the desktop application code.

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

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

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

You should create two custom values:

- custom_WarnIfAgentLoggedIn
- custom_RejectIfAgentLoggedIn

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

The following table lists the settings to use to control how duplicate log in attempts are to be handled:

Table 4-2 :CTI OS Server Registry Settings (To control duplicate log in)

Goal	custom_WarnIfAgentLoggedIn	custom_RejectIfAgentLoggedIn
To warn the agent of the duplicate log in attempt, but to allow the agent to proceed	1	0
To allow the agent to proceed with the duplicate log in attempt with no warning	0	0
To not allow the agent to proceed with a duplicate log in attempt	0 or 1	1

To create keys to control duplicate log in attempts:

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

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

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

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

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

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

How to Prevent Another Agent Log In with Incorrect Credentials

In order 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:

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

How to Get Registry Configuration Values to Your Desktop Application

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

Step 1 Create an instance of the Arguments class.

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

- 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.

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

The following example demonstrates steps 1 through 3 in Java:

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

/* 2. Set the desktop type.*/
rArgs.SetValue("CTIOS_DESKTOPTYPE",
     CtiOs_Enums.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 "IPCC/SoftACD" login configuration information, the key for which is established by the CTI OS Server installation.

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

Step 7 For convenience, get the int values for those Integers to test with, as described in the section How to Handle Duplicate Log In Attempts, page 4-31.

The following example demonstrates steps 4 through 7 in Java:

```
void OnGlobalSettingsDownloadConf(Arguments rArgs) {

     /* 4. Get the Arguments instance for the Login configuration
     information from the event Arguments parameter.*/

     Arguments logInArgs = rArgs.getValueArray(CTIOS_LOGIN);

/* 5. Get the Arguments instance for the Connection Profile
from the Login Arguments instance. */

Arguments connectionProfilesArgs = logInArgs.GetValueArray(CTIOS_CONNECTIONPROFILES);

/* 6. Get the Arguments instance for the specific switch from the Connection
Profiles instance */

Arguments IPCCLogInArgs = connectionProfilesArgs.GetValueArray("IPCC/SoftACD")

/* 7. Get the Integer instances for the custom values you entered in the CTI OS Server
registry.*/

     Integer warningIntObj = IPCCLogInArgs.GetValueIntObj(“custom_WarnIfAgentLoggedIn”);

     Integer rejectIntObj =IPCCLogInArgs.GetValueIntObj(“custom_RejectIfAgentLoggedIn”);

/* 8. Get the int values for those object to test later.*/

     custom_WarnIfAgentLoggedIn = warnIntObj.intValue();
     custom_RejectIfAgentLoggedIn = rejectIntObj.intValue();
}
```

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

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

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

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

```
(state != eLogout) && (state != eUnknown)
```

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

The following example demonstrates this task in Java:

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

     /*Test the agent state*/
     if (agentState.intValue() != eLogout
                                                    && agentState.intValue() !=
eUnknown) {

             /*If the agent is logged in, handle duplicate log in attempt.*/
     }
}
```

How to Handle Duplicate Log In Attempts in the Desktop Application

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

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

How to Log Out an Agent

To log out an agent:

Step 1 Create an instance of the Arguments class.

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

Note Unified CC 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:

```
/* 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's the sample code for the same:

```
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 will additionally receive this event if the Logout request succeeds.

Note You may 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 will guarantee that all event listeners can make CTI OS server requests in response to the logout OnAgentStateChange event.

See the following example code:

```
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_IPCC_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 will 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 (e.g. when the user closes your application's window) you should do something similar to the code below:

```
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;
}

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

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

Working with Calls

Handling Multiple Calls

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

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

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

What is the Current Call?

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

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

The Current Call is set according to the following rules:

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

How to Get a Call Object

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

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

```
String sThisUID = UIDArray[Index];
Call ThisCall =  (Call) m_Session.GetObjectFromObjectID(sThisUID);
```

How to Set the Current Call for the Session

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

The following line shows this in Java:

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

Call Wrapup

The agent/supervisor desktop will need to behave differently at the end of a call depending on factors including:

- the direction of the call (inbound or outbound)
- configuration of Unified CC or the ACD (whether wrapup data is required, optional, or not allowed)
- configuration of CTI OS server

The CTI Toolkit Combo Desktop .NET sample shows how to use this information to display a wrapup dialog 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 CTI OS server will provide the following key/value pairs in the event arguments to aid in determining whether wrapup data may be 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

This value may be obtained using the GetValueInt method on the Agent object.

CTIOS_WRAPUPOKENABLED 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 determines whether the "Ok" button on the wrapup dialog is enabled when no wrapup information has been selected.) This value may be obtained using the GetValueBool method on the Agent object.

The wrapup strings that are configured on CTI OS 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. They can be obtained using the GetValueArray method on the Agent object. For information on how to configure wrapup strings on CTI OS server see Chapter 4 of the CTI OS System Manager's Guide.

Logout and NotReady Reason Codes

Depending upon the configuration of Unified CC or the configuration of CTI OS 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)

CTI OS server informs the CTI OS 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. This value can be obtained using the GetValueBool method on the Agent object.

CTIOS_LOGOUTREASONCODES - This Arguments array provides a list of the logout reason codes configured on CTI OS server. They can be obtained using the GetValueArray method 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. This value can be obtained using the GetValueBool method on the Agent object.

CTIOS_NOTREADYREASONCODES - This Arguments array provides a list of the not ready reason codes configured on CTI OS server. They can be obtained using the GetValueArray method on the Agent object.

When Does the Application Receive the OnButtonEnablementChange() Event?

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

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

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

What to do in the OnButtonEnablementChange() Event

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

The following examples shows this in Java:

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

Checking Not Ready Bitmasks in OnButtonEnablementChange() Event

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

Caution Failure to check both the ENABLE_NOTREADY and ENABLE_NOTREADY_WITH_REASON bitmasks may lead to problems properly displaying a NotReady control to the agent.

The following example shows this in Java:

```
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_btnReady.setEnabled(((bitMask & ENABLE_READY) > 0));
                     m_appFrame.m_btnRelease.setEnabled(((bitMask & ENABLE_RELEASE)> 0));
                     m_appFrame.m_btnRetrieve.setEnabled
                                 (((bitMask & ENABLE_RETRIEVE) > 0));
                     m_appFrame.m_btnSSTransfer.setEnabled
                                (((bitMask & ENABLE_SINGLE_STEP_TRANSFER)> 0));
                     m_appFrame.m_btnSSConference.setEnabled
                                 (((bitMask & ENABLE_SINGLE_STEP_CONFERENCE) > 0));
                     m_appFrame.m_btnTransfer.setEnabled
                                   (((bitMask & ENABLE_TRANSFER_COMPLETE)> 0));
                                    m_appFrame.m_btnCCTransfer.setEnabled
                         (((bitMask & ENABLE_TRANSFER_INIT) > 0));
                        }
             });
    } // OnButtonEnablementChange
```

OnButtonEnablementChange() Event in Supervisor Desktop Applications

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

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 not be able to make multiple duplicate requests before the first request has a chance to be sent down to the switch and the appropriate events be sent back to the application, since this will result in either multiple failures or unexpected results.

Preventing Multiple Duplicate Requests

Since it is important for a custom application to prevent a user from making a number of duplicate requests, it should not allow the user to click the same button many times. In order to do this, it is recommended that a custom application disable a clicked button until such time as it should be re-enabled, indicating that it would be all right for the user to click it again.

Some examples of when Sample softphones re-enable a button that has been 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.

Working with Events

Handling Events in 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. It is recommended that the amount of time spent in a particular CIL event handler be kept to a minimum so as to ensure timely delivery of subsequent CIL events. If a screenpop based on a call event (such as the OnCallDelivered event or the OnCallDataUpdate event) takes longer than a few seconds (for example, remote database lookup), it is good to 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 upon 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 COM CIL API method from a COM CIL event callback routine it must ensure that the method call is made on the same thread as the CIL event callback. This rule applies to the following methods:

- CComSession::SetCurrentCall
- CComSession::SetAgent

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

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

Monitoring the OnCallEnd() Event

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

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

Working with Agent Statistics

Overview

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

- "How to Set Up an Agent Application to Receive Agent Statistics" on page 4-44
- "How to Set Up a Monitor-Mode Application to Receive Agent Statistics" on page 4-45
- "Accessing Agent Statistics" on page 4-48
- "Changing Which Agent Statistics are Sent" on page 4-49
- "Agent Statistics Computed by the Sample CTI OS Desktop" on page 4-49

How to Set Up an Agent Application to Receive Agent Statistics

To set up an Agent application to receive agent statistics:

Step 1 Create an instance of the Session class, as described on page 4-20.

Step 2 Subscribe for events for the session, as described on page 4-21.

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

Step 3 Set connection parameters, as described on page 4-21.

Step 4 Connect the desktop application to the CTI OS Server, as described on page 4-21.

Step 5 In the OnConnection() event handler, set the Agent for the session, as described on page 4-24.

Step 6 Log in the agent, as described on page 4-30.

Step 7 Enable agents statistics using the EnableAgentStatistics() method.

Note 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.

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

Step 8 To disable agents statistics, use the DisableAgentStatistics() method.

The following example demonstrates this task in Java:

```
/* 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.*.
returnCode = rSession.Connect(rArgs);

public void OnConnection(Arguments rArgs) {

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

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

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

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

To set up a Monitor-mode application to receive agent statistics, follow the instructions below.

Note The agent to monitor must be logged in Agent mode before a Monitor-mode application can receive statistics for that agent.

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

Step 1 Create an instance of the Session class, as described on page 4-20.

Step 2 Subscribe for events for the session, as described on page 4-21.

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 page 4-21.

Step 4 Connect the desktop application to the CTI OS Server, as described on page 4-21.

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

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

Step 6 Set the message filter as described on page 4-26.

- **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 you an empty Arguments instance as a parameter.

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

Step 10 To disable agents statistics, use the DisableAgentStatistics() method.

The following example demonstrates this task in Java:

```
/* 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;
}
```

Accessing Agent Statistics

Overview

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

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

Registering to the eOnNewAgentStatisticsEvent() (JAVA)

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

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

```
String filter = S_MESSAGEID + "=" +

CtiOs_Enums.EventID.eNewAgentStatisticsEvent;
```

For more information on message filters, see page 4-25.

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

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

For more information on message filters, see page 4-25.

How to Get Agent Statistics through the Agent Instance

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

To get the agent statistics

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

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

The following example demonstrates this task in Java:

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

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

Changing Which Agent Statistics are Sent

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

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

Agent Statistics Computed by the Sample CTI OS Desktop

The sample CTI OS Desktop 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:

- *AvgTalkTimeToday* = (AgentOutCallsTalkTimeToday + HandledCallsTalkTimeToday) / (AgentOutCallsToday + HandledCallsToday)
- *CallsHandledToday* = AgentOutCallsToday + HandledCallsToday
- *TimeLoggedInToday* = LoggedOnTimeToday
- *TimeTalkingToday* = AgentOutCallsTalkTimeToday + HandledCallsTalkTimeToday
- *TimeHoldingToday* = AgentOutCallsHeldTimeToday + IncomingCallsHeldTimeToday
- *TimeReadyToday* = AvailTimeToday
- *TimeNotReadyToday* = NotReadyTimeToday
- *AvgHoldTimeToday* = (AgentOutCallsHeldTimeToday + IncomingCallsHeldTimeToday) / (AgentOutCallsToday + HandledCallsToday)
- *AvgHandleTimeToday* = (AgentOutCallsTimeToday + HandledCallsTimeToday) / (AgentOutCallsToday + HandledCallsToday)
- *AvgIdleTimeToday* = NotReadyTimeToday / (AagentOutCallsToday + HandledCallsToday)
- *PercentUtilitizationToday* = (AgentOutCallsTimeToday + HandledCallsTimeToday) / (LoggedOnTimeToday + NotReadyTimeToday)

Working with Skill Group Statistics

Overview

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

- "How to Set Up a Monitor-Mode Application to Receive Skill Group Statistics" on page 4-50

- “Accessing Skill Group Statistics” on page 4-51
- “Changing Which Skill Group Statistics are Sent” on page 4-52
- “Skill Group Statistics Computed by the Sample CTI OS Desktop” on page 4-52

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

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

Step 1 Create an instance of the Session class, as described on page 4-20.

Step 2 Subscribe for events for the session, as described on page 4-21.

Note You must register to receive session and skill group events; therefore, 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 page 4-21.

Step 4 Connect the desktop application to the CTI OS Server, as described on page 4-21.

Step 5 Set the message filter as described on page 4-26.

a. Create String for the filter using the keyword `S_FILTERTARGET` 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.

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

Step 6 Enable individual statistics as needed.

a. Create an instance of the Arguments class.

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

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

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

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

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

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

a. Create an instance of the Arguments class.

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

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

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

The following example demonstrates this task in Java:

```
/* 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);
```

Accessing Skill Group Statistics

Overview

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

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

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

Registering to the eOnNewSkillGroupStatisticsEvent() (JAVA)

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

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

```
String filter = S_MESSAGEID + "=" +

CtiOs_Enums.EventID.eNewSkillGroupStatisticsEvent;
```

For more information on message filters, see page 4-25.

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

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

For more information on message filters, see page 4-25.

Changing Which Skill Group Statistics are Sent

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

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

Skill Group Statistics Computed by the Sample CTI OS Desktop

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

- *AvgCallsQTimeNow* = CallsQTimeNow/CallsQNow
- *AvgAgentOutCallsTalkTimeToHalf* = AgentOutCallsTalkTimeToHalf/AgentOutCallsToHalf
- *AvgAgentOutCallsTimeToHalf* = AgentOutCallsTimeToHalf/AgentOutCallsToHalf
- *AvgAgentOutCallsHeldTimeToHalf* = AgentOutCallsHeldTimeToHalf/AgentOutCallsHeldToHalf
- *AvgHandledCallsTalkTimeToHalf* = HandledCallsTalkTimeToHalf/HandledCallsToHalf
- *AvgHandledCallsAfterCallTimeToHalf* = HandledCallsAfterCallTimeToHalf/HandledCallsToHalf
- *AvgHandledCallsTimeToHalf* = HandledCallsTimeToHalf/HandledCallsToHalf
- *AvgIncomingCallsHeldTimeToHalf* = IncomingCallsHeldTimeToHalf/IncomingCallsHeldToHalf

- *AvgInternalCallsRcvdTimeToHalf* = InternalCallsRcvdTimeToHalf/InternalCallsRcvdToHalf
- *AvgInternalCallsHeldTimeToHalf* = InternalCallsHeldTimeToHalf/InternalCallsHeldToHalf
- *AvgCallsQTimeHalf* = CallsQTimeHalf/CallsQHalf
- *AvgAgentOutCallsTalkTimeToday* = AgentOutCallsTalkTimeToday/AgentOutCallsToday
- *AvgAgentOutCallsTimeToday* = AgentOutCallsTimeToday/AgentOutCallsToday
- *AvgAgentOutCallsHeldTimeToday* = AgentOutCallsHeldTimeToday/AgentOutCallsHeldToday
- *AvgHandledCallsTalkTimeToday* = HandledCallsTalkTimeToday/HandledCallsToday
- *AvgHandledCallsAfterCallTimeToday* = HandledCallsAfterCallTimeToday/HandledCallsToday
- *AvgHandledCallsTimeToday* = HandledCallsTimeToday/HandledCallsToday
- *AvgIncomingCallsHeldTimeToday* = IncomingCallsHeldTimeToday/IncomingCallsHeldToday
- *AvgInternalCallsRcvdTimeToday* = InternalCallsRcvdTimeToday/InternalCallsRcvdToday
- *AvgInternalCallsHeldTimeToday* = InternalCallsHeldTimeToday/InternalCallsHeldToday
- *AvgCallsQTimeToday* = CallsQTimeToday/CallsQToday

Enabling Silent Monitor in Your Application

There are two (mutually exclusive) silent monitoring methods:

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

Refer to the "Silent Monitoring" section of the *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted* for additional information on each. For information concerning how to enable silent monitor in your application, refer to Enabling CTI OS Based Silent Monitoring in Your Application, page 4-53 or Enabling Unified CM Based Silent Monitoring in your Application, page 4-57, as applicable.

Enabling CTI OS Based Silent Monitoring in Your Application

Note 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.

Note Silent Monitor will not work until you set the session mode using on of the following function calls:
Session.SetAgent() for an Agent mode application
Session.SetMessageFilters() for a Monitor mode application

Creating a Silent Monitor Object

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

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

```
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)
```

Setting the Session Mode

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

Monitoring Mode

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

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

```
Dim m_Args As New Arguments
‘Assemble arguments to set the work mode
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:

```
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)
```

Initiating and Ending a Silent Monitor Session

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

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

Monitoring Client Code Sample

```
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(By Val 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(By Val 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(By Val pIArguments As
CTIOSCLIENTLib.IArguments, bDoDefaultProcessing)
     Dim strRequestInfo As String

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

Shutting Down Silent Monitor Manager

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

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

```
'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()
```

Initiating SilentMonitor from Monitor Mode Applications

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. The OnCallRTPStarted and OnCallRTPStopped events must be included in the filter used by the monitor mode application. An example follows.

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

Note Refer to Session Modes, page 2-4 for additional information.

Enabling 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 the agent's conversation 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.

Initiating a CCM Based Silent Monitor Request

Before initiating CCM based silent monitor, ensure that CCM based silent monitor has been configured. Refer to Determining the Silent Monitor Mode, page 4-61 for additional information.

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

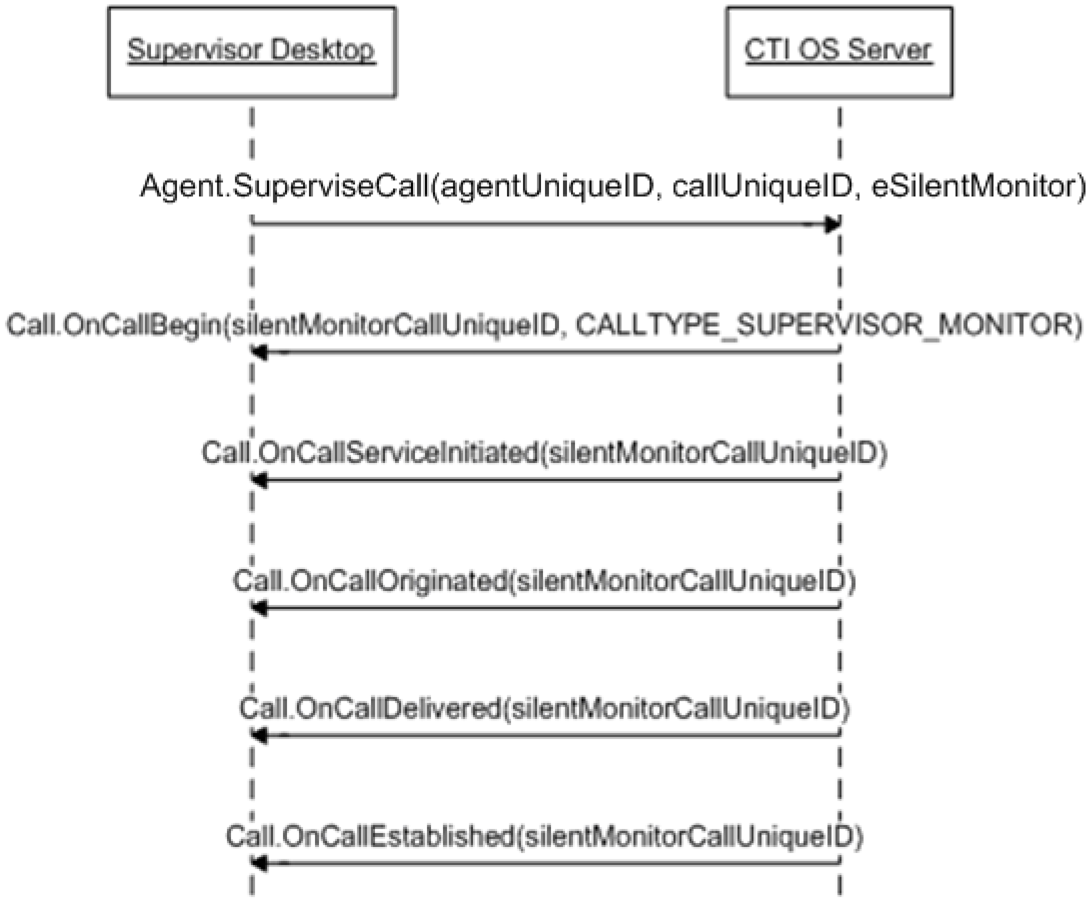
- The SupervisoryAction parameter must be set to eSupervisorMonitor.
- The AgentReference parameter must be set to the unique object ID of the agent to be silent monitored.

- The AgentCallReference parameter must be set 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. This event can be used to trigger application specific logic.

The Figure 4-5 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().

Figure 4-5 CIL-to-CTI OS Server Messaging when CCM Based Silent Monitor Initiated Using Agent.SuperviseCall()


Supervisor Desktop
CTI OS Server
Agent.SuperviseCall(agentUniqueID, callUniqueID, eSilentMonitor)
Call.OnCallBegin(silentMonitorCallUniqueID, CALLTYPE_SUPERVISOR_MONITOR)
Call.OnCallServiceInitiated(silentMonitorCallUniqueID)
Call.OnCallOriginated(silentMonitorCallUniqueID)
Call.OnCallDelivered(silentMonitorCallUniqueID)
Call.OnCallEstablished(silentMonitorCallUniqueID)

C# code sample for Initiating a Silent Monitor Session

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

string monAgentID;
monAgent.GetValueString(
    Enum_CtiOs.CTIOS_UNIQUEOBJECTID,
    out monAgentID);

string monCallID;
monCall.GetValueString(
    Enum_CtiOs.CTIOS_UNIQUEOBJECTID,
    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.eSupervisorMonitor) ;

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

Determining if the Current Agent is Being Silently Monitored

If an application needs to determine if the current agent is currently 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 the Current Agent is the Target of a Silent Monitor Call

Note The parameter args carries the payload of an OnSilentMonitorStartedEvent.

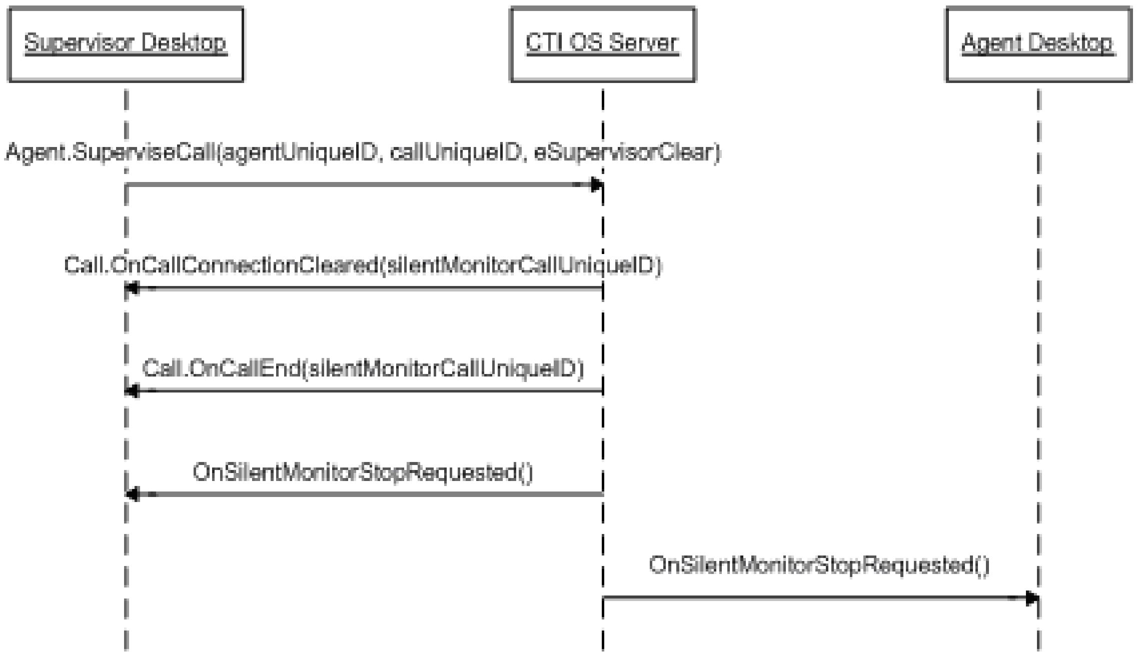
```
public bool IsCurrentAgentTargetAgent(Arguments args)
{
        bool isTarget = false ;

        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_SILENTMONITOR_TARGET_AGENTUID,
                          out targetAgentUID) ;

                                if ( targetAgentUID != null )
                                {
                                        isTarget = curAgentUID == targetAgentUID;
                                }
                        }
                }
        }

        return isTarget ;
}
```

Ending a CCM Based Silent Monitor Request

CCM Based silent monitoring is stopped using the SuperviseCall method associated with the supervisor's Agent object. To stop silent monitor, the SupervisoryAction parameter must be set to eSupervisorClear. The AgentReference parameter must be set to the unique object ID of the agent currently silent monitored. The AgentCallReference parameter must be set 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.

Figure 4-6 illustrates the message flow.

Figure 4-6 Message Flow when Ending a CCM Based Silent Monitor Request


Supervisor Desktop
CTI OS Server
Agent Desktop
Agent.SuperviseCall(agentUniqueID, callUniqueID, eSupervisorClear)
Call.OnCallConnectionCleared(silentMonitorCallUniqueID)
Call.OnCallEnd(silentMonitorCallUniqueID)
OnSilentMonitorStopRequested()
OnSilentMonitorStopRequested()

The silent monitor call can also be released using the Call.Clear() method.

Code Sample for Ending a Silent Monitor Session

```
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) ;
```

Determining the Silent Monitor Mode

In order 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.

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

Deployment of Custom CTI OS Applications

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

Deploying Applications Using the ActiveX Controls

ActiveX controls need all the components for COM deployment plus the components listed in Table 4-3.

Table 4-3 ActiveX Control DLLs

DLL	Description
Agentselectctl	AgentSelect ActiveX control
agentstatectl.dll	Agentstate ActiveX control
AlternateCtl.dll	Alternate ActiveX control
answerctl.dll	Answer/Release ActiveX control
arguments.dll	Arguments COM class
badlinectl.dll	Badline ActiveX control
buttoncontrol.dll	Basic Button ActiveX control
ccnsmt.dll	Cisco EVVBU Media Termination ActiveX control
chatctl.dll	Chat ActiveX control
conferencectl.dll	Conference ActiveX control
cticommondlgs.dll	Common Dialogs utility COM object
CTIOSAgentStatistics.dll	AgentStatistics ActiveX control
ctioscallappearance.dll	CallAppearance ActiveX control
ctiosclient.dll	COM cil interfaces
ctiossessionresolver.dll	COM sessionresolver

Table 4-3 ActiveX Control DLLs (continued)

DLL	Description
CTIOSSkillGroupStatistics.dll	SkillgroupStatistics ActiveX control
ctiosstatusbar.dll	StatusBar ActiveX control
EmergencyAssistCtl.dll	EmergencyAssist ActiveX control
gridcontrol.dll	GridControl ActiveX control
holdctl.dll	Hold/Retrieve ActiveX control
IntlResourceLoader.dll	Internationalization COM object
makecallctl.dll	MakeCall ActiveX control
ReconnectCtl.dll	Reconnect ActiveX control
recordctl.dll	Record ActiveX control
SilentMonitorCtl.dll	Standalone Silent Monitor ActiveX control
SubclassForm.dll	COM utility control
SupervisorOnlyCtl.dll	Supervisor ActiveX control
transferctl.dll	Transfer ActiveX control

ActiveX controls need to be copied and registered using the regsvr32 Windows utility. Some ActiveX controls are dependent on others. For example, all Button type controls (e.g. AgentStatectl.dll) depend on (buttoncontrol.dll) and all Grid type controls (e.g. CtiosCallappearance.dll) depend on Gridcontrol.dll. The following table below means that for a dll listed in the left column to work properly, all dll's listed in the right column ("dependencies") need to be available (copied and registered).

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

Table 4-4 Dependencies of CTI OS ActiveX Controls

DLL File	Dependencies
Agentselectctl	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll When used in a.NET application must include: AxInterop.AgentSelectCtl.dll Interop.AgentSelectCtl.dll
agentstatectl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommondlgs.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.AgentStateCtl.dll Interop.AgentStateCtl.dll

Table 4-4 Dependencies of CTI OS ActiveX Controls (continued)

DLL File	Dependencies
AlternateCtl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.AlternateCtl.dll Interop.AlternateCtl.dll
answerctl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.AnswerCtl.dll Interop.AnswerCtl.dll
arguments.dll	ATL80.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: Cisco.CTIOSARGUMENTSLib.dll
badlinectl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.BadLineCtl.dll Interop.BadLineCtl.dll
buttoncontrol.dll	ATL80.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.ButtonControl.dll Interop.ButtonControl.dll
ccnsmt.dll	Traceserver.dll, LIBG723.dll
chatctl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.ChatCtl.dll Interop.ChatCtl.dll
conferencectl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommondlgs.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.ConferenceCtl.dll Interop.ConferenceCtl.dll

Table 4-4 Dependencies of CTI OS ActiveX Controls (continued)

DLL File	Dependencies
cticommondlgs.dll	ATL80.dll, ctiosclient.dll, arguments.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: Cisco.CTICOMMONDLGSLib.dll
CTIOSAgentStatistics.dll	ATL80.dll, ctiosclient.dll, arguments.dll, Gridcontrol.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.CTIOSAgentStatistics.dll Interop.CTIOSAgentStatistics.dll
ctioscallappearance.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommondlgs.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.CTIOSCallAppearance.dll Interop.CTIOSCallAppearance.dll
ctiosclient.dll	ATL80.dll, arguments.dll, ctiosracetext.exe, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: Cisco.CTIOSCLIENTLib.dll If the client application will use silent monitoring in monitoring mode, ccnsmt.dll is also a dependency. If the client application will use silent monitoring in monitored mode, wpcap.dll is also a dependency.
ctiossessionresolver.dll	ATL80.dll, ctiosclient.dll, arguments.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: Cisco.CTIOSSESSIONRESOLVER Lib.dll

Table 4-4 Dependencies of CTI OS ActiveX Controls (continued)

DLL File	Dependencies
CTIOSSkillGroupStatistics.dll	ATL80.dll, ctiosclient.dll, arguments.dll, Gridcontrol.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.CTIOSSkillGroup Statistics.dll Interop.CTIOSSkillGroup Statistics.dll
ctiosstatusbar.dll	ATL80.dll, ctiosclient.dll, arguments.dll, cticommondlgs.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.CTIOSStatusBar.dll Interop.CTIOSStatusBar.dll
EmergencyAssistCtl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.EmergencyAssistCtl.dll Interop.EmergencyAssistCtl.dll
gridcontrol.dll	ATL80.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.GridControl.dll Interop.GridControl.dll
holdctl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.HoldCtl.dll Interop.HoldCtl.dll
IntlResourceLoader.dll	ATL80.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: Cisco.INTLRESOURCELOADER Lib.dll

Table 4-4 Dependencies of CTI OS ActiveX Controls (continued)

DLL File	Dependencies
makecallctl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommondlgs.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.MakeCallCtl.dll Interop.MakeCallCtl.dll
ReconnectCtl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.ReconnectCtl.dll Interop.ReconnectCtl.dll
recordctl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.RecordCtl.dll Interop.RecordCtl.dll
SilentMonitorCtl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, ccnsmt.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.SilentMonitorCtl.dll Interop.SilentMonitorCtl.dll
SubclassForm.dll	ATL80.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.SubclassForm.dll Interop.SubclassForm.dll

Table 4-4 Dependencies of CTI OS ActiveX Controls (continued)

DLL File	Dependencies
SupervisorOnlyCtl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.SupervisorOnlyCtl.dll Interop.SupervisorOnlyCtl.dll
transferctl.dll	ATL80.dll, ctiosclient.dll, arguments.dll, buttoncontrol.dll, cticommondlgs.dll, MSVCP80.dll, MSVCR80.dll **Note** When used in a.NET application must include: AxInterop.TransferCtl.dll Interop.TransferCtl.dll

Deploying Applications Using COM (But Not ActiveX Controls)

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

- CTIOSClient.dll
  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 will use silent monitoring in monitoring mode, ccnsmt.dll is needed. If the client application will use silent monitoring in monitored mode, wpcap.dll is also a dependency.

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

ATL80.dll is a Microsoft Dynamic Link Library implementing the Active Template Library used by CTI OS. It will usually be available on most Windows client systems in a windows system directory (e.g. \winnt\syste32 on Windows 2000). Since CTI OS depends on this DLL, it needs to be copied and registered if it is not already available at the target system.

Deploying Applications using C++ CIL

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

Deploying Applications using .NET CIL

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

Figure 4-7 .NET CIL libraries

Library	Description
NetCil.dll	.NET CIL Class library, contains the CTI OS object classes
NetUtil.dll	.NET Util Class library, contains helper and utility classes used in conjuction with .NET CIL

Both assmebly libraries are strongly signed such that they can be installed in the Global Assembly Cache (GAC) at the application host computer. Or of prefered by the developer they can be at the working directory of the custom client application.

Custom Application and CTI OS Security

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

SOFTWARE\Cisco Systems\CTI Desktop\CtiOs

If the **InstallDir** registry value doesn't 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
```

Building Supervisor Applications

This section describes how to build a supervisor desktop for Unified CC. 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. This class is used by the CTI OS Toolkit Combo Desktop to wrap CIL functionality.

The source code for the Combo Desktop can be 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. Once 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 will be 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 will receive 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 will receive 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.

Requesting and Monitoring the Supervisor's Teams

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:

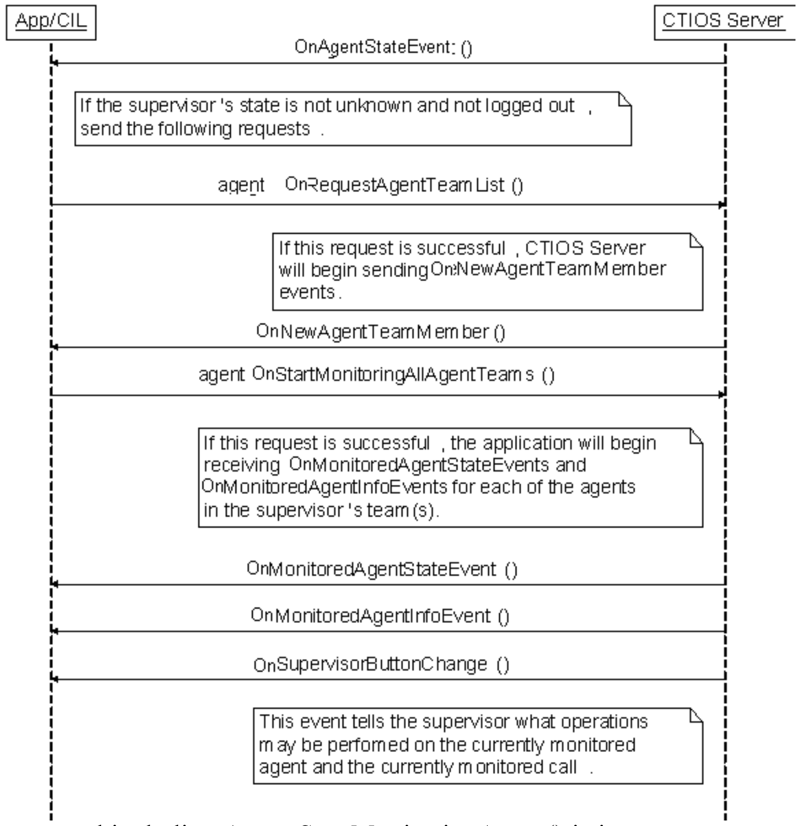
Agent.RequestAgentList(Arguments args)
Agent.StartMonitoringAgentTeams(Arguments args)

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. Since 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 4-8 Message Flow between the Application and the CTI OS Server


App/CIL
CTIOS Server
OnAgentStateEvent: ()
If the supervisor 's state is not unknown and not logged out , send the following requests .
agent OnRequestAgentTeamList ()
If this request is successful , CTIOS Server will begin sending OnNewAgentTeamMember events.
OnNewAgentTeamMember ()
agent OnStartMonitoringAllAgentTeams ()
If this request is successful , the application will begin receiving OnMonitoredAgentStateEvents and OnMonitoredAgentInfoEvents for each of the agents in the supervisor 's team(s).
OnMonitoredAgentStateEvent ()
OnMonitoredAgentInfoEvent ()
OnSupervisorButtonChange ()
This event tells the supervisor what operations may be performed on the currently monitored agent and the currently monitored call .

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 once the supervisor has logged 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 will begin receiving OnNewAgentTeamMember, OnMonitoredAgentStateChange, and MonitoredAgentInfoEvent events. The next sections describe how to handle each of these events.

OnNewAgentTeamMember

OnNewAgentTeamMember events should be processed as follows.

The OnNewAgentTeamMember event is received for two possible reasons. They are as follows.

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, the field "ConfigOperation" in the payload of the OnNewAgentTeamMember event must be examined. If this flag does not exist or exists and is set to TeamConfigFlag.CONFIG_OPERATION_ADDAGENT (1), the agent should be added to the grid. If the flag exists and is not set to TeamConfigFlag.CONFIG_OPERATION_ADDAGENT, the agent should be removed 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

Note Since the supervisor is considered part of the team, an OnNewAgentTeamMember event is sent for the supervisor logged into 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 will return 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 a team's 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 will generate OnMonitoredAgentStateChange events. The agent select grid normally updates when the OnMonitoredAgentStateChangeevent is received. In order to prevent this, supervisors who are not members of the team that they are supervising will need to be marked as such. This information can be used to avoid updates when an OnMonitoredAgentStateChange event is received for a supervisor that is not part of the team. In order to accomplish this, the application leverages the following.

1. OnNewAgentTeamMember events will not be 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 will mark every agent that is included in a OnNewAgentTeamMember event as a member of this supervisor's team. When OnMonitoredAgentStateChange events are received, the agent select grid will only update 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 a OnNewAgentTeamMember event to the CIL will not be 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 should use 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

This event can be used 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 has just been 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 has been 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 who's 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.

Populating an Agent Grid

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 4-5 Agent Grid Data Population

Column	Event	Field
Name	OnNewAgentTeamMember OnMonitoredAgentStateChange OnMonitoredAgentInfoEvent	Enum_CtiOs.CTIOS_AGENTFIRSTNAME Enum_CtiOs.CTIOS_AGENTLASTNAME
Login Name	OnMonitoredAgentStateChange OnMonitoredAgentInfoEvent	Enum_CtiOs.CTIOS_LOGINNAME
Agent ID	OnNewAgentTeamMember OnMonitoredAgentStateChange OnMonitoredAgentInfoEvent	Enum_CtiOs.CTIOS_AGENTID
Agent State	OnNewAgentTeamMember OnMonitoredAgentStateChange	Enum_CtiOs.CTIOS_STATE
Time in State	OnMonitoredAgentStateChange	See the section Time in State, page 4-72.
Skill Group	OnMonitoredAgentStateChange	Enum_CtiOs.CTIOS_NUMSKILLGROUPS
Skill Name	OnSkillInfoEvent	See the section OnSkillInfo Event, page 4-72.
Available for Call	OnNewAgentTeamMember	Enum_CtiOs.CTIOS_AGENTAVAILABILITYSTATUS

Note 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).

```
// 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 poisition {1}", sgStr, j)) ;

                m_skillGroupNumbers.Add(sgStr) ;
            }
            else
            {
                CtiOsDataGrid.Trace(
                    Logger.TRACE_MASK_WARNING,
                    methodName,
                    string.Format("No skillgroup number at poisition {0}", j)) ;

                m_skillGroupNumbers.Add(unknownStr) ;
            }
        }
    }
}
```

Monitoring Agents

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)

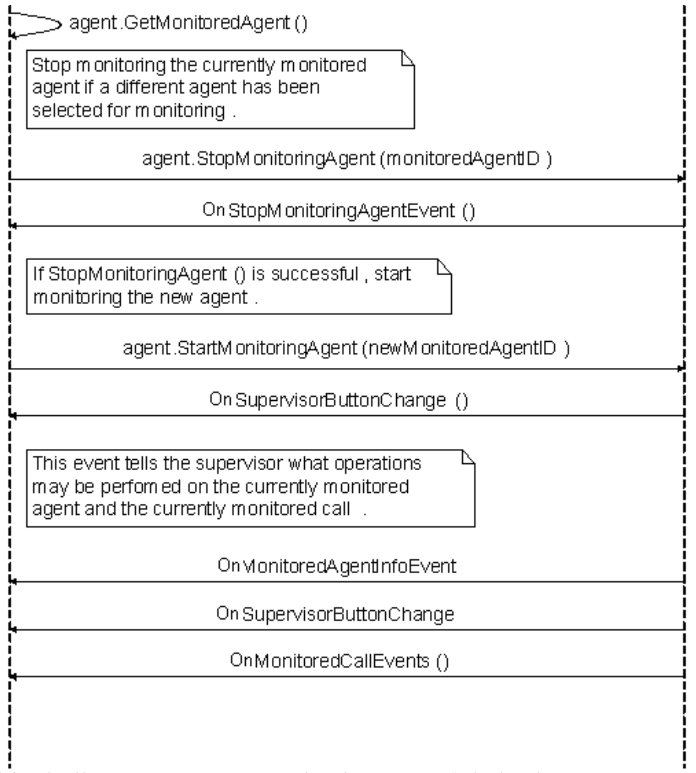
Events Processed:

OnSupervisorButtonChange
OnStopMonitoringAgent
OnMonitoredAgentStateChange
OnMonitoredCallBegin

OnMonitoredCallCleared
OnMonitoredCallConferenced
OnMonitoredCallConnectionCleared
OnMonitoredCallDataUpdate
OnMonitoredCallDelivered
OnMonitoredCallDequeued
OnMonitoredCallDiverted
OnMonitoredCallEstablished
OnMonitoredCallFailed
OnMonitoredCallHeld
OnMonitoredCallOriginated
OnMonitoredCallQueued
OnMonitoredCallReachedNetwork
OnMonitoredCallRetrieved
OnMonitoredCallServiceInitiated
OnMonitoredCallTransferred
OnMonitoredCallTranslationRoute
OnMonitoredCallEnd

Once a supervisor application has been 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 4-9 Sequence Diagram for StartMonitoringAgent() and successful Call Completion.


agent.GetMonitoredAgent ()
Stop monitoring the currently monitored agent if a different agent has been selected for monitoring .
agent.StopMonitoringAgent (monitoredAgentID)
On StopMonitoringAgentEvent ()
If StopMonitoringAgent () is successful , start monitoring the new agent .
agent.StartMonitoringAgent (newMonitoredAgentID)
On SupervisorButtonChange ()
This event tells the supervisor what operations may be performed on the currently monitored agent and the currently monitored call .
On MonitoredAgentInfoEvent
On SupervisorButtonChange
OnMonitoredCallEvents ()

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 4-6 Agent.StopMonitoring Agent Parameter

Key	Value
AgentReference	The UniqueObjectID of the currently monitored agent.

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 4-7 Agent.Start MonitoringAgent Parameter

Key	Value
AgentReference	The UniqueObjectID of the agent to begin monitoring.

OnSupervisorButtonChange

This event is delivered to define the operations that may be executed successfully by the supervisor. The operations included in this event are as follows.

- Logout 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

Note that the majority of events listed with StartMonitoringAgent() are monitored call events. These events inform the supervisor of monitored agent's calls beginning, ending, and changing. The combo desktop uses these events to populate its monitored calls grid.

Making Agents Ready and Logging Agents Out

When StartMonitoringAgent() is called for a given agent, the supervisor application will begin receiving SupervisorButtonChange events. This event may indicate that the monitored agent is in a state where the supervisor can make the agent ready or log the agent out. The following paragraphs describe 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 4-8 .Agent.SetAgentState Parameter

Key	Value
SupervisorID	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).
AgentState	The state to which to set the agent. In this case, the state is ready (integer with value 3).

To logout 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 4-9 .Agent.SetAgent State Parameter (logout)

Key	Value
SupervisorID	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).
AgentState	The state to which to set the agent. In this case, the state is ready (integer with value 3).
EventReasonCode	The value associated with this key is 999. The value 999 indicates to the rest of Unified CC that the agent was logged out by their supervisor.

An agent involved in a call will not be 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. This can be done by checking 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 will not be logged out until the agent has been 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.

Monitoring Calls

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

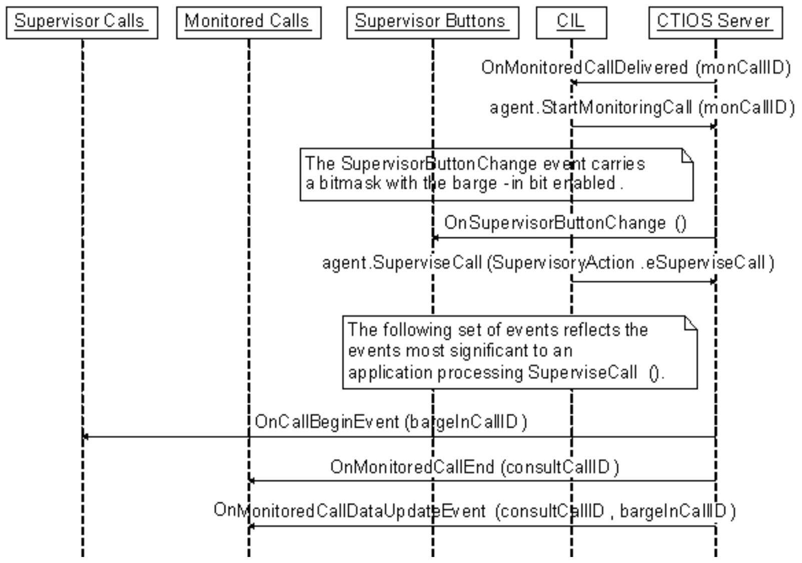
As stated in the "Monitoring Agents" section, calling Agent.StartMonitoringAgent() will trigger MonitoredCallEvents for the agent specified in Agent.StartMonitoringAgent(). The MonitoredCallEvents received by the supervisor desktop, inform the desktop of the state of the monitored agent's calls. The combo desktop uses these events to populate and update the monitored calls grid. Please see the SupervisorUIManager and CallAppearanceHelper classes for further details.

To monitor a given 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 4-10 Sequence Diagram for Barging into an Agent's Call


Supervisor Calls
Monitored Calls
Supervisor Buttons
CIL
CTIOS Server
OnMonitoredCallDelivered (monCallID)
agent.StartMonitoringCall (monCallID)
The SupervisorButtonChange event carries a bitmask with the barge -in bit enabled.
OnSupervisorButtonChange ()
agent.SuperviseCall (SupervisoryAction .eSuperviseCall)
The following set of events reflects the events most significant to an application processing SuperviseCall ().
OnCallBeginEvent (bargeInCallID)
OnMonitoredCallEnd (consultCallID)
OnMonitoredCallDataUpdateEvent (consultCallID , bargeInCallID)

Once Agent.StartMonitoringCall() is called for a specific call, the application will begin receiving SupervisorButtonChange events. When a call is being monitored, the SupervisorButtonChange event may 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 4-10 Agent.StartMonitoringCall Parameter

Key	Value
AgentReference	The UniqueObjectID of the currently monitored agent
CallReference	The UniqueObjectID of the currently monitored call
SupervisoryAction	The value 3. For the .NET CIL, this is SupervisoryAction.eSupervisorBargeIn

Upon successfully calling this method, the application will receive many events since 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 a OnButtonEnablementChange 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 will be 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 handing the resulting Call and MonitoredCall events. All CallEvents should be applied to whatever application specific object and/or UI element is managing calls directly connected to the supervisor's device (SoftphoneForm in the combo desktop). All MonitoredCallEvents should be applied 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, essentially 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 setup 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 will take 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 will be carried in all future events for the call. Application logic must be updated based on this information or new events for the call will not be tracked correctly.

Intercepting Calls

Once a supervisor has barged into an agent's call, the supervisor can intercept the call. This can be 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 4-11 Agent.SuperviseCall Parameter

Key	Value
AgentReference	The UniqueObjectID of the currently monitored agent
CallReference	The UniqueObjectID of the currently monitored call
SupervisoryAction	The value 4. For the .NET CIL, this is SupervisoryAction.eSupervisorIntercept

Calling this method will remove the agent from the call. This means that OnMonitoredEndCall events will be received for the agent. Also, OnSupervisorButtonChange events will be sent to reflect the current state of the monitored agent.

Updating Monitored Call Data

Setting monitored call data is very 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. The currently monitored call can be retrieved 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 the 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

Note Of all the samples provided in the CTI OS toolkit, the .NET sample applications provide the most complete set of coding examples. Therefore the .NET samples should be used as the reference implementation for custom CTI OS application development regardless of which language you plan to use in your custom development.

The Java and Win32 samples should be used asa 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 in general, 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

Configuring the CTI Toolkit Combo Desktop

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 should now be able to edit the file to fill in the appropriate values.

Following is an example configuration file.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
     <section name="JoeUser" type="System.Configuration.SingleTagSectionHandler" />
  <appSettings>
     <add key="LogFilePath" value=".\CtiOsClientLog" />
     <add key="CtiOsA" value="CtiOsServerA" />
     <add key="CtiOsB" value="CtiOsServerB" />
     <add key="PortA" value="42028" />
     <add key="PortB" value="42028" />
  </appSettings>
  </configSections>
  <JoeUser TraceMask="0xf" AgentID="1003" AgentInstrument="3009" PeripheralID="5000"
DialedNumbers="3011,3010" />
</configuration>
```

The configuration file is composed of the following elements. These elements are as follows.

configuration – This elements contains the configuration for the desktop.

appSettings – This section defines configuration settings that are shared by every Windows user that logs into 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" will be 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 is used to define Windows user specific sections of the configuration file. These sections are defined using the section element. You will notice in the sample configuration file that there is a section element under configSections corresponding to the element tagged with the Window's user name "JoeUser" under the configuration element. This section should not need to be manually modified. As different Windows users use the desktop, this section will be modified to include section elements for each of the users.

The rest of the configuration file is composed of 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, etc. Most of the attributes in this element should 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 IPCC 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's Toolkit includes a set of ActiveX controls to enable rapid application development. ActiveX controls are typically UI components (there are also ActiveX controls which are invisible at run time) that enable easy drag-and-drop creation of custom CTI applications in a variety of container applications. Container applications include: Microsoft Visual Basic 6.0, Microsoft Internet Explorer, Microsoft Visual C++ 7.1(1), 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.

See also the *CTI OS Agent Desktop User Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted* as well as the *CTI OS Supervisor Desktop User Guide for Cisco Unified Contact Center Enterprise & Hosted*.

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

Table 5-1 CTI OS ActiveX Controls

Control	Description
Agent Related Controls	
AgentStateCtl	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).
ChatCtl	Provides UI to send text messages to a supervisor or (if allowed) to other agents.
EmergencyAssistCtl	Provides UI to place Emergency and Supervisor Assist calls. These calls allow agents to conveniently contact a supervisor if they need help. The corresponding Unified ICM scripts must be configured in order for this control to work.
Call Related Controls	
AlternateCtl	Provides UI for alternate requests. If an agent has Call A active and Call B on hold, alternate will put call A on hold and make Call B active. Useful during consult calls.

Table 5-1 CTI OS ActiveX Controls (continued)

Control	Description
AnswerCtl	Provides UI to answer a call. Only a call with state "LCS_Alerting" (Ringing) can be answered.
BadLineCtl	Provides a UI to report a Bad Line, e.g. bad voice quality or equipment problems.
CallAppearanceCtl	A grid control displaying call information, including call status and context data.
ConferenceCtl	Provides UI to place a conference call in single step or consultative mode.
HoldCtl	Provides UI to put calls on hold and retrieve them.
MakeCallCtl	Provides UI to enter a telephone number and place a make call request.
ReconnectCtl	Provides a UI for reconnect requests. If an agent has Call A active and Call B on hold, reconnect will hang up call A and make Call B active. Useful during consult calls to return to the original call.
StatusBarCtl	Visually displays information about the logged on agent (id, instrument, extension, current state).
RecordCtl	Provides UI for Call Recording requests (start/stop recording), the requests will be forwarded to CTI Server, so they can be handled by a configured call recording service.
TransferCtl	Provides UI to transfer a call in single step or consultative mode.
Statistics Controls	
AgentStatisticsCtl	A grid control displaying real-time agent statistics. Columns are configurable at CTI OS server (see the *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted*).
SkillgroupStatisticsCtl	A grid control displaying real time skill group statistics. Columns are configurable at CTI OS server (see the *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted*).
Supervisor Controls	
AgentSelectCtl	Supervisor specific; displays all agent team members of a supervisor (configured by Unified ICM), including agent name, agentid, agentstate, timeinstate and skillgroups.

Table 5-1 CTI OS ActiveX Controls (continued)

Control	Description
SupervisorOnlyCtl	Provides UI for Supervisor functions including Barge-In, Intercept, logout monitored agent and make monitored agent ready.
SilentMonitorCtl.dll	Standalone control that provides the capability of creating a monitoring application that connects to CTI OS, but does not need to login as a supervisor.

Property Pages

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

Button Controls and Grid Controls

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

- Statusbarcontrol
- ChatCtl
- Utility controls (such as CtiCommonDlgs and SubClassFormCtl)

Note The Utility controls, such as CtiCommonDlgs 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.

Note 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 5-2 Button Enablement in a standard CTI OS desktop for Unified CCE

Scenarios	Buttons enabled and Description
Agent is not logged in to a desktop	Only the **Log-in** button is enabled.
Agent in the Not ready state	The agent is in the **Not Ready** state. The following buttons are enabled: Ready, Supervisor Assist, Emergency Assist, Statistics and Chat Control and Make Call Control. **Note** The Make Call Control button allows you to dial out only in the Not Ready (NR) state.
Agent in the Ready state	The agent is in the **Ready** state. The following buttons are enabled: Not Ready, Supervisor Assist, Emergency Assist, Statistics and Chat Control and Make call Control. **Note** The Make Call Control button allows you to dial out only in the Not Ready state.
Agent gets an incoming call which is alerted on the agent desktop	The agent is in the **Reserved** state. The following buttons are enabled: Statistics, Chat Control and Bad Line Ctrl.
Agent answers the call	The agents is in the **Talking** state. The following buttons are enabled: Hold, Release, Supervisor Assist, Emergency Assist, Conference, transfer, Statistics, Chat ctl 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.
Agent puts an answered call on hold	The agent is in the **On hold** state. The following buttons are enabled: Retrieve, Release, Supervisor Assist, Emergency Assist, Statistics, Chat ctl and Badline control.
Agent retrieves the call on hold	The agents is in the **Talking** state. The following buttons are enabled: Hold, Release, Supervisor Assist, Emergency Assist, Conference, transfer, Statistics, Chat ctl and Badline control.

Table 5-2 Button Enablement in a standard CTI OS desktop for Unified CCE (continued)

Scenarios	Buttons enabled and Description
Agent releases the call	The agent continues to be in the same state he was in, before talking the call. **Note** The agent will be 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 conference	The agent is in the **Talking** state. The following buttons are enabled: Statistics, Chat ctl, Bad Line Ctrl and Reconnect.
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. The following buttons are enabled: Hold, Release, Supervisor Assist, Emergency Assist, Conference, transfer, Statistics, Chat ctl and Badline control.
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 will be 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 ctl, BadLine Ctrl and Reconnect.

Table 5-2 Button Enablement in a standard CTI OS desktop for Unified CCE (continued)

Scenarios	Buttons enabled and Description
Consult transfer call is answered	*Consult transfer Initiator Desktop* The agent is in the **Talking** state. The following buttons are enabled: Statistics, Chat ctl, 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 ctl 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 will be 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 does a Single Step Transfer	The agent continues to be in the same state he was in, before talking the call. **Note** The agent will be 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 an Consult Transfer or a Consult Conference, till the Transfer/Conference is complete, there are two calls on the agent desktop of the initiator of Transfer and Conference.

Table 5-2 Button Enablement in a standard CTI OS desktop for Unified CCE (continued)

Scenarios	Buttons enabled and Description
The held call is selected for Consult Conference when the call rings on the conference agent desktop	The following buttons ar enabled: Statists, 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 IPCC 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 upon 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 once a call is answered and the agent is in the **Talking** state.
Agent desk setting configuration is set to **Required/Required with data**	No state change buttons are enabled when the agent is talking on a call. After the call, the agent state changes to **Wrap-up**. In the Wrap-up dialog box, if you select "Apply", the Ready and NR buttons are enabled and the agent's state changes based on the selection (button click) done by the agent. **Note** 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.

These are basic call scenarios and are to be used for reference, only. 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 will enable and disable themselves automatically based on the current state of the system. For example, if an agent is not logged in, the only button available to click is the Login Button (see AgentStateCtl), or if a call has not been answered and is selected in the CallAppearanceCtl, the Answer Button will be enabled (see AnswerCtl and CallAppearanceCtl). All buttons can be configured via their property pages to show custom text captions, custom icons and custom tooltip captions.

Grid Controls

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

- 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. It can also be set programmatically at run time.

CTI OS ActiveX Control Descriptions

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

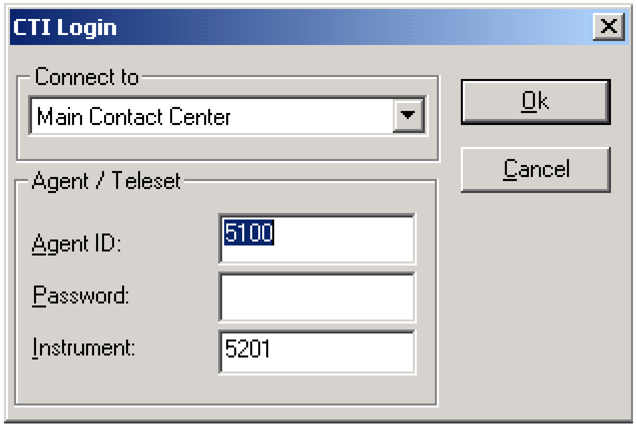
AgentStateCtl

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

- **Login Button**. Click the login button (Figure 5-2) to allow the agent to select a connection profile (see the *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted*), agent id and instrument or other switch specific fields.

Figure 5-1 Login Button

Figure 5-2 Login Dialog

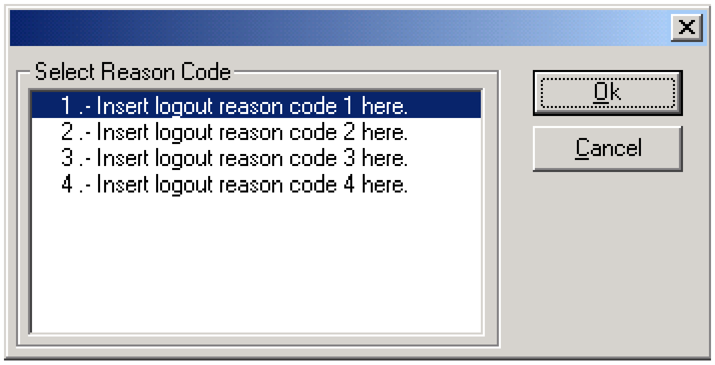

CTI Login
Connect to
Main Contact Center
Agent / Teleset
Agent ID: 5100
Password:
Instrument: 5201
Ok
Cancel

The fields displayed can be configured. The dialog shows a login dialog configured for Unified CCE. An agent logging in can select a connection profile for the **Connect To:** drop down box, enter, agent id, password and instrument and click on OK to send a Login request.

- **Logout Button**. Click the logout button to log out the currently logged in agent. For some switches, including Unified CCE, the agent needs to be in the not ready state in order for this button to be enabled. If Reason Codes are supported on the switch and configured on Unified ICM, a reason code dialog will pop up as shown in Figure 5-4.

Figure 5-3 Logout Button

Figure 5-4 Reason Code Dialog for Logout

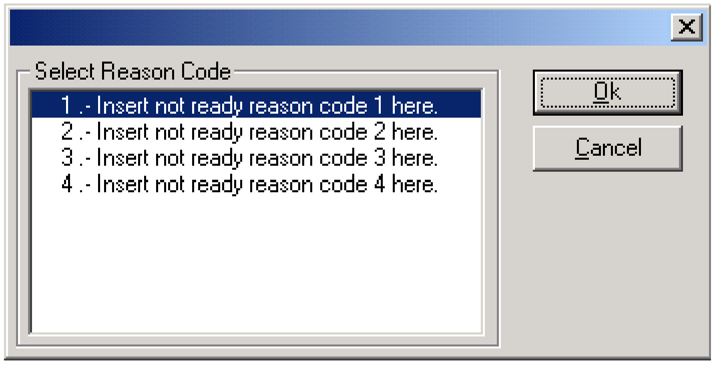

Select Reason Code
1 .- Insert logout reason code 1 here.
2 .- Insert logout reason code 2 here.
3 .- Insert logout reason code 3 here.
4 .- Insert logout reason code 4 here.
Ok
Cancel

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

- **Ready Button**. 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 will 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 will pop up as shown in Figure 5-5.

Figure 5-5 Reason Code Dialog for Not Ready



1 .- Insert not ready reason code 1 here.
2 .- Insert not ready reason code 2 here.
3 .- Insert not ready reason code 3 here.
4 .- Insert not ready reason code 4 here.

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

- **Work Ready Button**. 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 (see the *Administration Guide for Cisco Unified Contact Center Enterprise and Hosted Edition*).

- **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 (see the *Administration Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted*).

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:HRESULT ReasonCodeState([out, retval] enumReasonCodeState *pVal)
VB: ReasonCodeState()As AgentStateCtlLib.enumReasonCodeState
.NET:AgentStateCtlLib.enumReasonCodeState   ReasonCodeState()
```

Parameters

None

Return Value

Return value is enumReasonCodeState (This will return an Integer type)

ReasonCodeState

This version of the ReasonCodeState method sets the enumReasonCodeState value.

Syntax

```
COM:HRESULT ReasonCodeState([in] enumReasonCodeState newVal)
VB: ReasonCodeState = AgentStateCtlLib.enumReasonCodeState
.NET:ReasonCodeState = AgentStateCtlLib.enumReasonCodeState
```

Parameters

None

Return Value

None.

Following are the enumerated values for ReasonCodeState

```
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:HRESULT OnAgentStateChanged([in] LPDISPATCH vEventParam)
VB:
.NET:
```

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)
VB:
.NET:
```

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)
```

`VB:`
.NET:

Parameters

enabled

This is a control enabled changed value and returns a Boolean value.

Following are the Button enablement masks return by OnEnableControlReceived method.

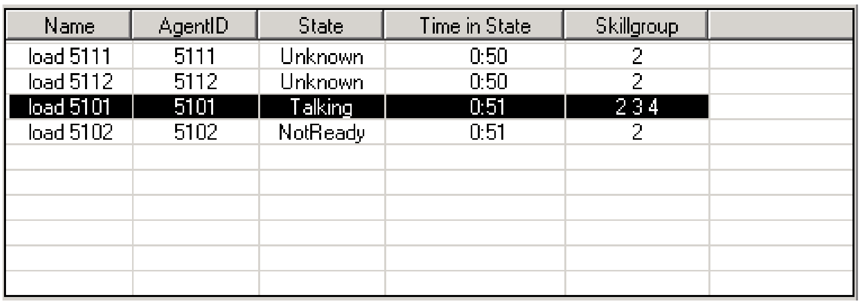
AgentSelectCtl

Table 5-3 OnEnableControlReceived Button Enablement Masks

m_eButtonType	nButtonMask
eLoginAgentBtn	ENABLE_LOGIN
eLoginSupervisorBtn	ENABLE_LOGIN
eLogoutBtn	ENABLE_LOGOUT or ENABLE_LOGOUT_WITH_ REASON
eReadyBtn	ENABLE_READY
eNotReadyBtn	ENABLE_NOTREADY or ENABLE_NOTREADY_WITH_REASON
eWorkReadyBtn	ENABLE_WORKREADY
eWorkNotReadyBtn	ENABLE_WORKNOTREADY

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

Figure 5-6 Agent Select Grid Populated with Sample Data



Name	AgentID	State	Time in State	Skillgroup	
load 5111	5111	Unknown	0:50	2	
load 5112	5112	Unknown	0:50	2	
load 5101	5101	Talking	0:51	2 3 4	
load 5102	5102	NotReady	0:51	2	

The agent select control handles the following events:

- **OnNewTeamMember**. Informs the supervisor of a new team member or a team member change. This will cause a row in the agentselect grid to be updated (add/remove agent).
- **OnMonitoredAgentStateChange**. Informs the supervisor of an agent state change. The new agentstate will be displayed in the State column and the TimeInState Column will be set to zero.
- **OnAgentInfo Event**.

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

Methods

Table 5-4 Available Methods for AgentSelectCtl

Method	Description
get_UserDefinedCell	If the column type is user defined, gets the text from the requested cell.
GetCellText	Gets the text from the requested cell in requested row.
GetColumnInfo	Gets the information about the requested column.
GetSelectedRow	Gets the selected row index.
SelectRow	Sets the requested row as selected.
set_ColumnHeader	Sets the column header of requested column with given text.
set_ColumnType	Sets the column type of requested column with given value.
set_ColumnWidth	Sets the column width of requested column with given value.
set_ UserDefinedCell	Sets the given text into the requested cell.
SetColumnInfo	Sets the given information for the requested column.

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

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.

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: GetColumnInfo(System.Int16 nCol, System.Int32 plcoltype, System.Int32 icolwidth, System.String bstrcoltitle)

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.

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

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 Value

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 Value

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 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)

.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 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)

.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 Value

None.

SetColumnInfo

Sets the given information for the requested column.

Syntax

COM: HRESULT SetColumnInfo([in] short nCol, [in] long lColType, [in] int iColWidth, [in] int iColTextAlign, [in] BSTR bstrColTitle)

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.

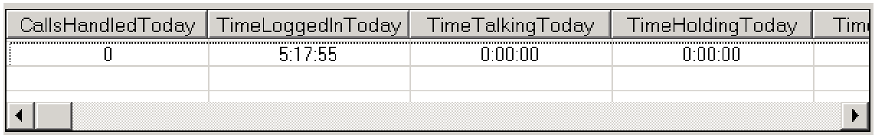
Return Value

None.

AgentStatisticsCtl

The AgentStatistics control is a grid based control displaying Unified ICM agent real time statistics. The columns displayed are configurable at CTI OS server (see the *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted*). Also, the update interval can be adjusted. It defaults to 10 seconds.

Figure 5-7 Agent Statistics Grid



CallsHandledToday	TimeLoggedInToday	TimeTalkingToday	TimeHoldingToday	Tim
0	5:17:55	0:00:00	0:00:00	

Methods

Table 5-5 Available Methods for AgentStatisticsCtl

Method	Description
get_UserDefinedCell	If the column type is user defined, gets the text from the requested cell.
GetCellText	Gets the text from the requested cell in requested row.
GetColumnInfo	Gets the information about the requested column.
set_ColumnHeader	Sets the column header of requested column with given text.
set_ColumnType	Sets the column type of requested column with given value.
set_ColumnWidth	Sets the column width of requested column with given value.
set_ UserDefinedCell	Sets the given text into the requested cell.
SetColumnInfo	Sets the given information for the requested column.

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

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.

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: GetColumnInfo(System.Int16 nCol, System.Int32 plcoltype, System.Int32 icolwidth, System.String bstrcoltitle)

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.

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 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)

.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 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)

.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 Value

None.

SetColumnInfo

Sets the given information for the requested column.

Syntax

COM: HRESULT SetColumnInfo([in] short nCol, [in] long lColType, [in] int iColWidth, [in] int iColTextAlign, [in] BSTR bstrColTitle)

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

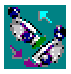
This is a column title text and an input parameter as type String.

Return Value

None.

AlternateCtl

Figure 5-8 AlternateCtl



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

AnswerCtl

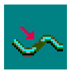
The Answer Control is a button that provides UI for sending answer and release call requests. The behavior (answer or release) can be set via the ButtonType set from the property page as explained under AgentState controls.

Figure 5-9 Answer Icon:

Figure 5-10 Release Icon:

BadLineCtl

Figure 5-11 BalLineCtl

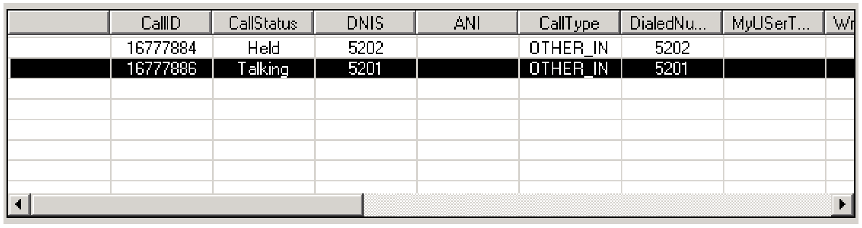


The Bad Line Control is a button that provides UI for reporting a Bad Line. This request will generate a database entry in Unified ICM and is an indicator for voice/equipment problems.

CallAppearanceCtl

The CallAppearance Control is a grid based control displaying call information, including call status and call context data (i.e., CallVariable1 through CallVariable10 and ECC variables).

Figure 5-12 CallAppearance Control Displaying Two Calls



	CallID	CallStatus	DNIS	ANI	CallType	DialedNu...	MyUSerT...	W
	16777884	Held	5202		OTHER_IN	5202		
	16777886	Talking	5201		OTHER_IN	5201		

Each incoming or outgoing call is displayed in one row in the grid. When a call first arrives, it will usually show a status of "Ringing", until it is answered. A call can be answered by a double click in the grid, similar to a click on the Answer Button. Some columns in the CallAppearance, grid can be edited if so configured (for example, the Columns displaying Callvariables) by selecting on the cell to be edited.

The grid can display multiple calls (see Figure 5-12). If the grid is displaying multiple calls, user can click and select a call anywhere on the row where the call is displayed. This will highlight the whole row displaying this call (e.g. in Figure 5-12 the call with id 16777886 is currently selected). Any button controls (e.g., Answer, Release, Hold,) will enable or disable themselves based on the state the newly selected call is in.

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

The CallAppearance grid can be in one of two modes. In "normal" mode it will show any calls for the agent/supervisor logged in; in "monitored" mode (only for supervisor), the CallAppearance grid will display all calls for a currently monitored agent (see Agent Select grid). A supervisor can 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

See Chapter 10, "Call Object."

GetValueInt

See Chapter 7, "CtiOs Object."

GetValueString

See Chapter 7, "CtiOs Object."

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

```
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 5-6 Available Methods for CallAppearanceCtl

Method	Description
GetCellText	Gets the text from the requested cell in requested row.
GetSelectedRow	Gets the selected row index.
SelectRow	Sets the requested row as selected.

Method	Description
set_ColumnECCName	Sets the column ECC name of requested column with given text.
set_ColumnECCOffset	Sets the column Offset value of requested column with given value.
set_ColumnHeader	Sets the column header of requested column with given text.
set_ColumnWidth	Sets the column width of requested column with given value.
SetCellText	Sets the given text to the requested cell in requested row.

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

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 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)

.NET: set_ColumnWidth(System.Int16 nCol, System.Int16 nNewValue)

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.

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.

nNewVal

This is a user passing header text and an input parameter as type String.

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)

.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 Value

None.

SetCellText

Sets the given text to the requested cell in requested row.

Syntax

COM: HRESULT SetCellText([in] int nRow, [in] int nCol, [in] BSTR bstrContent, [out,retval] VARIANT_BOOL * bStatus)

VB: SetCellText(nRow As Integer, nCol As Integer, bstrContent As String) As Boolean

.NET: System. Boolean SetCellText(System.Int16 nRow, System.Int16 nCol, System.String bstrContent)

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.

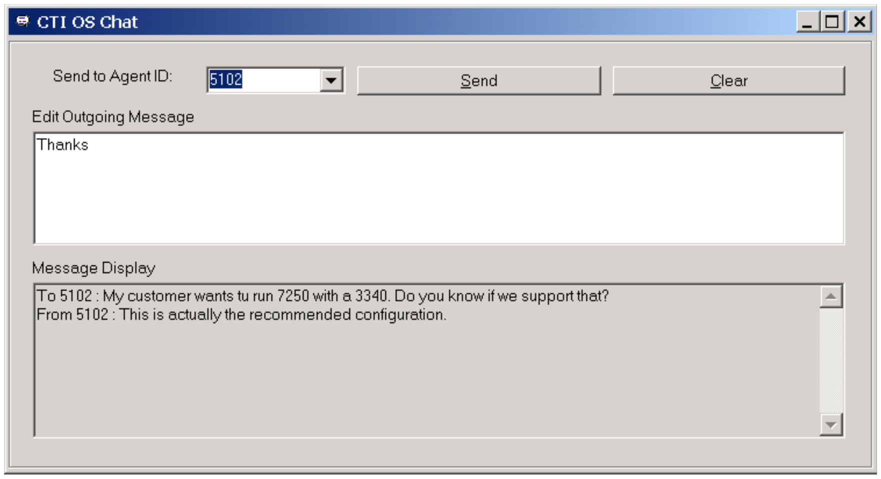
Return Value

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 (see the *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted*).

Figure 5-13 Chat Control



Thanks

To 5102 : My customer wants tu run 7250 with a 3340. Do you know if we support that?
From 5102 : This is actually the recommended configuration.

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

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

Methods

Table 5-7 Available Methods for ChatCtl

Method	Description
GetAddressee	Gets the current Addressee from the Send to Agent ID Combo box.
GetAllChatMessages	Gets the all chat messages from the Message Display Text Area.
GetChatMessageText	Gets the chat message from the Edit Outgoing Message Text Area.
OnMsgReceived	When message received from an Agent, appends the received message to the Message Display Text Area.
SendChatMessage	Sends the chat message to current Addressee in the Send to Agent ID Combo box.
SetAddressee	Sets the current Addressee to the Send to Agent ID Combo box.
SetChatMessageText	Sets the chat message to the Edit Outgoing Message Text Area.

GetAddressee

Gets the current Addressee from the Send to Agent ID Combo box.

Syntax

COM: HRESULT GetAddressee ([out,retval] BSTR* addressee)

VB: GetAddressee()As String

.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

None.

Return Value

Return type is String.

GetChatMessageText

Gets the chat message from the Edit Outgoing Message Text Area.

Syntax

COM: HRESULT GetChatMessageText ([out, retval] BSTR* MessageText)

VB: GetChatMessageText() As String

.NET: System.String GetChatMessageText()

Parameters

None.

Return Value

Return type is String.

OnMsgReceived

When message 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 form 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

COM: HRESULT SendChatMessage([in] BSTR addressee, [in] BSTR msg)

VB: SendChatMessage(addressee As String, msg As String)

.NET: SendChatMessage (System.String addressee, System.String msg)

Parameters

addressee

This is as Agent ID, who will receives the message and is an input parameter as type String.

msg

This is a message test send to an Agent and is an input parameter as type String.

Return Value

None.

SetAddressee

Sets the current Addressee to the Send to Agent ID Combo box.

Syntax

COM: HRESULT SetAddressee ([in] BSTR addressee)

VB: SetAddressee(addressee As String)

.NET: SetAddressee(System.String addressee)

Parameters

addressee

This is as Agent ID, who will 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 out going message text and is an input parameter as type String.

Return Value

None.

ConferenceCtl

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

Figure 5-14 Icon for ConferenceInit:

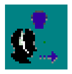
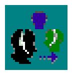
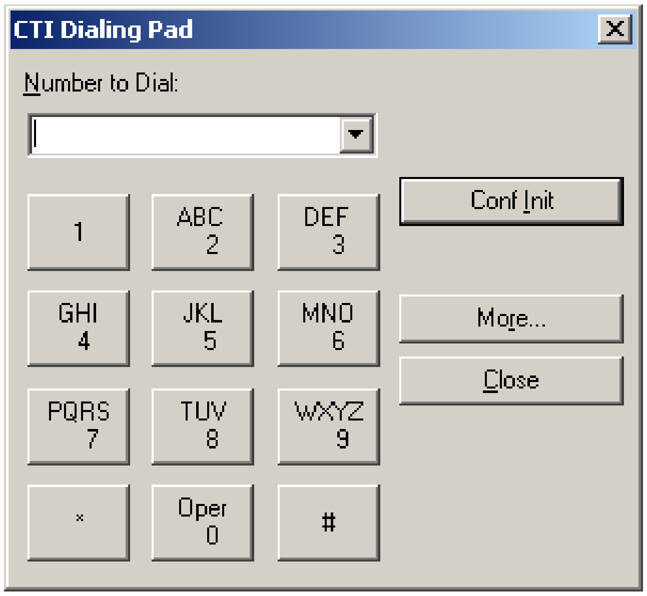


Figure 5-15 Icon for Conference Complete:



Depending on call status, selecting the Conference button once will bring up the dialog shown in Figure 5-16 (see also MakeCall dialog):

Figure 5-16 The Conference Init Dialog



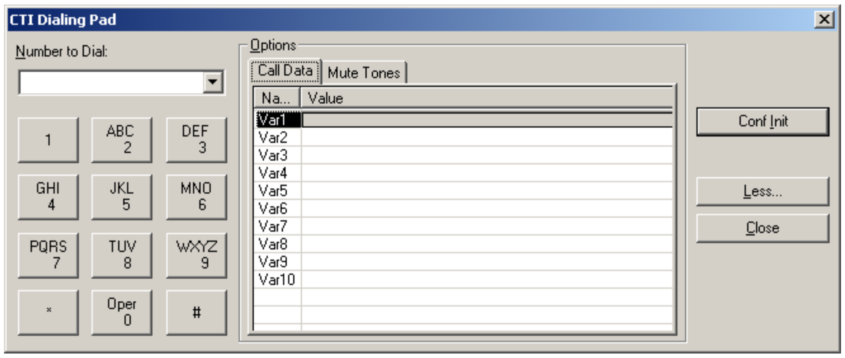
This dialog is similar to the Make Call dialog. It allows you to initiate a consultative Conference (Conf Init) or to place a Single Step Conference call.

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

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

The **More** button brings up an additional section of the dialog displaying all CallVariables along with any values set in the original call. The agent may double click on the appropriate line in the Value column to change or add values to send along with the consult call (see Figure 5-17).

Figure 5-17 Expanded Dialog



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 (see the *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted*). The main difference between the emergency call and supervisor assist requests is the script to be run. An agent may click this control whether he has a call or not. If the agent has an active customer call, clicking this button will place a consult call to the supervisor. The "Conference Complete" as well as the "Transfer Complete" will be enabled to allow the agent to either conference the supervisor into the call or to transfer the call to the supervisor. If configured, clicking this button can also cause a single step conference. The behavior (emergency call or supervisor assist) can be set via the ButtonType property set from the Property Page, as described under AgentState controls.

Figure 5-18 Emergency Icon:

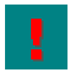
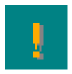


Figure 5-19 Supervisor Assist Icon:



HoldCtl

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

Figure 5-20 Hold Icon

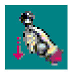


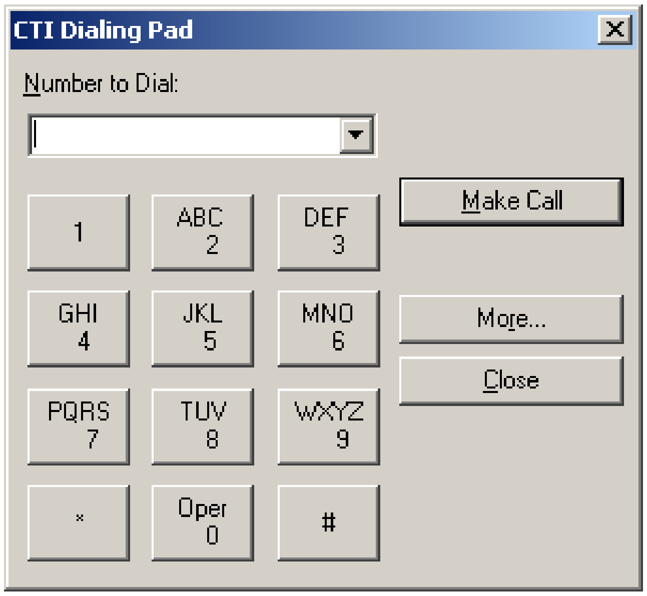
Figure 5-21 Retrieve Icon

MakeCallCtl

Figure 5-22 Make Call Iocn

The MakeCallCtl is used to place calls and to generate DTMF tones. When this button is clicked it will bring up the dialing pad dialog box to enter data and place a makecall request (Figure 5-23).

Figure 5-23 Dial Dialog

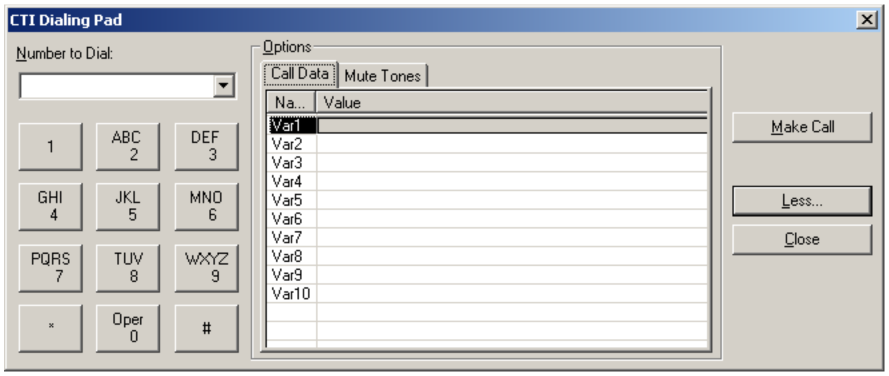


Enter the number you wish to dial by either typing it into the textbox labeled **Number to Dial** or click the numbers on the displayed keypad. Once the number is entered you can click on **Make Call** to send the MakeCall request.

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

You may enter values for CallVariable1 through CallVariable10 and ECC Call Variables via the Dial Dialog. Click the **More** button on the dialog extends to display a grid listing all possible Call Variables. A value may be entered for each of these variables by double clicking on the appropriate line in the Value column (see Figure 5-24).

Figure 5-24 Expanded Dialog



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

ReconnectCtl

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

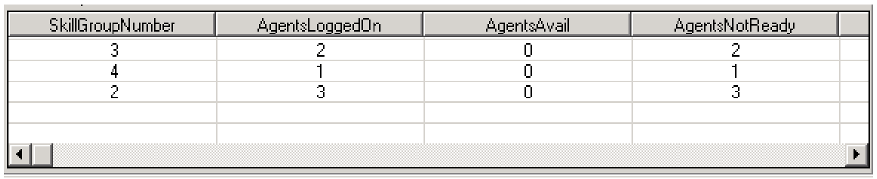
SkillgroupStatisticsCtl

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

The columns displayed are configurable at CTI OS server (see the *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted*). The update interval can be configured but defaults to 10 seconds.

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

Figure 5-25 SkillgroupStatisticsCtl Displaying Sample Data for Three Skillgroups



SkillGroupNumber	AgentsLoggedOn	AgentsAvail	AgentsNotReady
3	2	0	2
4	1	0	1
2	3	0	3

Methods

Table 5-8 Available Methods for SkillgroupStatisticsCtl

Method	Description
get_UserDefinedCell	If the column type is user defined, gets the text from the requested cell.
GetCellText	Gets the text from the requested cell in requested row.
GetColumnInfo	Gets the information about the requested column.
set_ColumnHeader	Sets the column header of requested column with given text.
set_ColumnType	Sets the column type of requested column with given value.
set_ColumnWidth	Sets the column width of requested column with given value.
set_ UserDefinedCell	Sets the given text into the requested cell.
SetColumnInfo	Sets the given information for the requested column.

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

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.

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: GetColumnInfo(System.Int16 nCol, System.Int32 plcoltype, System.Int32 icolwidth, System.String bstrcoltitle)

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.

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 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)

.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 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)

.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 Value

None.

SetColumnInfo

Sets the given information for the requested column.

Syntax

COM: HRESULT SetColumnInfo([in] short nCol, [in] long lColType, [in] int iColWidth, [in] int iColTextAlign, [in] BSTR bstrColTitle)

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.

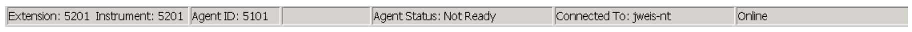
Return Value

None.

StatusBarCtl

The CTI OS statusbar control displays information about the logged on agent as well as CTI OS specific details (Figure 5-26).

Figure 5-26 StatusBar Control Displaying Sample Data



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

- Pane 1: displays current extension and instrument
- Pane 2: displays Agent ID
- Pane 3: Message Waiting Indicator. If media termination is used and Voicemail is active, this pane will display “Voicemail” to indicate Voicemail was left.
- Pane 4: displays Agent State

- Pane 5: displays the CTI OS server currently connected to
- Pane 6: displays overall status (online, offline)

SupervisorOnlyCtl

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

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

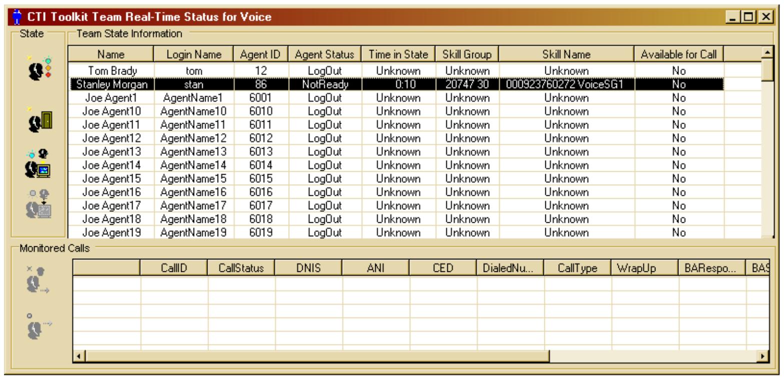
Set Monitored Agent Ready: Forces the currently monitored agent from the “not ready” state into the ready state:

Barge-In: Lets the supervisor participate in the currently monitored call. The currently monitored call is selected via the CallAppearanceCtl (in monitor mode). Barge-in is really a conference on behalf of the monitored agent

Intercept: Intercept can only be applied on a previously barged in call. The monitored agent will be dropped out of the call and the supervisor is left with the customer in a call.

Together with the AgentSelectCtl and the CallAppearanceCtl (in monitor mode) the SupervisorOnlyCtl is used in the CTI OS Supervisor Desktop application to build the Agent Real Time Status window, as shown in Figure 5-27.

Figure 5-27 Supervisor Softphone Agent-RealTime Status Window



Name	Login Name	Agent ID	Agent Status	Time in State	Skill Group	Skill Name	Available for Call
Tom Brady	tom	12	LogOut	Unknown	Unknown	Unknown	No
Stanley Morgan	stan	86	NotReady	0:10	20747 30	000923760272 VoiceSG1	No
Joe Agent1	AgentName1	6001	LogOut	Unknown	Unknown	Unknown	No
Joe Agent10	AgentName10	6010	LogOut	Unknown	Unknown	Unknown	No
Joe Agent11	AgentName11	6011	LogOut	Unknown	Unknown	Unknown	No
Joe Agent12	AgentName12	6012	LogOut	Unknown	Unknown	Unknown	No
Joe Agent13	AgentName13	6013	LogOut	Unknown	Unknown	Unknown	No
Joe Agent14	AgentName14	6014	LogOut	Unknown	Unknown	Unknown	No
Joe Agent15	AgentName15	6015	LogOut	Unknown	Unknown	Unknown	No
Joe Agent16	AgentName16	6016	LogOut	Unknown	Unknown	Unknown	No
Joe Agent17	AgentName17	6017	LogOut	Unknown	Unknown	Unknown	No
Joe Agent18	AgentName18	6018	LogOut	Unknown	Unknown	Unknown	No
Joe Agent19	AgentName19	6019	LogOut	Unknown	Unknown	Unknown	No

	CallID	CallStatus	DNIS	ANI	CED	DialedNu...	CallType	WrapUp	BARespo...	BAS

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

Start Silent Monitor: Initiates a silent monitor session with the currently monitored agent

Stop Silent Monitor: Terminates the currently ongoing silent monitored session

RecordCtl

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

Icon for Record Start:

Icon for Record Stop:

TransferCtl

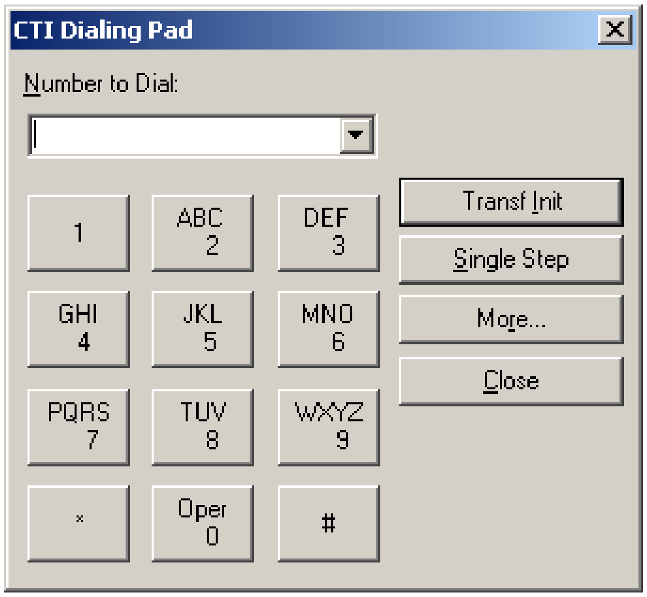
The TransferCtl is a button that provides UI to transfer a call in single step or consultative mode. The mechanism is the same as explained for the conference control.

Icon for TransferInit:

Icon for Transfer Complete:

Depending on call status, selecting the Transfer button once will bring up the dialog shown in Figure 5-28 (see also MakeCall dialog):

Figure 5-28 Dial Dialog



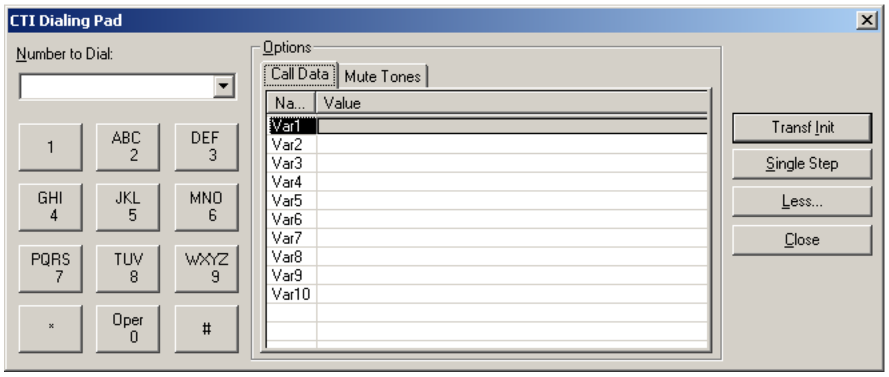
This dialog is similar to the Make Call dialog. It allows you to initiate a consultative Transfer (Transfer Init) or to place a Single Step Transfer call.

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

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

The **More** button brings up an additional section of the dialog displaying all CallVariables along with any values set in the original call. The agent may change or add values to send along with the consult call by double clicking on the appropriate line in the Value column (see Figure 5-17).

Figure 5-29 Expanded Dialog



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. The ComObject can be used 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 four methods (displayed in IDL format; IDL is the language used to define COM interfaces).

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

    };
```

Connect

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

Disconnect

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

StartMonitoring

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

Table 5-9 StartMonitoring Arguments Array Parameters

Keyword	Value
AgentID	AgentID of the agent to be monitored.
Peripheralnumber	PeripheralID of the Peripheral to which the Agent is logged in.

Note If a pointer to the agent object is available (e.g., a m_MonitoredAgent), The PeriperhalID can be retrieved 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 (Table 5-9).

SilentMonitor Com Object Events

The ComObject will fire 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 Chapter 6, "Event Interfaces and Events."

Table 5-10 SilentMonitor Com Object Events

Event	Description
OnConnection	Indicates that the connect method was successful in establishing a connection.
OnConnectionFailure	Indicates that an active connection has failed. Can also indicate a bad parameter in the Connect method.
OnMonitorModeEstablished	Signals a successful call to SetMsgFilte. The call to Setmsgfilter is hidden by the Standalone control.
OnConnectionClosed	Disconnect was called and the connection is now closed.
OnControlFailureConf	A ControlFailureConf was received and can be handled.
OnCtiOsFailure	A CtiosFailure event was received. This could be Silent Monitor specific error code.
OnCallRTPStartedEvent, OnCallRTPStoppedEvent	RTP events have been received signaling the start and stop of the RTP streams.
OnSilentMonitorStatusReport Event	This event is used to report status from a monitored client to the monitoring application.
OnStartSilentMonitorConf, OnStopSilentMonitorConf	These confs acknowledge that CTI OS handled the StartMonitoring and StopMonitoring request, respectively.
OnSilentMonitorSession Disconnected	Indicates that the Silent Monitor session has timed out on the monitoring side.

Deployment

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

Sample Usage in Visual Basic 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 CTI OS. It is important to note, that this control does not require supervisor privileges or even any login. The only event handler shown below (OnSessionDisconnected) is the one for a timed out Silent Monitor session. Other event handlers.

```
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
```

CHAPTER 6

Event Interfaces and Events

This chapter describes the CTI OS Client Interface Library's event publications mechanism. Programs written to take advantage of CTI interfaces are generally event driven, meaning that 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

Note The CIL event interfaces discussed in this section and the following sections apply only to the C++, COM, and VB interfaces. See "Events in Java CIL" for a discussion of Java CIL counterpart events and event handling in the Java CIL See "Events in .NET CIL", for a discussion of .NET CIL event handling.

The Client Interface Library provides a publisher-subscriber model for notifying clients of events. Client applications using the CIL can subscribe to one or more of the CIL event interfaces. For detailed information and examples for how to subscribe and unsubscribe for events, see Chapter 4, "Building Your Application."

The published CIL event interfaces are organized around the different classes of CTI objects that the CIL provides. The event interfaces described in this chapter are:

- **ISessionEvents.** This interface publishes the events that relate to actions on the Session object.
- **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.

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

ISessionEvents Interface

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

OnConnection

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

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 6-1 ISession Event Parameters

Keyword	Type	Description
EventTime	INT	Integer value with time of day expressed in milliseconds.
CurrentServer	STRING	Name or TCP/IP address of the current connected CTI OS server.

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 was initiated by the client (CIL or application using the CIL). 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 can be followed by an OnConnection event when connection is successfully reestablished. The client application need not take any special action but may use this event to display connection status.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-2 OnConnectionFailure Event Parameters

Keyword	Type	Description
EventTime	INT	Integer value with time of day expressed in milliseconds.
FailedServer	STRING	Name or TCP/IP address of the server that has failed to respond. See ReasonCode.
ReasonCode	SHORT	SERVER_CONNECTIONBROKEN, SERVER_MISSINGHEARTBEATS

OnConnectionRejected

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

Syntax

C++: `void OnConnectionRejected (Arguments& args)`
COM: `void OnConnectionRejected (IArguments * args)`
VB: `Session_OnConnectionRejected (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Not currently used, reserved for future use.

OnCTIOSFailure

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

Syntax

C++: `void OnCTIOSFailure (Arguments& args)`
COM: `void OnCTIOSFailure (IArguments * args)`
VB: `Session_OnCTIOSFailure (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-3 OnCTIOSFailure Event Parameters

Keyword	Type	Description
FailureCode	INT	A value according to an enumerated value, as shown immediately following this table.
SystemEventID	INT	Present only if FailureCode equals ServerConnectionStatus. Contains a value according to an enumerated value, as shown immediately following this table.
SystemEventArg1	INT	Present only if SystemEventID equals SysPeripheralOnline or SysPeripheralOffline. Contains the peripheral ID of the affected peripheral.
ErrorMessage	STRING	An error message.

Following are the enumerated values for Failure Code:

```
enum enumCTIOS_FailureCode
  {
      eDriverOutOfService                =   1,
      eServiceNotSupported               =   eDriverOutOfService + 1,
      eOperationNotSupported             =   eServiceNotSupported + 1,
      eInvalidPriviledge                 =   eOperationNotSupported + 1,
      eUnknownRequestID                  =   eInvalidPriviledge + 1,
      eUnknownEventID                    =   eUnknownRequestID + 1,
      eUnknownObjectID                   =   eUnknownEventID + 1,
      eRequiredArgMissing                =   eUnknownObjectID + 1,
      eInvalidObjectState                =   eRequiredArgMissing
      eServerConnectionStatus            =   eInvalidObjectState + 1,
      eInconsistentAgentData             =   eServerConnectionStatus + 1,
      eAgentAlreadyLoggedIn              =   eInconsistentAgentData + 1,
      eForcedNotReadyForConfigError      =   eAgentAlreadyLoggedIn + 1
      eMonitorModeConnectionDenied       =   eForcedNotReadyForConfigError + 1
  };
```

Following are the enumerated values for SystemEventID:

```
enum enumCTIOS_SystemEventID
  {    eSysCentralControllerOnline        =   1,
       eSysCentralControllerOffline       =   2,
       eSysPeripheralOnline               =   3,
       eSysPeripheralOffline              =   4,
       eSysTextFYI                        =   5,
       eSysPeripheralGatewayOffline       =   6,
       eSysCtiServerOffline               =   7,
       eSysCTIOSServerOnline              =   8,
       eSysHalfHourChange                 =   9,
       eSysInstrumentOutOfService         =   10,
       eSysInstrumentBackInService        =   11,
       eSysCtiServerDriverOnline          =   eSysInstrumentBackInService + 1,
       eSysCtiServerDriverOffline         =   eSysCtiServerDriveOnline + 1,
       eSysCTIOSServerOffline             =   eSysCtiServerDriverOffline + 1,
       eSysCTIOSServerOnline              =   eSysCTIOSServerOffline + 1,
       eSysAgentSummaryStatusOnline       =   eSysCTIOSServerOnline + 1,
       eSysAgentSummaryStatusOffline      =   eSysAgentSummaryStatusOnline + 1
  }
```

Remarks

See the descriptions of the CtiOs_Enums.FailureCode and CtiOs_Enums.SystemEvent interfaces in the Javadoc for information on Java CIL enumerations.

OnCurrentAgentReset

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

Syntax

C++: `void OnCurrentAgentReset(Arguments& args)`
COM:`void OnCurrentAgentReset (IArguments * args)`
VB: `session_OnCurrentAgentReset (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-4 OnCurrentAgentReset Parameters

Keyword	Type	Description
UniqueObjectID	STRING	Unique object ID (if any) of the old current agent that was just removed.

OnCurrentCallChanged

The OnCurrentCallChanged message is generated when the current call has changed to another call.

Syntax

C++: `void OnCurrentCallChanged(Arguments& args)`
COM: `void OnCurrentCallChanged (IArguments * args)`
VB: `session_OnCurrentCallChanged (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-5 OnCurrentCallChanged Parameters

Keyword	Type	Description
UniqueObjectID	STRING	Unique object ID (if any) of the new current call.

OnFailure Event

Not supported.

OnGlobalSettingsDownloadConf

You can configure the client once 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. Refer to the *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted* for more detailed information.

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.

To gain an understanding of what is available and how to configure these items, see the following sections in the *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted*.

- 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 CTI OS server. For a description of the Enterprise Desktop Settings values listed below, see the *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted.*

The following are the top level elements in the Enterprise Desktop Settings registry key. The CTI OS server passes configuration data for these elements to the client in the OnGlobalSettingsConf event.

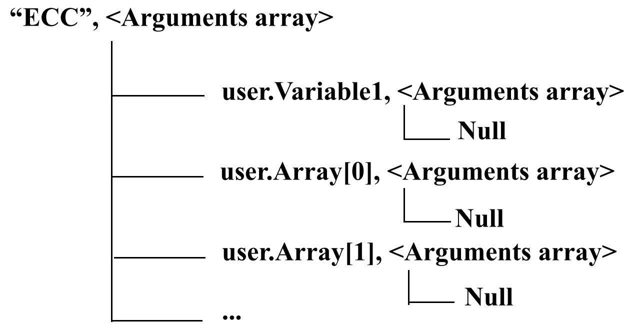
- ECC (Expanded Call Context) variables
- Grid
- IPCCSilentMonitor
- Login
- ScreenPreferences
- SoundPreferences

Other keys or values that are added to the EnterpriseDesktopSettings/All Desktops key in the CTI OS 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 CTI OS server in the "ECC/Name" registry subtree.

  The CTI OS server does not send any registry information contained in the CTI OS 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.


"ECC", <Arguments array>
user.Variable1, <Arguments array>
Null
user.Array[0], <Arguments array>
Null
user.Array[1], <Arguments array>
Null
...

  Each ECC scalar configured in the CTI OS server registry is represented as an empty Arguments array with keyword "user.<name>", where <name> is the ECC name as configured on CTI OS server.

Each ECC array configured in the CTI OS 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 CTI OS server and n is the size of the array as configured on the CTI OS server.

- **Grid –** Arguments array that contains information from the CTI OS 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 6-6 Agent Statistics Coloumn Number: Key Values

Keyword	Data Type
Type	string
Header	string
Custom values[1]	custom

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:

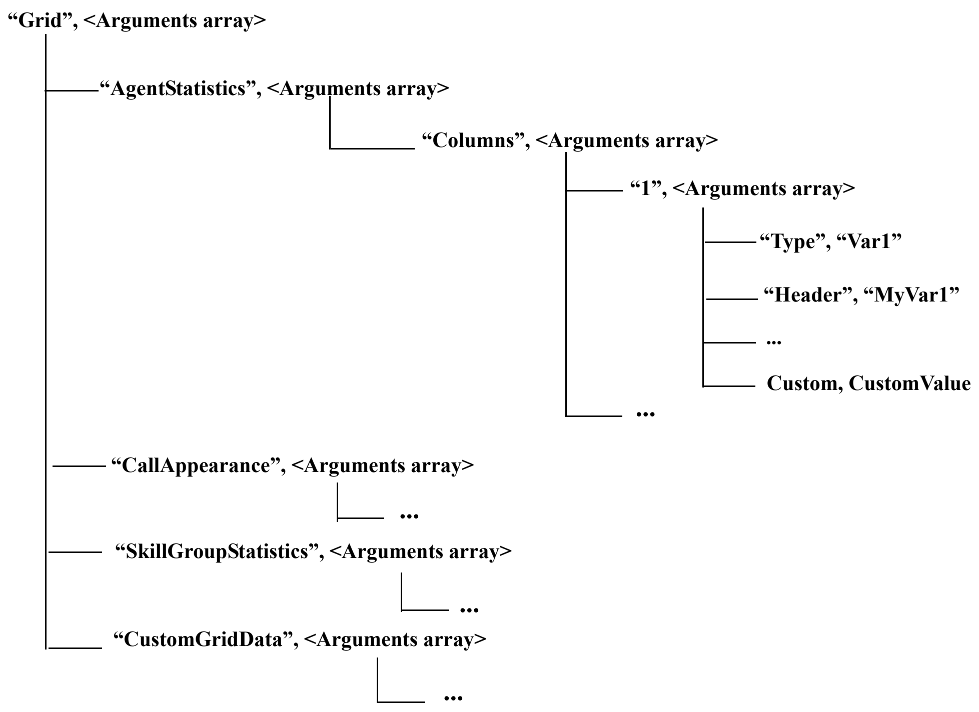
Table 6-7 CallAppearance Coloumn Number: Key Values

Keyword	Data Type
Type	string
Header	string
editable	boolean
maxchars	integer
Custom values[1]	custom

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.

You can add custom keys in the CTI OS Server registry's Grid subtree at the same level as the SkillGroupStatistics, AgentStatistics, and CallAppearance keys. The Grid Arguments array within this event will 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:


"Grid", <Arguments array>
"AgentStatistics", <Arguments array>
"Columns", <Arguments array>
"1", <Arguments array>
"Type", "Var1"
"Header", "MyVar1"
...
Custom, CustomValue
...
"CallAppearance", <Arguments array>
...
"SkillGroupStatistics", <Arguments array>
...
"CustomGridData", <Arguments array>
...

- **IPCCSilentMonitor** – Arguments array that contains configuration information from the CTI OS 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 6-8 IPCCSilentMonitor: Key Values

Keyword	Value
MediaTerminationPort	integer
HeartBeatInterval	integer
TOS	boolean
MonitoringIPPort	integer
HeartbeatTimeout	integer
CCMBasedSilentMonitor	boolean

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 will 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.

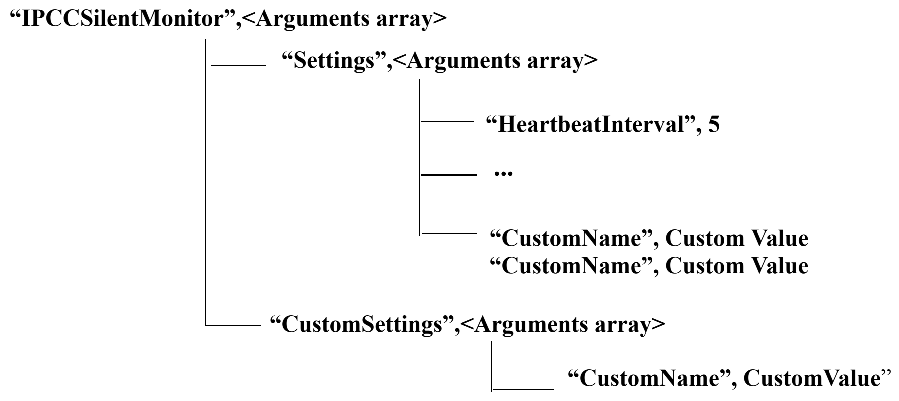
Two silent monitoring types are supported for Unified CCE:

- CTI OS based
- CCM based

The silent Monitor type used by CTI OS is configured 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 follows:


"IPCCSilentMonitor",<Arguments array>
"Settings",<Arguments array>
"HeartbeatInterval", 5
...
"CustomName", Custom Value
"CustomName", Custom Value
"CustomSettings",<Arguments array>
"CustomName", CustomValue"

- **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 6-9 Unified CCE Agent Statistics: Key Values

Keyword	Value
CtiOsA	string
CtiOsB	string
PortA	integer
PortB	integer
Heartbeat	integer
MaxHeartbeats	integer
AutoLogin	boolean
WarnIfAlreadyLoggedIn	boolean

Keyword	Value
ShowFieldBitMask	integer
RejectIfAlreadyLoggedIn	boolean
PeripheralID	integer
IPCCSilentMonitorEnabled	boolean
TOS	boolean
SwitchCapabilityBitMask	integer
WarnIfSilentMonitored	boolean
RasCallMode[1]	integer

1.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 follows.

```
"Login",<Arguments array>
    |
    |____"ConnectionProfiles",<Arguments array>
                         |
                         |
                         |_____"IPCC",<Arguments array>
                         |            |_____"CtiOsA", Machine1"
                         |            |_____"PortA", 42028
                         |            |_____ ...
                         |            |
                         |            |_____"CustomName", "CustomValue"
                         |_____"Aspect",<Arguments array>
                         |            |_____"CtiOsA", "Machine2"
                         |            |_____"PortA", 42028
                         |            |_____ ...
                         |            |
                         |            |_____"CustomName", "CustomeValue"
                         |
                         |
                         |_____ ...
```

SilentMonitorService Subkey

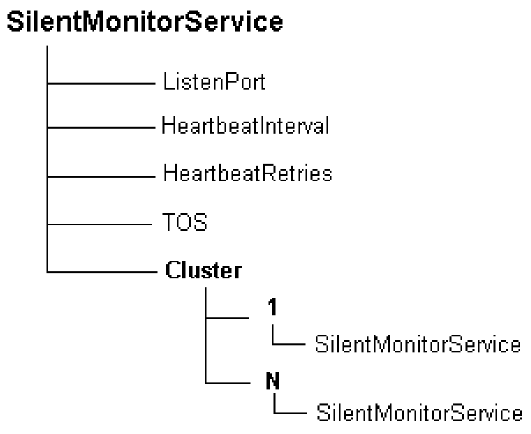
The <profile_name>/SilentMonitorService subkey contains parameters that clients use to connect to one of a set of silent monitor services. It contains the following keys.

Note The SilentMonitorService subkey is only applicable to CTI OS based silent monitor.

Table 6-10 SilentMonitorService Parameters

Keyword	Value	Description
ListenPort	integer	Port on which the silent monitor service is listening for incoming connections.
TOS	integer	QOS setting for the connection.
HeartbeatInterval	integer	Amount of time in milliseconds between heartbeats.
HeartbeatRetries	integer	Number of missed heartbeats before the connection is abandoned.
Cluster		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 adress of the silent monitor service.

The following diagram illustrates the hierarchy of the SilentMonitorService subkey.



- **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:

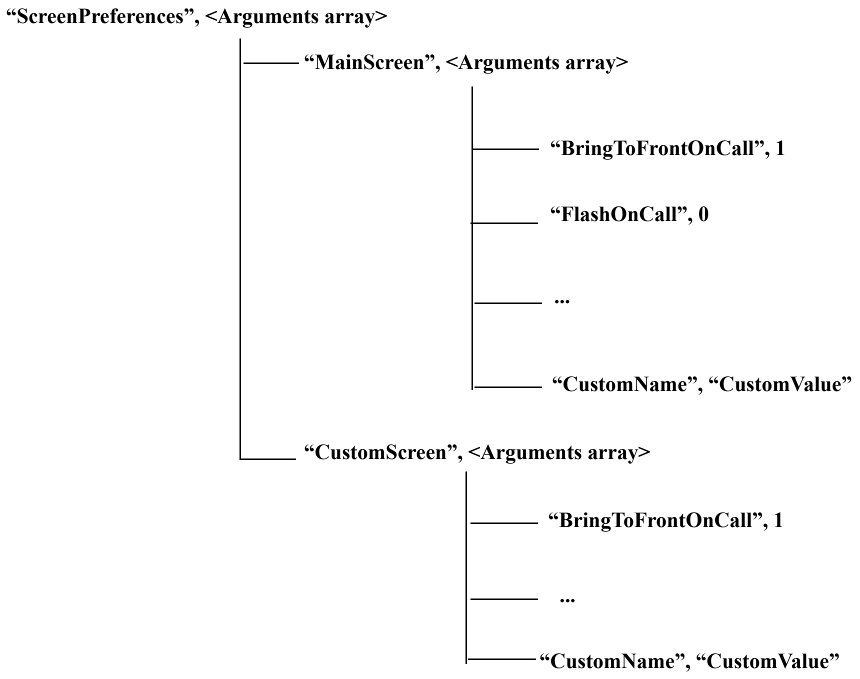
Table 6-11 ScreenPreferences: Key Values

Keyword	Value
AgentStatisticsIntervalSec	integer
BringToFrontOnCall	boolean
FlashOnCall	boolean
RecordingEnabled	boolean

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 will contain 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 follows.


"ScreenPreferences", <Arguments array>
"MainScreen", <Arguments array>
"BringToFrontOnCall", 1
"FlashOnCall", 0
...
"CustomName", "CustomValue"
"CustomScreen", <Arguments array>
"BringToFrontOnCall", 1
...
"CustomName", "CustomValue"

- **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 may also include custom name/value pairs for a custom application.

  The SoundPreferences array contains the following key-value pairs:

Table 6-12 SoundPreferences: Key Values

Keyword	Value	Subtree
DTMF*	Arguments array	SoundPreferences/Name/DTMF
DialTone*	Arguments array	SoundPreferences/Name/DialTone
OriginatingTone*	Arguments array	SoundPreferences/Name/OriginatingTone
RingInTone*	Arguments array	SoundPreferences/Name/RingInTone
All*	Arguments array	SoundPreferences/Name/All

* 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 follows.

```
"SoundPreferences",<Arguments array>
                   |____ "All",<Arguments array>
                   |               |____ "Mute", 0
                   |               |____ "CustomName", CustomValue
                   |____ "DTMF",<Arguments array>
                   |               |____ "Mute", 0
                   |               |____ "CustomName", CustomValue
                   |
                   |____ "CustomerName",<Arguments array>
                   |                       |
                   |                       |____ "CustomeName", CustomValue
                   |
                   |____ ...
```

This configuration is stored in the Windows System Registry database and many of the values are set when the CTI OS Server Setup is run. Custom configuration can be set 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 6-13 Heartbeat Parameters

Keyword	Type	Description
EventTime	INT	Integer value with time of day expressed in milliseconds.

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++:  void OnMissingHeartbeat(Arguments& args)
COM:  void OnMissingHeartbeat (IArguments * args)
VB:   session_OnMissingHeartbeat (ByVal args As CtiosCLIENTLib.IArguments)
```

Parameters

args

Arguments array containing the following fields.

Table 6-14 OnMissingHeartbeat Parameters

Keyword	Type	Description
EventTime	INT	Integer value with time of day expressed in milliseconds.
Consecutive MissedHeartbeats	INT	Integer value with the number of heartbeats missed.
HeartbeatInterval	INT	Integer value with the heartbeat interval, in milliseconds.

OnMonitorModeEstablished

The OnMonitorModeEstablished event is generated when Monitor Mode is established.

Syntax

C++:`void OnMonitorModeEstablished(Arguments& args)`
COM: `void OnMonitorModeEstablished (IArguments * args)`
VB: `session_OnMonitorModeEstablished (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-15 OnMonitorModeEstablished Parameters

Keyword	Type	Description
CIL ConnectionID	STRING	ID of the client's connection on the server.
StatusSystem	ARGUMENTS	Arguments array containing the following elements: • StatusCTIServer • StatusCtiServerDriver • StatusCentralController • StatusPeripherals (Arguments array with a peripheral ID for each key and a boolean true/false value indicating if that peripheral is online)

OnSnapshotDeviceConf

The OnSnapshotDeviceConf confirmation message is fired to the client as part of a snapshot operation. For AgentMode clients, the OnSnapshotDeviceConf will arrive at startup time, after the OnQueryAgentStateConf message. The OnSnapshotDeviceConf indicates the number of calls present at the device, and their UniqueObjectIDs.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-16 OnSnapshotDeviceConf Parameters

Keyword	Description	Type
UniqueObjectID	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.	STRING
NumCalls	The number of active calls associated with this device, up to a maximum of 16.	SHORT
ValidCalls	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.	ARGUMENTS

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 would be an agent talking on a call on a hardphone and a CTI failure occurs. The agent hangs up the call before CTI is recovered. Once CTI and the CIL recover, they use the snapshot to discover that the call it currently has is no longer on the device. CTI then fires an EndCall event to remove the call from its call list.

OnSnapshotSkillGroupList

Not supported.

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's 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 be able to get and set data, which enables some powerful data-prefetching applications. When a OnCallBeginEvent follows for this same contact, the `ICMEnterpriseUniqueID` field will be send along with the call data. At that point, a custom application can set the call data on the appropriate call object.

Syntax

```
C++:void OnTranslationRoute(Arguments& args)
COM:void OnTranslationRoute(IArguments * args)
VB:    session_OnTranslationRoute(ByVal args As CtiosCLIENTLib.IArguments)
```

Parameters

args

Arguments array containing the following fields.

Table 6-17 OnTranslationRoute Parameters

Keyword	Type	Description
ICMEnterpriseUniqueID	STRING	This string is a globally unique key for this contact, which corresponds to the Unified ICM 64 bit key. This parameter can be used to match this contact to a follow-on call event.
RouterCallKeyDay	INT	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.
RouterCalKeyCallID	INT	The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.
RouterCallKey SequenceNumber	INT	Together with RouterCallKeyDay and RouterCallKeyCallID fields forms the TaskID.
NumNamedVariables	SHORT	Number of Named variables.
NumNamedArrays	SHORT	Number of Named Arrays.
ANI	STRING	The calling line ID of the caller.
UserToUserInfo	STRING	The ISDN user-to-user information element.
DNIS	STRING	The DNIS number to which this call will arrive on the ACD/PBX.
DialedNumber	STRING	The number dialed.
CallerEnteredDigits	STRING	The digits entered by the caller in response to IVR prompting.
CallVariable1	STRING	Call-related variable data.
…		…
CallVariable10	STRING	Call-related variable data.
ECC	ARGUMENTS	A nested Arguments structure of key-value pairs for all of the ECC data arriving with this call.

ICallEvents Interface

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

Note Many of the parameters that CTI OS receives from the CTI Server are inconsequential to most customer applications. The most important parameters for doing a screenpop are included with the events described in this section. The more inconsequential parameters are suppressed at the CTI OS Server, to minimize network traffic to the clients. However, you can enable the complete set of available event arguments by setting the following registry setting:

[HKLM\Cisco Systems\CTIOS\Server\CallObject\MinimizeEventArgs = 0].

OnAgentPrecallEvent

Note 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, 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's 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 be able to get and set data, which enables some powerful data-prefetching applications. When an OnCallBeginEvent follows for this same contact, the `ICMEnterpriseUniqueID` field will be send along with the call data. At that point, a custom application can set the call data on the appropriate call object.

Syntax

```
C++:void OnAgentPrecallEvent(Arguments& args)
COM:  void OnAgentPrecallEvent (IArguments * args)
VB:   session_OnAgentPrecallEvent (ByVal args As CtiosCLIENTLib.IArguments)
```

Parameters

args

Arguments array containing the following fields.

Table 6-18 OnAgentPrecallEvent Parameters

Keyword	Type	Description
ICMEnterpriseUniqueID	STRING	This string is a globally unique key for this contact, which corresponds to the Unified ICM 64 bit key. This parameter can be used to match this contact to a follow-on call event.
RouterCallKeyDay	INT	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.
RouterCalKeyCallID	INT	The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.
AgentInstrument	STRING	The agent instrument that the call will be routed to.
NumNamedVariables	SHORT	Number of Named variables.
NumNamedArrays	SHORT	Number of Named Arrays.
ServiceNumber	INT	The service that the call is attributed to, as known to the peripheral.
ServiceID	INT	The Unified ICM ServiceID of the service that the call is attributed to.
SkillGroupNumber	INT	The number of the agent SkillGroup the call is attributed to, as known to the peripheral.
SkillGroupID	INT	The Unified ICM SkillGroupID of the agent SkillGroup the call is attributed to.
SkillGroupPriority	SHORT	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
ANI	STRING	The calling line ID of the caller.
UserToUserInfo	STRING	The ISDN user-to-user information element.
DNIS	STRING	The DNIS number to which this call will arrive on the ACD/PBX.
DialedNumber	STRING	The number dialed.
CallerEnteredDigits	STRING	The digits entered by the caller in response to IVR prompting.
CallVariable1	STRING	Call-related variable data.
…		…
CallVariable10	STRING	Call-related variable data.
ECC	ARGUMENTS	A nested Arguments structure of key-value pairs for all of the ECC data arriving with this call.

CallTypeIDTag	INT	Specifies CallType of the call and indicates that the agent is reserved via LegacyPreCall.
PreCallInvokeIDTag	INT	Specifies the invoking of the LegacyPreCall.

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's 64-bit unique key (`RouterCallKeyDay` and `RouterCallKeyCallID`). Upon receipt of an OnAgentPrecallAbortEvent, any data pre-fetch work that was started on an OnAgentPrecallEvent should be cleaned up.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-19 OnAgentPrecallAbortEvent Parameters

Keyword	Type	Description
ICMEnterpriseUniqueID	STRING	This string is a globally unique key for this contact, which corresponds to the Unified ICM 64 bit key. This parameter can be used to match this contact to a follow-on call event.
RouterCallKeyDay	INT	Together with the RouterCallKey CallID 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.
RouterCalKeyCallID	INT	The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.
AgentInstrument	STRING	The agent instrument that the call will be routed to.

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 6-20 OnAlternateCallConf Parameters

Keyword	Type	Description
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.

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 6-21 OnAnswerCallConf Parameters

Keyword	Type	Description
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.

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 signalled by an OnCallDataUpdate event containing the changed call data.

Note There can be multiple calls with the same ConnectionCallID value.

Syntax

C++: `void OnCallBegin(Arguments& args)`
COM: `void OnCallBegin (IArguments * args)`
VB: `session_OnCallBegin (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-22 OnCallBegin Parameters

Keyword	Type	Description
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where the call activity occurred.
PeripheralType	SHORT	The type of the peripheral.
CallType	SHORT	The general classification of the call type.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
RouterCallKeyDay	INT	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.
RouterCalKeyCallID	INT	The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.
RouterCallKey SequenceNumber	INT	Together with RouterCallKeyDay and RouterCallKeyCallID fields forms the TaskID.
ConnectionCallID	UINT	The Call ID value assigned to this call by the peripheral or the Unified ICM.
ANI (optional)	STRING	The calling line ID of the caller.
DNIS (optional)	STRING	The DNIS provided with the call.
UserToUserInfo (Optional)	STRING	The ISDN user-to-user information element. unspecified, up to 131 bytes.
DialedNumber (Optional)	STRING	The number dialed.
CallerEnteredDigits (Optional)	STRING	The digits entered by the caller in response to IVR prompting.
ServiceNumber (Optional)	INT	The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.

Keyword	Type	Description
ServiceID (Optional)	INT	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.
SkillGroupNumber (Optional)	INT	The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_ GROUP when not applicable or not available.
SkillGroupID (Optional)	INT	The Unified ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority (Optional)	SHORT	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
CallWrapupData (Optional)	STRING	Call-related wrap-up data.
CallVariable1 (Optional)	STRING	Call-related variable data.
…		…
CallVariable10 (Optional)	STRING	Call-related variable data.
CallStatus (optional)	SHORT	The current status of the call.
ECC (optional)	ARGUMENTS	Arguments array that contains all of the Expanded Call Context variables in use; for example: user.ArrayVariable[0] user.ArrayVariable[1] ... user.ArrayVariable[n] user.ScalarVariable
CTIClients (optional)	ARGUMENTS	Arguments array that contains the information about the number of clients that are using the call object; for example: CTIClient[1] CTIClientSignature CTIClientTimestamp
ICMEnterprise UniqueID (optional)	STRING	Required only when the call is pre-routed.

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.

Note If the CallCleared event is received after having received a CallFailed event, note that the event will not include a CallStatus since 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 CallCleared event is optional.

Syntax

C++:`void OnCallDelivered(Arguments& args)`
COM: `void OnCallCleared (IArguments * args)`
VB: `session_OnCallCleared (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-23 OnCallCleared Parameters

Keyword	Type	Description
EnablementMask	INT	Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
CallStatus	SHORT	The current status of the call.
ICMEnterprise UniqueID (Optional)	STRING	Required only when the call is pre-routed.

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, note that the event will not include a CallStatus since 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_OnCallConectionCleared (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-24 OnCallConnectionCleared Parameters

Keyword	Type	Description
EnablementMask	INT	Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
CallStatus	SHORT	The current status of the call.
ICMEnterprise UniqueID (Optional)	STRING	Required only when the call is pre-routed.

OnCallConferenced

The joining of calls into a conference call or the adding of a new call joining a conference may generate an OnCallConferenced event. With this event, the connections at the controller's device merge to become one connection with a status of LCS_CONNECT, and the status of the connections at the original caller's device and at the consulted device remain unchanged.

Syntax

C++:`void OnCallConferenced(Arguments& args)`
COM: `void OnCallConferenced (IArguments * args)`
VB: `session_OnCallConferenced (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-25 OnCallConferenced Parameters

Keyword	Type	Description
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where the call activity occurred.
PeripheralType	SHORT	The type of the peripheral.
CallType	SHORT	The general classification of the call type.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.

Keyword	Type	Description
RouterCallKeyDay	INT	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.
RouterCalKeyCallID	INT	The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.
ConnectionCallID	UINT	The Call ID value assigned to this call by the peripheral or the Unified ICM.
ANI (optional)	STRING	The calling line ID of the caller.
DNIS (optional)	STRING	The DNIS provided with the call.
UserToUserInfo (Optional)	STRING	The ISDN user-to-user information element. unspecified, up to 131 bytes.
DialedNumber (Optional)	STRING	The number dialed.
CallerEnteredDigits (Optional)	STRING	The digits entered by the caller in response to IVR prompting.
ServiceNumber (Optional)	INT	The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.
ServiceID (Optional)	INT	The 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.
SkillGroupNumber (Optional)	INT	The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupID (Optional)	INT	The Unified ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority (Optional)	SHORT	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
CallWrapupData (Optional)	STRING	Call-related wrap-up data.
CallVariable1 (Optional)	STRING	Call-related variable data.
…		…
CallVariable10 (Optional)	STRING	Call-related variable data.
CallStatus (optional)	SHORT	The current status of the call.

Keyword	Type	Description
ECC (optional)	ARGUMENTS	Arguments array that contains all of the Expanded Call Context variables in use; for example: user.ArrayVariable[0] user.ArrayVariable[1] ... user.ArrayVariable[n] user.ScalarVariable
CTIClients (Optional)	ARGUMENTS	Arguments array that contains the information about the number of clients that are using the call object; for example: CTIClient[1] CTIClientSignature CTIClientTimestamp
ICMEnterpriseUnique ID (Optional)	STRING	Required only when the call is pre-routed.

OnCallDataUpdate

Changes to the call context data will generate an OnCallDataUpdate event. Only the items that have changed will be in the event argument array. The initial call context is provided in the OnCallBegin event.

Syntax

C++: `void OnCallDataUpdate(Arguments& args)`
COM: `void OnCallDataUpdate (IArguments * args)`
VB: `session_OnCallDataUpdate (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-26 OnCallUpdate Parameters

Keyword	Type	Description
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where the call activity occurred.
PeripheralType	SHORT	The type of the peripheral.
CallType	SHORT	The general classification of the call type.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
RouterCallKeyDay	INT	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.
RouterCalKeyCallID	INT	The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.
RouterCallKey SequenceNumber	INT	Together with RouterCallKeyDay and RouterCallKeyCallID fields forms the TaskID.
ConnectionCallID	UINT	The Call ID value assigned to this call by the peripheral or the Unified ICM.
ANI (optional)	STRING	The calling line ID of the caller.
DNIS (optional)	STRING	The DNIS provided with the call.
UserToUserInfo (Optional)	STRING	The ISDN user-to-user information element. unspecified, up to 131 bytes.
DialedNumber (Optional)	STRING	The number dialed.
CallerEnteredDigits (Optional)	STRING	The digits entered by the caller in response to IVR prompting.
ServiceNumber (Optional)	INT	The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.
ServiceID (Optional)	INT	The 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.
SkillGroupNumber (Optional)	INT	The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupID (Optional)	INT	The Unified ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority (Optional)	SHORT	The priority of the skill group, or 0 when skill group priority is not applicable or not available.

Keyword	Type	Description
CallWrapupData (Optional)	STRING	Call-related wrap-up data.
CallVariable1 (Optional)	STRING	Call-related variable data.
…		…
CallVariable10 (Optional)	STRING	Call-related variable data.
CallStatus (optional)	SHORT	The current status of the call.
ECC (optional)	ARGUMENTS	Arguments array that contains all of the Expanded Call Context variables in use; for example: user.ArrayVariable[0] user.ArrayVariable[1] ... user.ArrayVariable[n] user.ScalarVariable
CTIClients (Optional)	ARGUMENTS	Arguments array that contains the information about the number of clients that are using the call object; for example: CTIClient[1] CTIClientSignature CTIClientTimestamp
ICMEnterprise UniqueID (Optional)	STRING	Required only when the call is pre-routed.

OnCallDelivered

The OnCallDelivered event may be generated when the call arrives at the agent's teleset. Both parties (call connections) receive this event. With this event, the called party's connection status becomes LCS_ALERTING but the calling party's connection status remains LCS_INITIATE.

Note 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++: `void OnCallDelivered(Arguments& args)`
COM: `void OnCallDelivered (IArguments * args)`
VB: `session_OnCallDelivered (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-27 OnCallDelivered Parameters

Keyword	Type	Description
ServiceNumber	INT	The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.
ServiceID	INT	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.
SkillGroupNumber (Optional)	INT	The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupID (Optional)	INT	The Unified ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority (Optional)	SHORT	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
LineType	SHORT	Indicates the type of the teleset line.
EnablementMask	INT	Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call. See Table 3-2.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
CallStatus	SHORT	The current status of the call.
ICMEnterpriseUniqueID (Optional)	STRING	Required only when the call is pre-routed.
TrunkNumber (optional)	INT	The number representing a trunk.
TrunkGroup Number (optional)	INT	The number representing a trunk group.

OnCallDequeuedEvent

The explicit removal of a call from the ACD queue may 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.

Keyword	Type	Description
Connection DeviceID	INT	The identifier of the connection between the call and the device.
ConnectionDevice IDType	SHORT	Indicates the type of the connection identifier supplied in the ConnectionDeviceID.
LocalConnection State	SHORT	The state of the local end of the connection.
EventCause	SHORT	Indicates a reason or explanation for the occurrence of the event.
LineHandle	SHORT	Identifies the teleset line being used.
LineType	SHORT	Indicates the type of the teleset line.
ServiceID	INT	The Unified ICM ServiceID of the service that the call is attributed to.
ServiceNumber	INT	The service that the call is attributed to, as known to the peripheral.
NumQueued	SHORT	The number of calls in the queue for this service.
NumSkillGroups	SHORT	The number of Skill Groups that the call has been removed from, up to a maximum of 99.

OnCallDiverted

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

Syntax

C++: `void OnCallDiverted(Arguments& args)`
COM: `void OnCallDiverted (IArguments * args)`
VB: `session_OnCallDiverted (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Keyword	Type	Description
UniqueObjectID	STRING	Unique reference generated for a call at client.
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where the call activity occurred.
PeripheralType	SHORT	The type of the peripheral.
ConnectionDevice IDType	SHORT	Indicates the type of ConnectionDeviceID value.

Keyword	Type	Description
Connection DeviceID	INT	The device identifier of the connection between the call and the device.
ConnectionCallID	UINT	The Call ID value assigned to this call by the peripheral or the Unified ICM.
ServiceNumber	INT	The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.
ServiceID	INT	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.
DivertingDevice Type	SHORT	Indicates the type of the device identifier supplied in the DivertingDeviceID field.
CalledDeviceType	SHORT	Indicates the type of the device identifier supplied in the CalledDeviceID field.
LocalConnection State	SHORT	The state of the local end of the connection.
EventCause	SHORT	Indicates a reason or explanation for the occurrence of the event.
DivertingDeviceID (Optional)	STRING	The device identifier of the device from which the call was diverted.
CalledDeviceID (Optional)	STRING	The device identifier of the device to which the call was diverted.

OnCallEnd

The OnCallEnd event is generated when the association between a call and the CTI Client is dissolved. The OnCallEnd event is the last event received for a Call.

Syntax

C++: `void OnCallEnd(Arguments& args)`
COM: `void OnCallEnd (IArguments * args)`
VB: `session_OnCallEnd (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-28 OnCallEnd Parameters

Keyword	Type	Description
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
CallStatus (optional)	SHORT	The current status of the call.
ICMEnterprise UniqueID (optional)	STRING	Required only when the call is pre-routed.

OnCallEstablished

The OnCallEstablished event may be generated when the call is answered at the agent's teleset. Both parties (call connections) receive this event when the call is answered. With this event, the call status of both parties becomes LCS_CONNECT.

Note 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++: `void OnCallEstablished(Arguments& args)`
COM: `void OnCallEstablished (IArguments * args)`
VB: `session_OnCallEstablished (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-29 OnCallEstablished Parameters

Keyword	Type	Description
ServiceNumber	INT	The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_ SERVICE when not applicable or not available.
ServiceID	INT	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.

Keyword	Type	Description
SkillGroupNumber (Optional)	INT	The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupID (Optional)	INT	The Unified ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority (Optional)	SHORT	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
LineType	SHORT	Indicates the type of the teleset line.
EnablementMask	INT	Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call. See Table 3-2.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
CallStatus	SHORT	The current status of the call.
ICMEnterpriseUniqueID (Optional)	STRING	Required only when the call is pre-routed.
TrunkNumber (optional)	INT	The number representing a trunk.
TrunkGroup Number (optional)	INT	The number representing a trunk group.

OnCallFailed

The OnCallFailed event may be generated when a call cannot be completed. With this event the connection status becomes LCS_FAIL. This would most likely happen as a result of a MakeCall or a MakeConsultCall request, but can occur at any other point in the call's lifetime if the call fails on an ACD. In this case, you should perform any required cleanup prior to arrival of an EndCall event.

Note The events (CallConnectionCleared and CallCleared) received after the CallFailed event will 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++: `void OnCallFailed(Arguments& args)`
COM: `void OnCallFailed (IArguments * args)`
VB: `session_OnCallFailed (ByVal args As CtiosCLIENTLib.IArguments`

Parameters

args

Arguments array containing the following fields.

Table 6-30 OnCallFailed Parameters

Keyword	Type	Description
EnablementMask	INT	Contains the bit mask that specifies what buttons can be enabled or disabled when this call is the current call.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
CallStatus	SHORT	The current status of the call.

OnCallHeld

Placing a call on hold at the agent's teleset may generate an OnCallHeld event. With this event the connection status becomes LCS_HELD.

Syntax

C++: `void OnCallHeld(Arguments& args)`
COM: `void OnCallHeld (IArguments * args)`
VB: `session_OnCallHeld (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-31 OnCallHeld Parameters

Keyword	Type	Description
EnablementMask	INT	Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
CallStatus	SHORT	The current status of the call.
ICMEnterpriseUniqueID (Optional)	STRING	Required only when the call is pre-routed.

OnCallOriginated

The initiation of a call from the peripheral may generate an OnCallOriginated event. Only the connection making the call receives this event. With this event the connection status becomes LCS_INITIATE.

Syntax

C++: `void OnCallOriginated(Arguments& args)`
COM: `void OnCallOriginated (IArguments * args)`
VB: `session_OnCallOriginated (ByVal args As CtiosCLIENTLib.IArguments`

Parameters

args

Arguments array containing the following fields.

Table 6-32 OnCallOriginatedParameters

Keyword	Type	Description
ServiceNumber	INT	The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.
ServiceID	INT	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.
SkillGroupNumber (Optional)	INT	The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupID (Optional)	INT	The Unified ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_ GROUP when not applicable or not available.
SkillGroupPriority (Optional)	SHORT	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
LineType	SHORT	Indicates the type of the teleset line.
EnablementMask	INT	Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
CallStatus	SHORT	The current status of the call.

OnCallQueuedEvent

The placing of a call in a queue pending the availability of some resource may generate an OnCallQueuedEvent message to the CTI Client. Clients with Client Events Service may receive this message when an outbound call is queued waiting for a trunk or other resource. Clients with All Events Service may also receive this message when inbound calls are queued.

Syntax

C++: `void OnCallQueuedEvent(Arguments& args)`
COM: `void OnCallQueuedEvent (IArguments * args)`
VB: `session_OnCallQueuedEvent (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-33 OnCallQueuedEvent Parameters

Keyword	Type	Description
Connection DeviceID	INT	The identifier of the connection between the call and the device.
ConnectionDevice IDType	SHORT	Indicates the type of the connection identifier supplied in the ConnectionDeviceID
QueuedDeviceID	STRING	The device identifier of the queuing device.
QueuedDeviceIDType	SHORT	Indicates the type of the device identifier supplied in the QueuedDeviceID.
CallingDeviceID	STRING	The device identifier of the calling device.
CallingDeviceIDType	SHORT	Indicates the type of the device identifier supplied in the CalledDeviceID.
CalledDeviceID	STRING	The device identifier of the called device.
CalledDeviceIDType	SHORT	Indicates the type of the device identifier supplied in the CalledDeviceID.
LastRedirectedDeviceID	STRING	The device identifier of the redirecting device.
LastRedirected DeviceIDType	SHORT	Indicates the type of the device identifier supplied in the LastRedirectDeviceID.
LocalConnection State	SHORT	The state of the local end of the connection.
EventCause	SHORT	Indicates a reason or explanation for the occurrence of the event.
LineHandle	SHORT	Identifies the teleset line being used.
LineType	SHORT	Indicates the type of the teleset line.
ServiceID	INT	The Unified ICM ServiceID of the service that the call is attributed to.

Keyword	Type	Description
ServiceNumber	INT	The service that the call is attributed to, as known to the peripheral.
NumQueued	SHORT	The number of calls in the queue for this service.
NumSkillGroups	SHORT	The number of Skill Group queues that the call has queued to, up to a maximum of 50.

OnCallReachedNetworkEvent

The connection of an outbound call to another network may generate an OnCallReachedNetworkEvent. With some switches outside of the ACD, this may be the last event the outbound connection receives. For these switches, you may not assume that when the called party receives and answers the call that the OnCallDelivered and OnCallEstablished events will be received.

Syntax

C++: `void OnCallReachedNetworkEvent(Arguments& args)`
COM: `void OnCallReachedNetworkEvent (IArguments * args)`
VB: `session_OnCallReachedNetworkEvent (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-34 OnCallReachedNetworkEvent Parameters

Keyword	Type	Description
Connection DeviceID	STRING	The identifier of the connection between the call and the device.
ConnectionDevice IDType	SHORT	Indicates the type of the connection identifier supplied in the ConnectionDeviceID.
TrunkUsedDeviceID	STRING	The device identifier of the selected trunk.
TrunkUsedDeviceID Type	SHORT	Indicates the type of the device identifier supplied in the TrunkUsedDeviceID.
CalledDeviceID	STRING	The device identifier of the called device.
CalledDeviceIDType	SHORT	Indicates the type of the device identifier supplied in the CalledDeviceID.
LocalConnection State	SHORT	The state of the local end of the connection.
EventCause	SHORT	Indicates a reason or explanation for the occurrence of the event.
LineHandle	SHORT	Identifies the teleset line being used.
LineType	SHORT	Indicates the type of the teleset line.

Keyword	Type	Description
TrunkNumber (optional)	INT	The number representing a trunk.
TrunkGroup Number (optional)	INT	The number representing a trunk group.

OnCallRetrieved

Resuming a call previously placed on hold at the agent's teleset may generate an OnCallRetrieved event. With this event the connection status becomes LCS_CONNECT.

Syntax

C++: `void OnCallRetrieved(Arguments& args)`
COM: `void OnCallRetrieved (IArguments * args)`
VB: `session_OnCallRetrieved (ByVal args As CtiosCLIENTLib.IArguments`

Parameters

args

Arguments array containing the following fields.

Table 6-35 OnCallRetrieved Parameters

Keyword	Type	Description
EnablementMask	INT	Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
CallStatus	SHORT	The current status of the call.

OnCallServiceInitiatedEvent

The initiation of telecommunications service ("dial tone") at the agent's teleset may generate an OnCallServiceInitiatedEvent to the CTI Client. However, when the call is made through the software there is no way to detect the equivalent of the phone off hook. Therefore, after a call is made this event is received in sequence along with the OnCallOriginated and OnCallDelivered events. With this event the connection status becomes LCS_INITIATE.

Syntax

C++: `void OnCallServiceInitiatedEvent(Arguments& args)`
COM: `void OnCallServiceInitiatedEvent (IArguments * args)`
VB: `session_OnCallServiceInitiatedEvent (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-36 OnCallServiceInitiatedEvent Parameters

Keyword	Type	Description
ServiceNumber	INT	The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.
ServiceID	INT	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.
SkillGroupNumber (Optional)	INT	The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupID (Optional)	INT	The Unified ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority (Optional)	SHORT	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
LineType	SHORT	Indicates the type of the teleset line.
EnablementMask	INT	Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
CallStatus	SHORT	The current status of the call.

OnCallStartRecordingConf

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

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 6-37 OnCallStartRecordingConf Parameters

OnCallStopRecordingConf

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

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 6-38 OnCallStopRecordingConf Parameters

Keyword	Type	Description
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.

OnCallTransferred

The transfer of a call to another destination may generate an OnCallTransferred event. With this event the two connections at the controller's device end and the status of the connections at the original caller's device and the consulted device are unchanged.

Syntax

C++: `void OnCallTransferred(Arguments& args)`
COM: `void OnCallTransferred (IArguments * args)`
VB: `session_OnCallTransferred (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-39 OnCallTransferred Parameters

Keyword	Type	Description
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where the call activity occurred.
PeripheralType	SHORT	The type of the peripheral.
CallType	SHORT	The general classification of the call type.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
RouterCallKeyDay	INT	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.
RouterCalKeyCallID	INT	The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.
ConnectionCallID	UINT	The Call ID value assigned to this call by the peripheral or the Unified ICM.
ANI (optional)	STRING	The calling line ID of the caller.
DNIS (optional)	STRING	The DNIS provided with the call.
UserToUserInfo (Optional)	STRING	The ISDN user-to-user information element. unspecified, up to 131 bytes.
DialedNumber (Optional)	STRING	The number dialed.
CallerEnteredDigits (Optional)	STRING	The digits entered by the caller in response to IVR prompting.
ServiceNumber (Optional)	INT	The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.
ServiceID (Optional)	INT	The 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.
SkillGroupNumber (Optional)	INT	The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupID (Optional)	INT	The Unified ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority (Optional)	SHORT	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
CallWrapupData (Optional)	STRING	Call-related wrap-up data.
CallVariable1 (Optional)	STRING	Call-related variable data.
…		…
CallVariable10 (Optional)	STRING	Call-related variable data.

Keyword	Type	Description
CallStatus (Optional)	SHORT	The current status of the call.
ECC (optional)	ARGUMENTS	Arguments array that contains all of the Expanded Call Context variables in use; for example: user.ArrayVariable[0] user.ArrayVariable[1] ... user.ArrayVariable[n] user.ScalarVariable
CTIClients (Optional)	ARGUMENTS	Arguments array that contains the information about the number of clients that are using the call object; for example: CTIClient[1] CTIClientSignature CTIClientTimestamp
ICMEnterpriseUniqueID (Optional)	STRING	Required only when the call is pre-routed.

OnClearCallConf

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

Syntax

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

Parameters

args

Arguments array containing the following field.

Table 6-40 OnClearCallConf Parameters

Keyword	Type	Description
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.

OnClearConnectionConf

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

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 6-41 OnClearConnectionConf Parameters

Keyword	Type	Description
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.

OnConferenceCallConf

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

Syntax

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

Parameters

args

Arguments array containing the following field.

Table 6-42 OnConferenceCallConf Parameters

OnConsultationCallConf

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

Syntax

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

Parameters

args

Arguments array containing the following field.

Table 6-43 OnConsulationCallConf Parameters

Keyword	Type	Description
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.

OnControlFailureConf

The OnControlFailureConf event is generated when a request to the peripheral (the ACD) fails.

Syntax

C++:`void OnControlFailureConf(Arguments& args)`
COM: `void OnControlFailureConf (IArguments * args)`
VB: `session_OnControlFailureConf (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-44 OnControlFailureConf Parameters

Keyword	Type	Description
PeripheralID	INT	Peripheral ID.
FailureCode	SHORT	Code ID.
PeripheralError Code	INT	Peripheral-specific error data, if available. Zero otherwise.
AgentID	STRING	Agent ID that represents a specific client.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
MessageType	INT	Contains the CTI OS Command Request ID that failed to execute. The message types that can be included in this parameter are those to used to control Call, Agent State and Supervisor actions. Refer to Appendix A, "CTI OS Keywords and Enumerated Types" for a complete list
ErrorMessage	STRING	String text containing the description of the failure.

OnHoldCallConf

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

Syntax

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

Parameters

args

Arguments array containing the following field.

Table 6-45 OnHoldCallConf Parameters

Keyword	Type	Description
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.

OnMakePredictiveCallConf

Not supported.

OnReconnectCallConf

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

Syntax

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

Parameters

args

Arguments array containing the following field.

Table 6-46 OnMakePredictiveCallConf Parameters

Keyword	Type	Description
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.

OnReleaseCallConf

Not supported.

OnRetrieveCallConf

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

Syntax

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

Parameters

args

Arguments array containing the following field.

Table 6-47 OnReleaseCallConf Parameters

Keyword	Type	Description
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.

OnSendDTMFConf

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

Syntax

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

Parameters

args

Not used; reserved for future use.

OnSetCallDataConf

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

Syntax

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

Parameters

args

Arguments array containing the following field.

Table 6-48 OnReleaseCallConf Parameters

Keyword	Type	Description
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.

OnSnapshotCallConf

The OnSnapshotCallConf event is generated when a SnapshotCall request for a specific call is successful. It will contain all the information known about the specific connection at that point in time.

Syntax

C++: `void OnSnapshotCallConf(Arguments& args)`
COM: `void OnSnapshotCallConf (IArguments * args)`
VB: `session_OnSnapshotCallConf (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-49 OnSnapShotCallConf Parameters

Keyword	Type	Description
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where the call activity occurred.
CallType	SHORT	The general classification of the call type.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
DialedNumber	STRING	The number dialed.
CallerEnteredDigits	STRING	The digits entered by the caller in response to IVR prompting.
CallWrapupData	STRING	Call-related wrap-up data.
CallVariable1 (Optional)	STRING	Call-related variable data.
…		…
CallVariable10 (Optional)	STRING	Call-related variable data.
CustomerPhone Number	STRING	The customer phone number associated with the call.
CustomerAccount Number	STRING	The customer account number associated with the call.
ECC	ARGUMENTS	Arguments array that contains all of the Expanded Call Context variables in use; for example: user.ArrayVariable[0] user.ArrayVariable[1] ... user.ArrayVariable[n] user.ScalarVariable
CTIClients (Optional)	ARGUMENTS	Arguments array that contains the information about the number of clients that are using the call object; for example: CTIClient[1] CTIClientSignature CTIClientTimestamp
RouterCallKeyDay	INT	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.
RouterCallKeyCallID	INT	The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.

Keyword	Type	Description
NumNamedVariables	SHORT	Number of Named variables.
NumNamedArrays	SHORT	Number of Named Arrays.
NumCallDevices	SHORT	Number of devices associated with the call.
CalledDeviceID	STRING	The device identifier of the called device.
ConnectionCallID	UINT	The Call ID value assigned to this call by the peripheral or the Unified ICM.
CallStatus	SHORT	The current status of the call.
The following fields appear *if* they have information in them.		
ANI	STRING	The calling line ID of the caller.
UserToUserInfo	STRING	The ISDN user-to-user information element associated with the call.
DNIS	STRING	The DNIS provided with the call.

If the MinimizeEventArgs registry entry is set to 0, the SnapshotCallConf event contains the following additional fields.

Table 6-50 SnapshotCallConf additional fields

Keyword	Type	Description
ICMEnterpriseUnique ID	STRING	This string is a globally unique key for this contact, which corresponds to the Unified ICM 64 bit key. This parameter can be used to match this contact to a follow-on call event.
CallConnectionCallID *(optional)*	UINT	The CallID value assigned to the call.
CallConnectionDeviceID Type (optional)	SHORT	Indicates the type of the connection identifier supplied in the following CallConnectionDeviceID floating field. This field always immediately follows the corresponding CallConnectionCallID field.
CallConnectionDeviceID (optional)	STRING	The identifier of the call connection. This field always immediately follows the corresponding CallConnectionDeviceIDType field.
CallDeviceConnection State	SHORT	The active state of the call. This field always immediately follows the corresponding CallConnection DeviceID field.
CallDeviceType	SHORT	Indicates the type of the device identifier supplied in the CallDeviceID field.

OnTransferCallConf

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

Syntax

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

Parameters

args

Arguments array containing the following field.

Table 6-51 OnTransferCallConf Parameters

Keyword	Type	Description
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.

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 6-52 OnAgentDeskSettingsConf Parameters

Table 6-53 DeskSettingsMasks Values

Mask Name	Description	Numeric Value
DESK_AVAIL_AFTER_ INCOMING_MASK	Set for automatically consider the agent available after handling an incoming call.	0x00000001
DESK_AVAIL_AFTER_O UTGOING_MASK	Set for automatically consider the agent available after handling an outbound call.	0x00000002

Table 6-53 DeskSettingsMasks Values

Mask Name	Description	Numeric Value
DESK_AUTO_ ANSWER_ENABLED_ MASK	Set when calls to the agent are automatically answered.	0x00000004
DESK_IDLE_REASON_ REQUIRED_MASK	Set when the agent must enter a reason before entering the Idle state.	0x00000008
DESK_LOGOUT_ REASON_REQUIRED_ MASK	Set when the agent must enter a reason before logging out.	0x00000010
DESK_SUPERVISOR_ CALLS_ALLOWED_MA SK	Set when the agent can initiate supervisor assisted calls.	0x00000020
DESK_AGENT_TO_ AGENT_CALLS_ ALLOWED	Set when calls to other agents are allowed.	0x00000040
DESK_OUTBOUND_AC CESS_INTERNATIONA L_MASK	Set when the agent can initiate international calls.	0x00000080
DESK_OUTBOUND_AC CESS_PUBLIC_NET_M ASK	Set when the agent can initiate calls through the public network.	0x00000100
DESK_OUTBOUND_AC CESS_PRIVATE_NET_M ASK	Set when the agent can initiate calls through the private network.	0x00000200
DESK_OUTBOUND_AC CESS_OPERATOR_ASSI STED_MASK	Set when the agent can initiate operator assisted calls.	0x00000400
DESK_OUTBOUND_AC CESS_PBX_MASK	Set when the agent can initiate outbound PBX calls.	0x00000800
DESK_NON_ACD_CAL LS_ALLOWED_MASK	Set when the agent can place or handle non-ACD calls.	0x00001000
DESK_AGENT_CAN_SE LECT_GROUP_MASK	Set when the agent can select which groups they are logged into.	0x00002000

OnAgentInfoEvent

The OnAgentInfoEvent event is generated as a response to a query to the Agent Name Lookup Service and carries the agent's name. This query is generated by the CTI OS server when it is configured to do agent name lookup. The OnAgentInfoEvent event is sent to the client if the server has obtained the information.

Syntax

C++:`void OnAgentInfoEvent(Arguments& args)`
COM:`void OnAgentInfoEvent (IArguments * args)`
VB: `session_OnAgentInfoEvent (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields

Table 6-54 OnAgentInfoEvent Parameters

Keyword	Type	Description
UniqueObjectID	STRING	A unique object ID for the agent object
AgentLastName	STRING	Agent's last name
AgentFirstName	STRING	Agent's first name.

OnAgentStateChange

The OnAgentStateChange event is generated when the agent state at the ACD changes. This may be as a response to a Login, Logout or SetAgentState request.

Syntax

C++:`void OnAgentStateChange(Arguments& args)`
COM: `void OnAgentStateChange (IArguments * args)`
VB: `session_OnAgentStateChange (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-55 OnAgentIStateChange Parameters

Keyword	Type	Description
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where the agent state change occurred.
PeripheralType	SHORT	The type of the peripheral.
AgentState	SHORT	One of the values in Table 6-56 representing the current overall state of the associated agent.
SkillGroupNumber	INT	The number of the agent SkillGroup affected by the state change, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupID	INT	The Unified ICM SkillGroupID of the agent SkillGroup affected by the state change. May contain the special value NULL_SKILL_ GROUP when not applicable or not available.
StateDuration	INT	The number of seconds since the agent entered this state (typically 0).

Keyword	Type	Description
SkillGroupPriority	SHORT	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
EventReasonCode	SHORT	A peripheral-specific code indicating the reason for the state change.
SkillGroupState	SHORT	Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.
AgentID	STRING	The agent's ACD login ID.
AgentExtension	STRING	The agent's ACD teleset extension.
CTIClientSignature (Optional)	STRING	The Client Signature of the CTI Client that is associated with this agent.
Enablement Mask		Contains the bit-mask that specifies what buttons can be enabled or disabled when the agent is on this state.
UniqueObjectID	STRING	A unique object ID for the agent object.
AgentInstrument	STRING	The agent's ACD instrument number.

The following table provides the AgentState values.

Table 6-56 AgentState Values

enum Value	Description	Numeric Value
eLogin	The agent has logged on to the ACD. It does not necessarily indicate that the agent is ready to accept calls.	0
eLogout	The agent has logged out of the ACD and cannot accept any additional calls.	1
eNotReady	The agent is unavailable for any call work.	2
eAvailable	The agent is ready to accept a call.	3
eTalking	The agent is currently talking on a call (inbound, outbound, or inside).	4
eWorkNotReady	The agent is performing after call work, but will not be ready to receive a call when completed.	5
eWorkReady	The agent is performing after call work, and will be ready to receive a call when completed.	6
eBusyOther	The agent is busy performing a task associated with another active SkillGroup.	7
eReserved	The agent is reserved for a call that will arrive at the ACD shortly.	8
eUnknown	The agent state is currently unknown.	9
eHold	The agent currently has all calls on hold.	10

Note Not all switches support all the states listed in Table 6-56, and no assumptions should be made about which states are supported on a particular switch without verification.

OnAgentStatistics

The OnAgentStatistics event is fired to the client to indicate that a request to enable agent statistics (via the EnableAgentStatistics method) was received by the CTIServer. The arrival of events event is determined by the configuration on the server.

The table under Parameters details all the agent statistics that could be received. To optimize bandwidth, the default configuration on the server is set to minimize the agent statistics sent. Only the statistics that the Agent Statistics grid is configured for are sent to the client. Refer to the *CTI OS System Manager's Guide for Cisco Unified ICM/Contact Center Enterprise & Hosted* for details on how to configure the agent statistics grid and minimize agent statistics.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-57 OnAgentStatistics Parameters

Keyword	Description	Type
PeripheralID	The Unified ICM PeripheralID of the ACD where the agent is located.	INT
AgentExtension (required)	The agent's ACD teleset extension.	STRING
AgentID (required)	The agent's ACD login ID.	STRING
AgentInstrument (required)	The agent's ACD instrument number.	STRING

The OnAgentStatistics event will contain 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 from another user (agent) is received.

Syntax

C++:`void OnChatMessage(Arguments& args)`
COM:`void OnChatMessage (IArguments * args)`
VB: `session_OnChatMessage (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-58 OnChatMessage Parameters

Keyword	Type	Description
Distribution	STRING	Currently the only supported value is "agent".
AgentID	STRING	The AgentID of the message target.
Target	STRING	The AgentID of the message target.
Message	STRING	The text message provided by the sender.
Source	STRING	The AgentID of the message sender.

OnControlFailureConf

The OnControlFailureConf event is generated when the previously issued request, identified by the InvokeID field failed. It is sent in place of the corresponding confirmation message for that request.

Syntax

C++:`void OnControlFailureConf(Arguments& args)`
COM:`void OnControlFailureConf (IArguments * args)`
VB: `session_OnControlFailureConf (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-59 OnControlFailureConf Parameters

Keyword	Type	Description
InvokeID	INT	InvokeID of the request that failed
FailureCode	SHORT	A value specifying the reason that the request failed. See Table 6-60 for a list of the Control Failure Codes.
PeripheralError Code	INT	Peripheral-specific error data, if available. Zero otherwise.
AgentID	STRING	Agent ID that represents a specific client.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
MessageType	INT	Contains the CTI OS Command Request ID that failed to execute. The message types that can be included in this parameter are those to used to control Call, Agent State and Supervisor actions. Refer to Appendix A, "CTI OS Keywords and Enumerated Types" for a complete list.
ErrorMessage	STRING	String text containing the description of the failure.

Table 6-60 Control Failure Codes

Status Code	Description	Value
E_CTI_NO_ERROR	No error occurred.	0
E_CTI_INVALID_ VERSION	The CTI Server does not support the protocol version number requested by the CTI client.	1
E_CTI_INVALID_ MESSAGE_ TYPE	A message with an invalid message type field was received.	2
E_CTI_INVALID_ FIELD_TAG	A message with an invalid floating field tag was received.	3
E_CTI_SESSION_ NOT_OPEN	No session is currently open on the connection.	4

Table 6-60 Control Failure Codes

Status Code	Description	Value
E_CTI_SESSION_ ALREADY_ OPEN	A session is already open on the connection.	5
E_CTI_REQUIRED_ DATA_ MISSING	The request did not include one or more floating items that are requir ed.	6
E_CTI_INVALID_ PERIPHERAL_ID	A message with an invalid PeripheralID value was received.	7
E_CTI_INVALID_ AGENT_ DATA	The provided agent data items are invalid.	8
E_CTI_AGENT_NOT_ LOGGED_ON	The indicated agent is not currently logged on.	9
E_CTI_DEVICE_IN_ USE	The indicated agent teleset is already associated with a different CTI client.	10
E_CTI_NEW_ SESSION_ OPENED	This session is being terminated due to a new session open request from the client.	11
E_CTI_FUNCTION_ NOT_ AVAILABLE	A request message was received for a function or service that was not granted to the client.	12
E_CTI_INVALID_ CALLID	A request message was received with an invalid CallID value.	13
E_CTI_PROTECTED_ VARIABLE	The CTI client may not update the requested variable.	14
E_CTI_CTI_SERVER_ OFFLINE	The CTI Server is not able to function normally. The CTI client should close the session upon receipt of this error.	15
E_CTI_TIMEOUT	The CTI Server failed to respond to a request message within the time-out period, or no messages have been received from the CTI client within the IdleTimeout period.	16
E_CTI_UNSPECIFIED_ FAILURE	An unspecified error occurred.	17
E_CTI_INVALID_ TIMEOUT	The IdleTimeout field contains a value that is less than 20 seconds (4 times the minimum heartbeat interval of 5 seconds).	18
E_CTI_INVALID_ SERVICE_MASK	The ServicesRequested field has unused bits set. All unused bit positions must be zero.	19
E_CTI_INVALID_ CALL_MSG_MASK	The CallMsgMask field has unused bits set. All unused bit positions must be zero.	20
E_CTI_INVALID_ AGENT_ STATE_ MASK	The AgentStateMask field has unused bits set. All unused bit positions must be zero.	21
E_CTI_INVALID_ RESERVED_ FIELD	A Reserved field has a non-zero value.	22
E_CTI_INVALID_ FIELD_ LENGTH	A floating field exceeds the allowable length for that field type.	23
E_CTI_INVALID_ DIGITS	A STRING field contains characters that are not digits (“0” through “9”).	24

Table 6-60 Control Failure Codes

Status Code	Description	Value
E_CTI_BAD_ MESSAGE_ FORMAT	The message is improperly constructed. This may be caused by omitted or incorrectly sized fixed message fields.	25
E_CTI_INVALID_ TAG_FOR_MSG_ TYPE	A floating field tag is present that specifies a field that does not belong in this message type.	26
E_CTI_INVALID_ DEVICE_ID_ TYPE	A DeviceIDType field contains an invalid. value.	27
E_CTI_INVALID_ LCL_CONN_ STATE	A LocalConnectionState field contains an invalid value.	28
E_CTI_INVALID_ EVENT_ CAUSE	An EventCause field contains an invalid value.	29
E_CTI_INVALID_ NUM_ PARTIES	The NumParties field contains a value that exceeds the maximum (16).	30
E_CTI_INVALID_ SYS_ EVENT_ID	The SystemEventID field contains an invalid value.	31
E_CTI_ INCONSISTENT_ AGENT_DATA	The provided agent extension, agent id, and/or agent instrument values are inconsistent with each other.	32
E_CTI_INVALID_ CONNECTION_ID_ TYPE	A ConnectionDeviceIDType field contains an invalid value.	33
E_CTI_INVALID_ CALL_TYPE	The CallType field contains an invalid value.	34
E_CTI_NOT_CALL_ PARTY	A CallDataUpdate or Release Call request specified a call that the client is not a party to.	35
E_CTI_INVALID_ PASSWORD	The ClientID and Client Password provided in an OPEN_REQ message is incorrect.	36
E_CTI_CLIENT_ DISCONNECTED	The client TCP/IP connection was disconnected without a CLOSE_REQ.	37
E_CTI_INVALID_ OBJECT_ STATE	An invalid object state value was provided.	38
E_CTI_INVALID_ NUM_ SKILL_GROUPS	An invalid NumSkillGroups value was provided.	39
E_CTI_INVALID_ NUM_LINES	An invalid NumLines value was provided.	40
E_CTI_INVALID_ LINE_TYPE	An invalid LineType value was provided.	41
E_CTI_INVALID_ ALLOCATION_STATE	An invalid AllocationState value was provided.	42
E_CTI_INVALID_ ANSWERING_ MACHINE	An invalid AnsweringMachine value was provided.	43

Table 6-60 Control Failure Codes

Status Code	Description	Value
E_CTI_INVALID_ CALL_MANNER_ TYPE	An invalid CallMannerType value was provided.	44
E_CTI_INVALID_ CALL_PLACEMENT_ TYPE	An invalid CallPlacementType value was provided.	45
E_CTI_INVALID_ CONSULT_ TYPE	An invalid ConsultType value was provided.	46
E_CTI_INVALID_ FACILITY_ TYPE	An invalid FacilityType value was provided.	47
E_CTI_INVALID_ MSG_TYPE_ FOR_ VERSION	The provided MessageType is invalid for the opened protocol version.	48
E_CTI_INVALID_ TAG_FOR_ VERSION	A floating field tag value is invalid for the opened protocol version.	49
E_CTI_INVALID_ AGENT_WORK_ MODE	An invalid AgentWorkMode value was provided.	50
E_CTI_INVALID_ CALL_OPTION	An invalid call option value was provided.	51
E_CTI_INVALID_ DESTINATION_ COUNTRY	An invalid destination country value was provided.	52
E_CTI_INVALID_ ANSWER_DETECT_ MODE	An invalid answer detect mode value was provided.	53
E_CTI_MUTUALLY_ EXCLUS_DEVICEID_ TYPES	A peripheral monitor request may not specify both a call and a device.	54
E_CTI_INVALID_ MONITORID	An invalid monitorID value was provided.	55
E_CTI_SESSION_ MONITOR_ ALREADY_EXISTS	A requested session monitor was already created.	56
E_CTI_SESSION_ MONITOR_IS_ CLIENTS	A client may not monitor its own session.	57
E_CTI_INVALID_ CALL_CONTROL_ MASK	An invalid call control mask value was provided.	58
E_CTI_INVALID_ FEATURE_MASK	An invalid feature mask value was provided.	59

Table 6-60 Control Failure Codes

Status Code	Description	Value
E_CTI_INVALID_ TRANSFER_ CONFERENCE_ SETUP_MASK	An invalid transfer conference setup mask value was provided.	60
E_CTI_INVALID_ ARRAY_INDEX	An invalid named array index value was provided.	61
E_CTI_INVALID_ CHARACTER	An invalid character value was provided.	62
E_CTI_CLIENT_NOT_ FOUND	There is no open session with a matching ClientID.	63
E_CTI_SUPERVISOR_ NOT_FOUND	The agent's supervisor is unknown or does not have an open CTI session.	64
E_CTI_TEAM_NOT_ FOUND	The agent is not a member of an agent team.	65
E_CTI_NO_CALL_ ACTIVE	The specified agent does not have an active call.	66
E_CTI_NAMED_ VARIABLE_NOT_ CONFIGURED	The specified named variable is not configured in the Unified ICM database.	67
E_CTI_NAMED_ ARRAY_NOT_ CONFIGURED	The specified named array is not configured in the Unified ICM database.	68
E_CTI_INVALID_ CALL_VARIABLE_ MASK	The specified call variable mask in not valid.	69
E_CTI_ELEMENT_ NOT_FOUND	An internal error occurred manipulating a named variable or named array element.	70
E_CTI_INVALID_ DISTRIBUTION_TYPE	The specified distribution type is invalid.	71
E_CTI_INVALID_ SKILL_GROUP	The specified skill group is invalid.	72
E_CTI_TOO_MUCH_ DATA	The total combined size of named variables and named arrays may not exceed the limit of 2000 bytes.	73
E_CTI_VALUE_TOO_L ONG	The value of the specified named variable or named array element exceeds the maximum permissible length.	74
E_CTI_SCALAR_ FUNCTION_ON_ ARRAY	A NamedArray was specified with a NamedVariable tag.	75
E_CTI_ARRAY_ FUNCTION_ON_ SCALAR	A NamedVariable was specified with a NamedArray tag.	76

Table 6-60 Control Failure Codes

Status Code	Description	Value
E_CTI_INVALID_ NUM_NAMED_ VARIABLES	The value in the NumNamedVariables field is different than the number of NamedVariable floating fields in the message.	77
E_CTI_INVALID_ NUM_NAMED_ ARRAYS	The value in the NumNamedArrays field is different than the number of NamedArray floating fields in the message.	78

OnEmergencyCall

The OnEmergencyCall event indicates that a CTI client (with Supervisory capabilities) is handling the indicated call as an emergency call. This event only applies to ACDs with Supervisor capabilities.

Syntax

C++: `void OnEmergencyCall(Arguments& args)`
COM:`void OnEmergencyCall (IArguments * args)`
VB: `session_OnEmergencyCall (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-61 OnEmergencyCall Parameters

Keyword	Type	Description
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where the call is located.
Connection CallID	INT	The Call ID value assigned to the call by the peripheral or the Unified ICM.
ConnectionDevice IDType	SHORT	Indicates the type of the connection identifier supplied in the ConnectionDeviceID floating field.
SessionID	INT	The CTI client SessionID of the CTI client making the notification.
Connection DeviceID	INT	The identifier of the connection between the call and the agent's device.
ClientID (required)	STRING	The ClientID of the client making the notification.
ClientAddress (Required)	STRING	The IP address of the client making the notification.
AgentExtension (Required)	STRING	The Agent's teleset extension.
AgentID (required)	STRING	The Agent's ACD login ID.
AgentInstrument (required)	STRING	The Agent's ACD instrument number.

Remarks

Supported for use with Unified CCE only.

OnLogoutFailed

The OnLogoutFailed ia always generated before (or along with) an OnControlFailureConf event and is identical to it but is generated only when a Logout request fails.

Syntax

C++:`void OnLogoutFailed (Arguments& args)`
COM:`void OnLogoutFailed (IArguments * args)`
VB: `session_OnLogoutFailed (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-62 OnLogoutFailed Parameters

Keyword	Type	Description
InvokeID	INT	InvokeID of the request that failed
FailureCode	SHORT	A value specifying the reason that the request failed. See Table 6-60 for a list of the Control Failure Codes.
Peripheral ErrorCode	INT	Peripheral-specific error data, if available. Zero otherwise.

OnMakeCallConf

The OnMakeCallConf event confirms the successful completion of the MakeCall request. It conveys the information detailed in the table under Parameters.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-63 OnMakeCallConf Parameters

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 6-64 OnNewAgentTeamMember Parameters

Keyword	Type	Description
PeripheralID	STRING	The Unified ICM PeripheralID of the agent's ACD .
UniqueObjectID	STRING	Unique object ID of the agent object for this agent.
AgentState	SHORT	One of the values in Table 6-56 representing the current state of the associated agent.
NumSkillGroups	INT	The number of skill groups that the agent is currently associated with, up to a maximum of 99.
AgentID	STRING	Agent's ACD login.
AgentExtension	STRING	Agent's ACD teleset extension.
AgentInstrument	STRING	Agent's ACD instrument number.

AgentLastName	STRING	Agent's last name.
AgentFirstName	STRING	Agent's first name.
AgentName	STRING	Agent's full name.
AgentAvailability Status	SHORT	The current status of the agent's availability to take a call.
EnablementMask	INT	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.
SupervisorID	STRING	The ID of the agent's supervisor.
AgentFlags	INT	Used to describe the agent carried in this event. The possible values for this field as well as their meanings are as follows. • TeamMemberFlags.AGENT_FLAG_REGULAR_AGENT - Value is 0. The agent is a regular agent • TeamMemberFlags.AGENT_FLAG_PRIMARY_SUPERVISOR - Value is 1. The agent is a primary supervisor • TeamMemberFlags.AGENT_FLAG_TEMPORARY_AGENT - Value is 2. The agent is a temporary agent • TeamMemberFlags.AGENT_FLAG_SUPERVISOR - Value is 4. The agent is a supervisor
Skillgroup[1}	ARGUMENTS	Arguments array containing information about the agent's first skillgroup. The array contains the following arguments. • SkillGroupNumber • SkillGroupID • StateDuration • SkillGroupPriority
Skillgroup[n]	ARGUMENTS	Arguments array containing information about the agent's nth skillgroup.
ConfigOperation	USHORT	used to describe a change to the team. The possible values for this field as well as their meanings are as follows. • TeamMemberFlags.CONFIG_OPERATION_ADD_AGENT - Value is 1 - The agent belongs to the team • TeamMemberFlags.CONFIG_OPERATION_REMOVE_AGENT - Value is 2 - The agent no longer belongs to the team

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 has been 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 6-65 OnPostLogout Parameters

Keyword	Type	Description
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where the agent state change occurred.
PeripheralType	SHORT	The type of the peripheral.
AgentState	SHORT	One of the values in Table 6-56 representing the current overall state of the associated agent.
SkillGroupNumber	INT	The number of the agent SkillGroup affected by the state change, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupID	INT	The Unified ICM SkillGroupID of the agent SkillGroup affected by the state change. May contain the special value NULL_SKILL_ GROUP when not applicable or not available.
StateDuration	INT	The number of seconds since the agent entered this state (typically 0).
SkillGroupPriority	SHORT	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
EventReasonCode	SHORT	A peripheral-specific code indicating the reason for the state change.
SkillGroupState	SHORT	Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.
AgentID	STRING	The agent’s ACD login ID.
AgentExtension	STRING	The agent’s ACD teleset extension.
CTIClientSignature (Optional)	STRING	The Client Signature of the CTI Client that is associated with this agent.

Keyword	Type	Description
EnablementMask	INT	Contains the bit-mask that specifies what buttons can be enabled or disabled when the agent is on this state.
UniqueObjectID	STRING	A unique object ID for the agent object.
AgentInstrument	STRING	The agent's ACD instrument number.

Remarks

When PG failover occurs, it is possible that the client application will receive an OnPostLogout event with an EventReasonCode of CTIOS_IPCC_FORCED_LOGOUT_REASON_CODE. For example, this may 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 CTI OS Server.

OnPreLogout

The OnPreLogout event just before the agent is logged out. It allows for any cleanup or logic that needs to be done before logout is completed.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-66 OnPreLogout Parameters

Keyword	Type	Description
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where the agent state change occurred.
PeripheralType	SHORT	The type of the peripheral.
AgentState	SHORT	One of the values in Table 6-56 representing the current overall state of the associated agent.
SkillGroupNumber	INT	The number of the agent SkillGroup affected by the state change, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.

Keyword	Type	Description
SkillGroupID	INT	The Unified ICM SkillGroupID of the agent SkillGroup affected by the state change. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
StateDuration	INT	The number of seconds since the agent entered this state (typically 0).
SkillGroupPriority	SHORT	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
EventReasonCode	SHORT	A peripheral-specific code indicating the reason for the state change.
SkillGroupState	SHORT	Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.
AgentID	STRING	The agent's ACD login ID.
AgentExtension	STRING	The agent's ACD teleset extension.
CTIClientSignature (Optional)	STRING	The Client Signature of the CTI Client that is associated with this agent.
Enablement Mask		Contains the bit-mask that specifies what buttons can be enabled or disabled when the agent is on this state.
UniqueObjectID	STRING	A unique object ID for the agent object.
AgentInstrument	STRING	The agent's ACD instrument number.

OnQueryAgentStateConf

The OnQueryAgentStateConf event is generated and returned by the server at login as a response to the QueryAgentState() request. A user cannot issue this request.

Syntax

C++:`void OnQueryAgentStateConf(Arguments& args)`
COM:`void OnQueryAgentStateConf (IArguments * args)`
VB: `session_OnQueryAgentStateConf (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-67 OnQueryAgentStateConf Parameters

Keyword	Type	Description
AgentID	STRING	Agent's ACD login.
AgentExtension	STRING	Agent's ACD teleset extension.
AgentInstrument	STRING	Agent's ACD instrument number.

AgentState	SHORT	One of the values in Table 6-56 representing the current state of the associated agent.
NumSkillGroups	INT	The number of skill groups that the agent is currently associated with, up to a maximum of 20.
SkillGroup[j]	ARGUMENTS	Argument array that contains Skill Group information for the j-*th* element less than NumSkillGroups. The message will contain up to NumSkillGroups elements of this type.
MRDID	INT	Media Routing Domain ID as configured in Unified ICM and the ARM client.
NumTasks	INT	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.
AgentMode	SHORT	The mode that the agent will be in when the login completes. ROUTABLE = 0, NOT ROUTABLE = 1
MaxTaskLimit	INT	The maximum number of tasks that the agent can be simultaneously working on.
ICMAgentID	INT	The Unified ICM Skill Target ID, a unique agent identifier for Unified ICM.
Agent Availability Status	INT	An agent is *available* to work on a task in this Media Routing Domain if the agent meets all of these conditions: • The agent is routable for this Media Routing Domain • The agent is not in Not Ready state for skill groups in other Media Routing Domain • The agent is *temp routable*, meaning that the agent is not in Reserved, Active, Work-Ready, or Work-Not Ready state on a non-interruptible task in another Media Routing Domain. • The agent has not reached the maximum task limit for this Media Routing Domain An available agent is eligible to be assigned a task. Who can assign a task to the agent is determined by whether or not the agent is Routable. An agent is *ICMAvailable* in MRD X if he is available in X and Routable with respect to X. An agent is *ApplicationAvailable* in MRD X if he is available in X and not Routable with respect to X. Otherwise an agent is NotAvailable in MRD X. NOT AVAILABLE = 0, ICM AVAILABLE = 1, APPLICATION AVAILABLE=2

Each SkillGroup*[j]* field in the message contains the following information.

Table 6-68 SkillGroup Parameters

Keyword	Type	Description
SkillGroupNumber	INT	The number of an agent SkillGroup queue that the call has been added to, as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupID	INT	The Unified ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or available.
SkillGroupPriority	SHORT	The priority of the skill group, or 0 when the skill group priority is not applicable or not available.
SkillGroupState	SHORT	One of the values representing the current state associated agent with respect to the skill group.

OnSetAgentModeEvent

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

Syntax

C++:`void OnSetAgentModeEvent (Arguments& args)`
COM: `void OnSetAgentModeEvent (IArguments * args)`
VB: `Session_OnSetAgentModeEvent (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-69 OnSetAgentModeEven Parameters

Keyword	Type	Description
PeripheralID	STRING	ID of the Unified ICM Peripheral ACD associated with the agent.
AgentID	STRING	The agent’s ID.
UniqueObject ID	STRING	The new unique object ID for the agent object.
ClientAgent TemporaryID	STRING	Temporary ID used before server passes the new unique object ID.

CIL ConnectionID	STRING	ID of the client's connection on the server.
StatusSystem	ARGUMENTS	Arguments array containing the following elements: • StatusCTIServer • StatusCtiServerDriver • StatusCentralController • StatusPeripherals (Arguments array with a peripheral id for each key and a boolean true/false value indicating if that peripheral is online)

OnSetAgentStateConf

The OnSetAgentStateConf confirmation message is fired to the client to indicate that the SetAgentState request was received by the CTI Server. This confirmation message does not indicate that the agent has changed to the desired state; rather, the programmer should expect one or more OnAgentStateChange events to indicate the change of state.

Syntax

```
C++:int OnSetAgentStateConf (Arguments & args);
COM: HRESULT OnSetAgentStateConf ([out] IArguments * args);
VB: Session_ OnSetAgentStateConf (ByVal args as CTIOSCLIENTLIB.IArguments)
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 6-70 OnStartMonitoringAgent Parameters

Keyword	Type	Description
UniqueObjectID	STRING	Unique object ID for the supervisor object.
AgentReference	STRING	String containing the Agent ID for the agent to be monitored.
SupervisorID	STRING	String containing the supervisor's AgentID
SupervisorKey	STRING	Supervisor's unique object ID.
BargedInCallID	STRING	If the supervisor has barged in on the agent's call, the unique object ID of that call.
Supervisor AgentState	STRING	The supervisor's agent state.

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 6-71 OnStopMonitoringAgent Parameters

Keyword	Type	Description
UniqueObjectID	STRING	Unique object ID for the supervisor object.
AgentReference	STRING	String containing the Agent ID for the agent to be monitored.
SupervisorID	STRING	String containing the supervisor's AgentID
SupervisorKey	STRING	Supervisor's unique object ID.

BargedInCallID	STRING	If the supervisor has barged in on the agent's call, the unique object ID of that call.
Supervisor AgentState	STRING	The supervisor's agent state.

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. The update frequency of OnSkillGroupStatisticsUpdated can be configured on the CTI OS server (see the *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 6-72 OnSkillGroupStatisticsUpdated Parameters

Keyword	Type	Description
PeripheralID	INT	The Unified ICM PeripheralID of the ACD on which the agent resides.
SkillGroupNumber	INT	The number of the agent skill group as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not available.
SkillGroupID	INT	The Unified ICM SkillGroupID of the skill group. May contain the special value NULL_SKILL_GROUP when not available.

The statistics event will also contain all the statistics fields listed in Table 11-2 in a nested arguments array named STATISTICS.

OnSkillInfoEvent

Provides information about a particular skill group. This event will be sent to any client that has enabled skill group statistics.

```
C++: void OnSkillInfoEvent(Arguments& args)
COM: void OnSkillInfoEvent(IArguments * args)
VB:skillgroup_OnSkillInfoEvent(ByVal args As CtiosCLIENTLib.IArguments)
```

Parameters

args

Arguments array containing the following fields.

Table 6-73 OnSkillInfoEVent Parameters

Keyword	Type	Description
SkillGroupNumber	INT	Skill group number
SkillGroupName	STRING	Skill group name associated with the skill group number above

IButtonEnablementEvents

This interface allows a client application to receive events that indicate what buttons can be enabled on the user interface, given the current agent and current call states.

OnButtonEnablementChange

The OnButtonEnablementChange event is received by a client in agent mode whenever CIL receives an agent or call event that carries the EnablementMask field in its parameters. This event allows the client application to enable or disable elements on the user interface. The fields in the event are the same as in OnButtonEnablementChange.

```
C++:void OnButtonEnablementChange (Arguments& args)
COM:void OnButtonEnablementChange (IArguments * args)
VB: session_ OnButtonEnablementChange (ByVal args As CtiosCLIENTLib.IArguments)
```

Parameters

args

Arguments array containing the following fields.

Table 6-74 OnButtonEnablementChange Parameters

Note Table 6-75 represents the C++/COM/VB enumerations. Enumerations for Java can be found in the description of CtiOs_Enums.ButtonEnablement in the Javadoc. It is strongly recommended that bits be referenced by the enumeration rather than the actual number in the bit mask.

Table 6-75 Table of Enablement Bits

Button	Bit Mask
DISABLE_ALL	0x00400000
ENABLE_ANSWER	0X00000001
ENABLE_RELEASE	0X00000002
ENABLE_HOLD	0X00000004
ENABLE_RETRIEVE	0X00000008
ENABLE_MAKECALL	0X00000010
ENABLE_TRANSFER_INIT	0X00000020
ENABLE_TRANSFER_COMPLETE	0X00000040
ENABLE_SINGLE_STEP_TRANSFER	0X00000080
ENABLE_CONFERENCE_INIT	0X00000100
ENABLE_CONFERENCE_COMPLETE	0X00000200
ENABLE_SINGLE_STEP_ CONFERENCE	0X00000400
ENABLE_ALTERNATE	0X00000800
ENABLE_RECONNECT	0X00001000
ENABLE_WRAPUP	0X00002000
ENABLE_INSIDE_MAKECALL	0X00004000
ENABLE_OUTSIDE_MAKECALL	0X00008000
ENABLE_SUPERVISOR_ASSIST	0X00010000
ENABLE_EMERGENCY_CALL	0X00020000
ENABLE_BAD_LINE_CALL	0X00040000
ENABLE_STATISTICS	0X00080000
ENABLE_CHAT	0X00100000

Table 6-75 Table of Enablement Bits

ENABLE_RECORD	0X00200000
ENABLE_LOGIN	0X01000000
ENABLE_LOGOUT	0X02000000
ENABLE_LOGOUT_WITH_REASON	0x04000000
ENABLE_READY	0X08000000
ENABLE_NOTREADY	0X10000000
ENABLE_NOTREADY_WITH_ REASON	0X20000000
ENABLE_WORKREADY	0X40000000
ENABLE_WORKNOTREADY	0x80000000
DISABLE_READY	0xF7FFFFFF
DISABLE_NOTREADY	0xCFFFFFFF
DISABLE_WORKREADY	0xBFFFFFFF
Supervisor Button Enablement Masks	
ENABLE_SET_AGENT_LOGOUT	0x00000001
ENABLE_SET_AGENT_READY	0x00000002
ENABLE_SILENTMONITOR	0x00000004
ENABLE_BARGE_IN	0x00000004
ENABLE_INTERCEPT	0x00000008
ENABLE_CLEAR	0x00000010
ENABLE_START_SILENTMONITOR	0x00000020
ENABLE_STOP_SILENTMONITOR	0x00000040
DISABLE_SET_AGENT_LOGOUT	0xFFFFFFFE
DISABLE_SET_AGENT_READY	0xFFFFFFFD
DISABLE_SILENTMONITOR	0xFFFFFFFB
DISABLE_BARGE_IN	0xFFFFFFFB
DISABLE_INTERCEPT	0xFFFFFFF7
DISABLE_CLEAR	0xFFFFFFEF
DISABLE_START_SILENTMONITOR	0xFFFFFFDF
DISABLE_STOP_SILENTMONITOR	0xFFFFFFBF
DISABLE_SUPERVISE_CALL	DISABLE_BARGE_IN & DISABLE_INTERCEPT & DISABLE_CLEAR & DISABLE_SILENTMONIT OR & DISABLE_START _SILENTMONITOR & DISABLE_STOP_SILENT MONITOR

Table 6-75 Table of Enablement Bits

DISABLE_SET_AGENT_STATE	DISABLE_SET_AGENT_ LOGOUT, DISABLE_SET_ AGENT_READY
DISABLE_ALL_AGENT_SELECT	DISABLE_BARGE_IN & DISABLE_INTERCEPT & DISABLE_CLEAR & DISABLE_SILENTMONIT OR & DISABLE_START _SILENTMONITOR & DISABLE_STOP_SILENT MONITOR

OnSupervisorButtonChange

The OnSupervisorButtonChange is received by a client in agent mode working as supervisor whenever CIL receives a Monitored Agent, Monitored call event that carries the SupervisorBtnEnablementMask field in its parameters. This event allows the client application to enable or disable elements on the user interface. The fields in the event are the same as in OnButtonEnablementChange

C++:`void OnSupervisorButtonChange (Arguments& args)`
COM:`void OnSupervisorButtonChange (IArguments * args)`
VB: `session_ OnSupervisorButtonChange (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-76 OnSupervisorButtonChange Parameters

Keyword	Type	Description
SupervisorBtn EnablementMask	INT	Contains the bit-mask that specifies what buttons can be enabled or disabled when this call is the current call. See Table 6-75.

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 (see Chapter 9, "Agent Object"). For details on the event parameters please see the IAgentEvents interface.

The most common event being handled is the OnMonitoredAgentStateChange event, which informs a supervisor of agent state changes of agents in the supervisor's team. All the parameters are the same as for regular OnAgentStateChange events, except for an additional keyword called CTIOS_MONITORED, which indicates that this event is for a monitored agent.

List of Monitored Agent events:

```
OnMonitoredAgentStateChange([in] IArguments *pIArguments);
OnMonitoredAgentInfoEvent([in] IArguments *pIArguments);
```

IMonitoredCallEvents Interface

Note The events in this section are supported for use with 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 (see Chapter 9, "Agent Object"), the supervisor will start receiving MonitoredCallEvents for this "currently" monitored agent. Monitored call events will be received until the supervisor sends a StopMonitoringAgent request for this agent.

The IMonitoredCallEvents interface includes OnMonitoredCallBegin, OnMonitoredCallEnd, and OnMonitoredCallDataUpdate as well as other call events (see list below). These events are described in detail for the ICallEventsInterface. The only difference is that the arguments array contains an additional keyword call CTIOS_MONITORED, indicating that this event is for a monitored call.

List of Monitored Call events:

```
OnMonitoredCallBegin([in] IArguments *pIArguments);
OnMonitoredCallEnd([in] IArguments *pIArguments);
OnMonitoredCallDataUpdate([in] IArguments *pIArguments);
OnMonitoredCallDelivered([in] IArguments *pIArguments);
OnMonitoredCallEstablished([in] IArguments *pIArguments);
OnMonitoredCallHeld([in] IArguments *pIArguments);
OnMonitoredCallRetrieved([in] IArguments *pIArguments);
OnMonitoredCallCleared([in] IArguments *pIArguments);
OnMonitoredCallConnectionCleared([in] IArguments *pIArguments);
MonitoredCallReachedNetworkEvent([in] IArguments *pIArguments);
OnMonitoredCallOriginated([in] IArguments *pIArguments);
OnMonitoredCallFailed([in] IArguments *pIArguments);
OnMonitoredCallTransferred([in] IArguments *pIArguments);
OnMonitoredCallConferenced([in] IArguments *pIArguments);
OnMonitoredCallDiverted([in] IArguments *pIArguments);
OnMonitoredTranslationRoute([in] IArguments *pIArguments);
OnMonitoredCallAgentPrecallEvent([in] IArguments *pIArguments);
OnMonitoredCallAgentPrecallAbortEvent([in] IArguments *pIArguments);
MonitoredCallServiceInitiatedEvent([in] IArguments *pIArguments);
```

MonitoredCallQueuedEvent([in] IArguments *pIArguments);

MonitoredCallDequeuedEvent([in] IArguments *pIArguments);

ISilentMonitorEvents

The silent monitor manager object fires events on the ISilentMonitorEvents interface. The following events are published to subscribers of the ISilentMonitorEvents interface.

Note The events in this section are supported for use with Unified CCE only.

Note The following events only apply to CTI OS based silent monitor unless noted otherwise.

OnCallRTPStarted

The OnCallRTPStarted event indicates that an RTP media stream has been started. This event accompanies the call object in an Unified CCE environment.

Syntax

C++: `void OnCallRTPStarted(Arguments& args)`
COM: `void OnCallRTPStarted (IArguments * args)`
VB: `session_OnCallRTPStarted (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-77 OnCallRTPStarted Parameters

Keyword	Type	Description
MonitorID	UINT	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).
PeripheralID	UINT	The Unified ICM PeripheralID of the ACD where the device is located.
ClientPort	UINT	The TCP/IP port number of the CTI Client connection
Direction	USHORT	The direction of the event. One of the following values: 0: Input; 1: Output; 2: Bi-directional.

RTPType	USHORT	The type of the event. One of the following values:
		0: Audio;
		1: Video;
		2: Data.

BitRate	UINT	The media bit rate, used for g.723 payload only
EchoCancellation	USHORT	on/off
PacketSize	UINT	In milliseconds
PayloadType	USHORT	The audio codec type
ConnectionDevice IDType	USHORT	Indicates the type of the connection identifier supplied in the ConnectionDeviceID floating field
ConnectionCallID	UINT	The Call ID value assigned to this call by the peripheral or Unified ICM.
Connection DeviceID	STRING	The identifier of the connection between the call and the device.
ClientAddress	STRING	The IP address of the phone.
AgentID (optional)	STRING	The agent’s ACD login ID.
AgentExtension (optional)	STRING	The agent’s ACD teleset extension
AgentInstrument (optional)	STRING	The agent’s ACD instrument number

OnCallRTPStopped

The OnCallRTPStopped event indicates that an RTP media has been stopped. This event accompanies the call object in an Unified CCE environment.

Syntax

C++: `void OnCallRTPStopped(Arguments& args)`
COM: `void OnCallRTPStopped (IArguments * args)`
VB: `session_OnCallRTPStopped (ByVal args As CtiosCLIENTLib.IArguments)`

Parameters

args

Arguments array containing the following fields.

Table 6-78 OnCallRTPStopped Parameters

OnStartSilentMonitorConf

The OnStartSilentMonitorConf event is sent to the monitoring application to indicate that a StartSilentMonitorRequest has been processed at the CTI OS server.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-79 OnStartSilentMonitorConf Parameters

OnSilentMonitorStartedEvent

For CTI OS Based Silent Monitor

The OnSilentMonitorStartedEvent event is fired to the subscriber to indicate that a silent monitor session has been started on its behalf and that audio transmission to the monitoring client has been started.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-80 OnSilentMonitorStartedEvent Parameters

Keyword	Type	Description
MonitoredUniqueObjectID	STRING	Unique Object ID of the object being monitored
AgentID	STRING	Agent ID of the agent being monitored. This message will contain either AgentID or DeviceID, but not both.

DeviceID	STRING	Device ID of the agent being monitored. This message will contain either AgentID or DeviceID, but not both.
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where silent monitoring has started.
MonitoringIPAddress	STRING	TCP/IP address of the monitoring application
MonitoringIPPort	INT	TCP/IP port of the monitoring application
SMSessionKey	UNSIGNED SHORT	Unique identifier for the Silent Monitor Session.
HeartbeatInterval	INT	Heartbeat interval for the silent monitor session
HeartbeatTimeout	INT	Timeout for no activity.
OriginatingServerID	STRING	TCP/IP Address:Port of the CTI OS server from which the request originated
OriginatingClientID	STRING	Client Identification of the monitoring application

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 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 may receive multiple OnSilentMonitorStartedEvents.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-81 OnSilentMonitorStartedEvent

Keyword	Type	Description
SilentMonitorInitiatingAgentUID	STRING	Unique object ID of the agent that initiated silent monitor.
SilentMonitorInitiatingDeviceID	STRING	ID of the device that initiated silent monitor.
SilentMonitorTargetAgentUID	STRING	Unique object ID of the silently monitored agent.
SilentMonitorTargetDeviceID	STRING	ID of the silently monitored device.
SilentMonitorCallUID	STRING	Unique object ID of the silent monitor call.

OnSilentMonitorStartRequestedEvent

The OnSilentMonitorStartRequestedEvent event is fired to the subscriber to indicate that a silent monitor session request has arrived and that it will be established on its behalf if the DoDefaultMessageHandling parameter is set to True. The default behavior is to start sending audio and establish the session automatically. If the subscriber wishes to process the event by itself, it must set the DoDefaultMessageHandling parameter to False and invoke AcceptSilentMonitoring when it is ready to start the session and call ReportSMSessionStatus to the monitoring client.

CTI OS server generates this event when ever a remote application calls the StartSilentMonitorRequest method.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-82 OnSilentMonitorStartRequetsedEvent Parameters

Keyword	Type	Description
MonitoredUniqueObject ID	STRING	Unique Object ID of the object being monitored.
AgentID	STRING	Agent ID of the agent to be monitored. This message will contain either AgentID or DeviceID, but not both.
DeviceID	STRING	Device ID of the agent to be monitored. This message will contain either AgentID or DeviceID, but not both.
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where the silent monitor start has been requested.
MonitoringIPAddress	STRING	TCP/IP address of the monitoring application
MonitoringIPPort	INT	TCP/IP port of the monitoring application
SMSessionKey	UNSIGNED SHORT	Unique identifier for the Silent Monitor Session.
HeartbeatInterval	INT	Heartbeat interval for the silent monitor session
HeartbeatTimeout	INT	Timeout for no activity
OriginatingServerID	STRING	TCP/IP Address:Port of the CTI OS server from which the request originated

OriginatingClientID	STRING	Client Identification of the monitoring application
DoDefaultMessage Handling	BOOLEAN	When this parameter is set to True, it instructs the SilentMonitorManager to immediately start sending audio and establish the silent monitor session. If this parameter is set to False, it instructs the SilentMonitorManager to not send voice and to not establish the silent monitor session. In this case, it is the responsibility of the subscriber to report this status accordingly.

OnSilentMonitorSessionDisconnected

The OnSilentMonitorSessionDisconnected event is sent to the application to report errors if the connection fails between the monitoring and monitored clients

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-83 OnSilentMonitorSessionDisconnected Parameters

Keyword	Type	Description
MonitoredUniqueObject ID	STRING	Unique Object ID of the object being monitored
SMSessionKey	UNSIGNED SHORT	Unique identifier for the Silent Monitor Session.
StatusCode	SHORT	One of the ISilentMonitorEvent status codes in Table 6-87.

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 his behalf. CTI OS server generates this event whenever a remote application calls the StopSilentMonitorRequest method.

Syntax

```
C++:void OnSilentMonitorStopRequestedEvent(Arguments & args);
```

```
COM:  HRESULT OnSilentMonitorStopRequestedEvent([in] Arguments* args);
VB:  Session_ OnSilentMonitorStopRequestedEvent(ByVal args as CTIOSCLIENTLIB.IArguments)
```

Parameters

args

Arguments array containing the following fields.

Table 6-84 OnSilentMonitorStopRequestedEvent Parameters

Keyword	Type	Description
MonitoredUniqueObjectID	STRING	Unique Object ID of the object being monitored.
AgentID	STRING	Agent ID of the agent who had been monitored. This message will contain either AgentID or DeviceID, but not both.
DeviceID	STRING	Device ID of the agent who had been monitored. This message will contain either AgentID or DeviceID, but not both.
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where silent monitoring has stopped.
MonitoringIPAddress	STRING	TCP/IP address of the monitoring application.
SMSessionKey	UNSIGNED SHORT	Unique identifier for the Silent Monitor Session.
OriginatingServerID	STRING	TCP/IP Address:Port of the CTI OS server from which the request originated.
OriginatingClientID	STRING	Client Identification of the monitoring application.

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 will be set to the error returned in OnControlFailure.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-85 OnSilentMonitorStopRequestedEvent Parameters

OnSilentMonitorStatusReportEvent

The OnSilentMonitorStatusReportEvent event indicates a change in status of a silent monitor session. This event sent only to the monitoring application.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-86 OnSilentMonitorStatusReportEvent Parameters

Keyword	Type	Description
MonitoredUniqueObject ID	STRING	Unique Object ID of the object being monitored
SMSessionKey	UNSIGNED SHORT	Unique identifier for the Silent Monitor Session.
StatusCode	SHORT	One of the ISilentMonitorEvent status codes in Table 6-87.
OriginatingServerID	STRING	TCP/IP Address:Port of the CTI OS server from which the request originated
OriginatingClientID	STRING	Client Identification of the monitoring application
TargetCILClientID	STRING	CIL Client ID of the monitoring application

Table 6-87 ISilentMonitorEvent Status Codes

enum Value	Numeric Value (Hex)
General Codes	
eSMStatusUnknown	-1
eSMStatusOK	0
eSMStatusFailed	0x00000001
eSMStatusComError	0x00000002
eSMStatusMonitorStarted	0x00000003
eSMStatusMonitorStopped	0x00000004
eSMStatusHeartbeatTimeout	0x00000005
eSMStatusOutOfMemory	0x00000006
eSMStatusPortUnavailable	0x00000007
eSMStatusIncorrectStateForThisAction	0x00000008
eSMStatusResourceError	0x00000009
eSMStatusRejectedBadParameter	0x0000000A
eSMStatusWinsockError	0x0000000B
eSMStatusMediaTerminationNotPresent	0x0000000C
eSMStatusIPPhoneInformatioNotAvailable	0x0000000D
eSMStatusMissingParameter	0x0000000E
eSMStatusSessionNotFound	0x0000000F
eSMStatusSessionAlreadyExists	0x00000010
eSMStatusDisconnected	0x00000011
eSMStatusInvalidStateForAction	0x00000012
eSMStatusInProgress	0x00000013
eSMStatusMaxSessionsExceeded	0x00000014
eSMStatusCCMSilentMonitor	0x00000015
Silent Monitor Session Codes	
eSMStatusSessionTerminatedAbnormally	0x10000000
eSMStatusRejectedAlreadyInSession	0x10000001
eSMStatusRejectedWinPcapNotPresent	0x10000002
eSMStatusWinPcapError	0x10000003
eSMStatusMediaUnknownCodec	0x10000004
eSMStatusIncorrectSessionMode	0x10000005
eSMStatusPeerSilentMonitorNotEnabled	0x10000006
eSMStatusSilentMonitorNotEnabled	0x10000007
eSMStatusNoResponseFromPeer	0x10000008
eSMStatusPeerLoggedOut	0x10000009

Table 6-87 ISilentMonitorEvent Status Codes

enum Value	Numeric Value (Hex)
eSMStatusSessionTerminatedByMonitoredClient	0x1000000A
eSMStatusSessionTerminatedByMonitoringClient	0x1000000B
eSMStatusNoRTPPacketsReceivedFormIPPhone	0x1000000C
eSMStatusSessionConnectionToDelegateLost	0x1000000D
eSMStatusMTError	0x20000000
Voice Capture-Specific Codes	
eSMStatusWPNoPacketsReceived	0x30000000
eSMStatusWPFailedToOpenDevice	0x30000001
eSMStatusWPFailedToSetFilterExp	0x30000002
eSMStatusWPErrorInFilterExp	0x30000003

OnStopSilentMonitorConf

This OnStopSilentMonitorConf event is sent to the monitoring application to indicate that a StopSilentMonitorRequest has been processed at the CTI OS server.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-88 OnStopSilentMonitorConf Parameters

Keyword	Type	Description
MonitoredUniqueObjectID	STRING	Unique Object ID of the object being monitored
AgentID	STRING	Agent ID of the agent who had been monitored. This message will contain either AgentID or DeviceID, but not both.
DeviceID	STRING	Device ID of the agent who had been monitored. This message will contain either AgentID or DeviceID, but not both.
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where silent monitoring has stopped.
MonitoringIPAddress	STRING	TCP/IP address of the monitoring application

SMSessionKey	UNSIGNED SHORT	Unique identifier for the Silent Monitor Session.
OriginatingServerID	STRING	TCP/IP Address:Port of the CTI OS server from which the request originated
OriginatingClientID	STRING	Client Identification of the monitoring application

OnRTPStreamTimedoutEvent

The OnRTPStreamTimedoutEvent event is sent to the monitored application to report that no RTP voice packets have been received from the monitored IP Phone.

Syntax

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

Parameters

args

Arguments array containing the following fields.

Table 6-89 OnRTPStreamlinedEvent Parameters

Keyword	Type	Description
MonitoredUniqueObjectID	STRING	Unique Object ID of the object being monitored
SMSessionKey	UNSIGNED SHORT	Unique identifier for the Silent Monitor Session.
StatusCode	SHORT	One of the ISilentMonitorEvent status codes in Table 6-87.

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: void OnEvent(int iEventID, Arguments rArgs
```

.NET: `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.

```
int addAllInOneEventListener(IAllInOne  allInOneEvents)
int removeAllInOneEventListener(IAllInOne  allInOneEvents)
```

IAgentEvents

The following methods subscribe and unsubscribe the CTI OS Session Object for the IAgentEventsinterface.

```
int addAgentEventListener(IAgentEvents  agentEvents)
int removeAgentEventListener(IAgentEvents  agentEvents)
```

IButtonEnablementEvents

The following methods subscribe and unsubscribe the CTI OS Session Object for the IButtonEnablementEvents interface.

```
int addButtonEnablementEventListener(IButtonEnablementEvents buttonEvents)
int removeButtonEnablementEventListener(IButtonEnablementEvents buttonEvents)
```

ICallEvents

The following methods subscribe and unsubscribe the CTI OS Session Object for the ICallEvents interface.

```
int addCallEventListener (ICallEvents callEvents)
int removeCallEventListener (ICallEvents callEvents)
```

ISkillGroupEvents

The following methods subscribe and unsubscribe the CTI OS Session Object for the ISkillGrouEvents interface.

```
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:

```
int AddEventListener(IGenericEvents Listener, int iListID)
```

where Listener is the IGenericEvents object that is subscribing for events and iListID is the ID of the subscriber list to add this listener to. Java subscriber list IDs are part of the CtiOs_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). See the Javadoc file For more information on the CtiOs_Enums.SubscriberList interface.

The IGenericEvents interface, though it contains the C++/COM/VB events documented in this chapter, does not have a callback method for each event. Instead, the OnEvent method passes the event ID code and arguments for each event. The OnEvent method has the following syntax:

```
void OnEvent(int iEventID, Arguments rArgs)
```

where iEventID is the event ID code for the event and rArgs is the arguments for the event. The arguments for each Java event are the same as for the corresponding C++/COM/VB event. See the Javadoc file for details on the IGenericEvents interface.

To unsubscribe for events in the Java CIL, use the RemoveEventListener method. This method has the following syntax:

```
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 is subscribing 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.

Getting All Event Parameters

How to Get All Parameters from an Event

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 6-90 MinimizeEventArgs event parameters

Event Name	Parameters
eCallRetrievedEvent	CTIOS_RETRIEVINGDEVICEID
	CTIOS_RETRIEVINGDEVICEIDFULL
	CTIOS_ENABLEMENTMASK
	CTIOS_ICMENTERPRISEUNIQUEID
	CTIOS_UNIQUEOBJECTID
	CTIOS_DEVICEUNIQUEOBJECTID
	CTIOS_CALLSTATUS*
	CTIOS_FILTERTARGET**
eCallHeldEvent	CTIOS_HOLDINGDEVICEID
	CTIOS_HOLDINGDEVICEIDFULL
	CTIOS_ENABLEMENTMASK
	CTIOS_ICMENTERPRISEUNIQUEID
	CTIOS_UNIQUEOBJECTID
	CTIOS_DEVICEUNIQUEOBJECTID
	CTIOS_FILTERTARGET**
	CTIOS_CALLSTATUS*
eCallConnectionClearedEvent	CTIOS_RELEASINGDEVICEID
	CTIOS_RELEASINGDEVICEIDFULL
	CTIOS_ENABLEMENTMASK
	CTIOS_ICMENTERPRISEUNIQUEID
	CTIOS_UNIQUEOBJECTID
	CTIOS_DEVICEUNIQUEOBJECTID
	CTIOS_FILTERTARGET**
	CTIOS_CALLSTATUS*

Event Name	Parameters
eCallTransferredEvent	CTIOS_PRIMARYCALLID CTIOS_SECONDARYCALLID CTIOS_TRANSFERRINGDEVICEID CTIOS_TRANSFERRINGDEVICEIDFULL CTIOS_TRANSFERREDDEVICEID CTIOS_TRANSFERREDDEVICEIDFULL CTIOS_NUMPARTIES ConnectedParty[PartyNumber] CTIOS_ISTRANSFERCONTROLLER GenerateCallDataUpdateArgs()*** CTIOS_ENABLEMENTMASK CTIOS_ICMENTERPRISEUNIQUEID CTIOS_UNIQUEOBJECTID CTIOS_DEVICEUNIQUEOBJECTID CTIOS_FILTERTARGET** CTIOS_CALLSTATUS*
eCallConferencedEvent	CTIOS_PRIMARYCALLID CTIOS_SECONDARYCALLID CTIOS_CONTROLLERDEVICEID CTIOS_CONTROLLERDEVICEIDFULL CTIOS_ADDEDPARTYDEVICEID CTIOS_ADDEDPARTYDEVICEIDFULL CTIOS_PRIMARYDEVICEID CTIOS_PRIMARYDEVICEIDFULL CTIOS_SECONDARYDEVICEID CTIOS_SECONDARYDEVICEIDFULL CTIOS_NUMPARTIES ConnectedParty[PartyNumber] GenerateCallDataUpdateArgs()*** CTIOS_ENABLEMENTMASK CTIOS_ICMENTERPRISEUNIQUEID CTIOS_UNIQUEOBJECTID CTIOS_DEVICEUNIQUEOBJECTID CTIOS_FILTERTARGET** CTIOS_CALLSTATUS*

Event Name	Parameters
eCallBeginEvent, eCallDataUpdateEvent	GenerateCallDataUpdateArgs()*** CTIOS_DEVICEID CTIOS_DIVERTINGDEVICEID CTIOS_DIVERTINGDEVICEIDFULL CTIOS_ENABLEMENTMASK CTIOS_ICMENTERPRISEUNIQUEID CTIOS_UNIQUEOBJECTID CTIOS_DEVICEUNIQUEOBJECTID CTIOS_FILTERTARGET** CTIOS_CALLSTATUS*
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

Event Name	Parameters
eCallEstablishedEvent	CTIOS_ANSWERINGDEVICEID
	CTIOS_ANSWERINGDEVICEIDFULL
	CTIOS_CALLINGDEVICEID
	CTIOS_CALLINGDEVICEIDFULL
	CTIOS_CALLEDDEVICEID
	CTIOS_CALLEDDEVICEIDFULL
	CTIOS_SKILLGROUPID
	CTIOS_SKILLGROUPNUMBER
	CTIOS_SKILLGROUPPRIORITY
	CTIOS_SERVICEID
	CTIOS_SERVICENUMBER
	CTIOS_LINETYPE
	CTIOS_MEASUREDCALLQTIME
	CTIOS_CAMPAIGNID
	CTIOS_QUERYRULEID
	CTIOS_ENABLEMENTMASK
	CTIOS_ICMENTERPRISEUNIQUEID
	CTIOS_UNIQUEOBJECTID
	CTIOS_DEVICEUNIQUEOBJECTID
	CTIOS_FILTERTARGET**
	CTIOS_CALLSTATUS*

Event Name	Parameters
eCallDeliveredEvent	CTIOS_ALERTINGDEVICEID CTIOS_ALERTINGDEVICEIDFULL CTIOS_CALLINGDEVICEID CTIOS_CALLEDDEVICEID CTIOS_CALLINGDEVICEIDFULL CTIOS_CALLEDDEVICEIDFULL CTIOS_SKILLGROUPID CTIOS_SKILLGROUPNUMBER CTIOS_SKILLGROUPPRIORITY CTIOS_SERVICEID CTIOS_SERVICENUMBER CTIOS_LINETYPE CTIOS_MEASUREDCALLQTIME CTIOS_CAMPAIGNID CTIOS_QUERYRULEID CTIOS_ENABLEMENTMASK CTIOS_ICMENTERPRISEUNIQUEID CTIOS_UNIQUEOBJECTID CTIOS_DEVICEUNIQUEOBJECTID CTIOS_FILTERTARGET** CTIOS_CALLSTATUS*

Event Name	Parameters
eCallServiceInitiatedEvent, eCallOriginatedEvent, eCallQueuedEvent, eCallDequeuedEvent	CTIOS_CALLINGDEVICEIDFULL CTIOS_CALLEDDEVICEIDFULL CTIOS_CALLINGDEVICEID CTIOS_CALLEDDEVICEID CTIOS_SKILLGROUPID CTIOS_SKILLGROUPNUMBER CTIOS_SKILLGROUPPRIORITY CTIOS_SERVICEID CTIOS_SERVICENUMBER CTIOS_LINETYPE CTIOS_MEASUREDCALLQTIME CTIOS_CAMPAIGNID CTIOS_QUERYRULEID CTIOS_ENABLEMENTMASK CTIOS_ICMENTERPRISEUNIQUEID CTIOS_UNIQUEOBJECTID CTIOS_DEVICEUNIQUEOBJECTID CTIOS_FILTERTARGET[**] CTIOS_CALLSTATUS[*]
eControlFailureConf	CTIOS_PERIPHERALERRORCODE CTIOS_ERRORMESSAGE CTIOS_FAILURECODE CTIOS_ENABLEMENTMASK CTIOS_ICMENTERPRISEUNIQUEID CTIOS_UNIQUEOBJECTID CTIOS_DEVICEUNIQUEOBJECTID CTIOS_FILTERTARGET[**] CTIOS_CALLSTATUS[*]

Event Name	Parameters
eFailureConf, eFailureEvent, eCallFailedEvent	CTIOS_ERRORMESSAGE CTIOS_FAILURECODE CTIOS_ENABLEMENTMASK CTIOS_ICMENTERPRISEUNIQUEID CTIOS_UNIQUEOBJECTID CTIOS_DEVICEUNIQUEOBJECTID CTIOS_FILTERTARGET** CTIOS_CALLSTATUS*
eCallEndEvent	CTIOS_DEVICEID CTIOS_ENABLEMENTMASK CTIOS_ICMENTERPRISEUNIQUEID CTIOS_UNIQUEOBJECTID CTIOS_DEVICEUNIQUEOBJECTID CTIOS_FILTERTARGET** CTIOS_CALLSTATUS*

* If the eCallFailedEvent notification is received, the CTIOS_CALLSTATUS parameter will not be 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, CTIOS_ANI,
CTIOS_USERTOUSERINFO, CTIOS_DNIS, CTIOS_DIALEDNUMBER,
CTIOS_CALLERENTEREDDIGITS, 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
```

CHAPTER 8

Session Object

The Client Interface Library's Session object is used to establish a connection to an active CTI OS server. The main functions of the Session object are:

- Managing the connection to the CTI OS Server
- Distributing events to the appropriate objects and event subscribers
- Creating and managing the collections of Agent, Call, and SkillGroup objects
- Automatically recovering from failures

Typically, an application has a single instance of the Session object, which is used by all other CIL objects to send and receive events. However, there are no restrictions on the number or types of Session objects one application can employ. It is possible, and sometimes desirable, to establish and manage multiple independent Sessions, for example to use multiple current event streams. If there is more than one Session object monitoring the same Agent or Call, each Session object will receive its own events. The order in which events are received is not guaranteed when there are multiple Session objects.

For a detailed explanation of using the Session object to connect with CTI OS Server, see the section "Connecting to the CTI OS Server" in Chapter 4, "Building Your Application."

The Session object creates new Call, Agent, and SkillGroup objects upon receipt of an event for that object if the targeted object does not already exist. The Session object maintains collections of all Agents, Calls, SkillGroups, and WaitObjects. Object lifetime is managed by the Session object, and thus it is important that the client application not delete the objects, which would render the object reference invalid and lead to unpredictable results. When the Session is Released, the connection to CTI OS server is dropped. Any remaining Agent, Call, Skill Group, or WaitObjects will be released.

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

Session Object Properties

Table 8-1 lists the available Session properties.

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

Table 8-1 Session Properties

Keyword	Type	Description
CCMBasedSilent Monitor	INT	If this value is present and set to true, supervisor applications should 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 will need to invoke silent monitor using the SilentMonitorManager object. This property only applies to the COM CIL.
ConnectedSince	INT	Time of day in milliseconds when connected.
ConnectionMode	INT	eAgentConnection (1), eMonitorConnection (2), or eNotConnected (0).
CtiosA	STRING	Name or TCP/IP address passed as CTI OS server A.
CtiosB	STRING	Name or TCP/IP address passed as CTI OS server B
CurrentAgent	object reference	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.
CurrentCall	object reference	Valid only if in Agent Connect mode. When there is more than one call, this references the current call. The current call is the call selected. For additional information, refer to CurrentCall in Chapter 10, "Call Object."
CurrentPort	INT	TCP/IP address of the current connected CTI OS server. May be port A or B.
CurrentServer	STRING	Name or TCP/IP address of the current connected CTI OS server. The value is blank when the client is not connected to any server. The name may be blank while attempting to reconnect after a lost connection. Otherwise, the name of the server should be the name of CTI OS server A or B.
ForcedDisconnect	INT	The presence of this keyword,
Heartbeat	INT	Heartbeat time, expressed in seconds. If not set, default heartbeats are configurable on the CTI OS server.
LastError	INT	Last error code, if any. Otherwise this value is 0.
MaxHeartbeats	INT	Max heartbeats that can be missed before switching CTI OS servers. Default is 3 missed heartbeats.
MessageFilter	STRING	The filter that controls the events received by the CIL.
Object References	ARGUMENTS	Array of object references maintained by the session object. Typically includes Agent References, CallReferences, and SkillGroupReferences. Can also include EmailReferences or Chat References.
PortA	INT	TCP/IP port for ctiosA.
PortB	INT	TCP/IP port for ctiosB.
TryingPort	INT	TCP/IP address of the server where a connection is being attempted. May be port A or B.

Table 8-1 Session Properties (continued)

Keyword	Type	Description
TryingServer	STRING	Contains the name or TCP/IP address of the server where a connection is being attempted. The value is blank if no connection is being attempted (see CurrentServer). The name of the server should be the name of CTI OS server A or B.
TryingSince	INT	Time of day in milliseconds when try began.

Methods

Table 8-2 lists the available session object methods.

Table 8-2 Session Object Methods

Method	Description
AddEventListener	Subscribes a .NET IGenericEvents object as a listener on a particular subscriber list.
AddListener methods	Registers the subscriber for an event listener.
Connect	Establishes a connection to a CTI OS server.
CreateSilentMonitorManager	Creates a SilentMonitorManager object instance.
CreateWaitObject	Creates and returns the pointer to a new CWaitObject.
DestroySilentMonitor Manager	Deletes a SilentMonitorManager object instance.
DestroyWaitObject	Destroys the specified wait object
Disconnect	Closes the connection to the CTI OS server.
DumpProperties	See Chapter 7, "CtiOs Object."
GetAllAgents	Returns a collection of all the agents in the session.
GetAllCalls	Returns a collection of all the calls in the session.
GetAllProperties	See Chapter 7, "CtiOs Object."
GetAllSkillGroups	Returns a collection of all the skill groups in the session.
GetCurrentAgent	Returns the currently selected agent.
GetCurrentCall	Returns the currently selected call.
GetCurrentSilentMonitor	Returns a pointer to the SilentMonitorManager object instance that is set as the current manager in the CTI OS session object.
GetElement	See Chapter 7, "CtiOs Object."
GetNumProperties	See Chapter 7, "CtiOs Object."
GetObjectFromObjectID	Returns a Call, Agent, or SkillGroup, given the object's UniqueObjectID.
GetPropertyName	See Chapter 7, "CtiOs Object."
GetPropertyType	See Chapter 7, "CtiOs Object."
GetValue	See Chapter 7, "CtiOs Object."

Table 8-2 Session Object Methods (continued)

Method	Description
GetValueArray	See Chapter 7, "CtiOs Object."
GetValueInt	See Chapter 7, "CtiOs Object."
GetValueString	See Chapter 7, "CtiOs Object."
IsAgent	Checks the current agent and returns true if the current agent is an agent and not a supervisor.
IsCCMSilentMonitor	The IsCCMSilentMonitor method determines whether CTI OS has been configured to use CCM based silent monitor. This method is only supported with the C++, Java, COM, and .Net CILs.
IsSupervisor	Checks the current agent and returns true if the current agent is a supervisor.
IsValid	See Chapter 7, "CtiOs Object."
RemoveListener methods	Unregisters the subscriber from an event listener.
RequestDesktopSettings	Sends a message request to the CTI OS Server to retrieve the desktop settings configured for this site.
SetAgent	Sets an agent to a session object.
SetCurrentCall	Associates the current call to a session object.
SetCurrentSilentMonitor	Sets the SilentMonitorManager object instance specified as the current manager in the CTI OS session object.
SetMessageFilter	Sets the message filter that controls the set of events sent to the CIL.
SetSupervisorSilentMonitor Mode	Forces supervisors into monitored mode.
SetValue	See Chapter 7, "CtiOs Object."

AddEventListener (Java and .NET only)

The AddEventListener method subscribes an IGenericEvents object as a listener on a particular subscriber list.

Syntax

```
JAVA:   int AddEventListener(IGenericEvents Listener, int iListID)
.NET    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

```
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

Also, see the section "Subscribing for Events in C++" in Chapter 4, "Building Your Application."

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 Table 8-3.

Table 8-3 Connect Parameters

Keyword	Type	Description
CtiosA	STRING	Name or TCP/IP address of CTI OS server A. If this value is not provided, the value of Ctios B is used. **Note** If values of neither Ctios A or Ctios B is provided, an error is returned.
CtiosB	STRING	Name or TCP/IP address of CTI OS server B. If this value is not provided, the value of Ctios A is used. **Note** If values of neither Ctios A or Ctios B is provided, an error is returned.
PortA (optional)	INT	TCP/IP port for ctiosA, default = 42028.
PortB (optional)	INT	TCP/IP port for ctiosB, default = 42028.
Heartbeat (optional)	INT	Heartbeat time, expressed in seconds. If not set, default heartbeats are configurable on CTI OS server.

errorcode

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

Return Values

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

Remarks

A successful request will result in an OnConnection event.

A failure will result in an OnConnectionFailure event. This means that the CIL is in failover. The CIL will continue to attempt to connect, alternating between hosts CTIOS_CTIOSA and CTIOS_CTIOSB until connection succeeds at which point CIL will fire OnConnection. If application wishes to stop failover, it must call Disconnect.

In some cases, additional failure codes and events may occur:

- Connect will return a failure code of -1 if it cannot connect with the initial side of the duplexed CTI OS server pair chosen from the connect parameters. This error code requires no action on the part of the developer as the CIL will automatically attempt to connect using the parameters corresponding to the other side of the duplexed pair.
- The CIL will retry the connection attempt five times and then will not attempt to reconnect any longer. The final OnConnectionFailure event will contain the keyword "FinalAttempt" which informs the client application that the CIL has discontinued its attempts to reconnect.

Note This behavior will only occur after global settings download has completed. If global settings download has not completed, the CIL will continue to retry until successful.

- The Connect method will cause an OnCTIOSFailure event to be fired to the client indicating the current state of the system. This is in addition to OnConnection or OnConnectionFailure.

The following error codes can occur:

- **CIL_OK -** no obvious errors, application should wait for an event indicating whether or not Connect has succeeded
- **CIL_FAIL** - initial attempt to connect with host has failed. CIL will fire OnConnectionFailure and go into failover mode. CIL will continue to attempt to connect, alternating between hosts CTIOS_CTIOSA and CTIOS_CTIOSB until connection succeeds at which point CIL will fire OnConnection. If application wishes to stop failover, it must call Disconnect.
- **E_CTIOS_INVALID_ARGUMENT** - a null Arguments parameter was supplied. Connect is aborted. No events are fired.
- **E_CTIOS_MISSING_ARGUMENT** - indicates that method call provided no value for both CTIOS_CTIOSA or CTIOS_CTIOSB. At least one of these values must be provided. Connect is aborted. No events are fired.
- **E_CTIOS_IN_FAILOVER** - a previous call to connect failed and CIL is currently in failover attempting to establish a connection. This will continue until a connection is established at which point the CIL will send OnConnection indicating that previous call to Connect has succeeded. If developer wishes to call Connect again with different parameters, he/she must call Disconnect prior to calling Connect again.
- **E_CTIOS_MODE_CONFLICT** - Session is not disconnected (i.e a previous call to Connect is in progress or session is already connected). Disconnect must be called before attempting to establish another connection. CIL may fire an OnConnection event corresponding to previous call to Connect if connection was in progress but will not fire one corresponding to this method call.
- **E_CTIOS_SESSION_NOT_CONNECTED** - unanticipated error. Connect is aborted. No events are fired.

CreateSilentMonitorManager

The CreateSilentMonitorManager method creates a SilentMonitorManager object instance. To delete the object you must call DestroySilentMonitorManager.

Syntax

C++: `CSilentMonitorManager * CreateSilentMonitorManager(Arguments & args);`
COM: `HRESULT CreateSilentMonitorManager ( /*[in]*/ IArguments * args, /*[out,retval]*/ ISilentMonitorManager * * 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 to be initialized with the default values specify an empty array.

Table 8-4 CrateSilentMonitorManager Parameters

Keyword	Type	Description
HeartbeatInterval	INT	Heartbeat interval for the silent monitor session.
HeartbeatTimeout	INT	Timeout for no activity.
MediaTerminationPort	INT	Required only if manager will be used in monitoring mode. TCP/IP port where monitored conversation will be sent for playback on system sound card.

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 8-1).

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: `WaitObject CreateWaitObject(Arguments rObjParam)`

Parameters

args (C++). rObjParam (Java)

A reference to an Arguments object that contains the list of events the object will wait for. The Arguments should contain values where the keys are “Event1” through “EventN” and the values are the enumerated event IDs.

Return Values

If successful it returns a pointer to the new Wait object. Otherwise, it returns NULL.

For more information about CWaitObject see Chapter 12, “Helper Classes.”

DestroySilentMonitorManager

The DestroySilentMonitorManager method deletes a SilentMonitorManager object instance.

Syntax

```
C++: int DestroySilentMonitorManager(CSilentMonitorManager * pSilentMonitor);
COM: HRESULT DestroySilentMonitorManager (/*[in]*/ ISilentMonitorManager *
pSilentMonitor, /*[out,retval]*/ int * errorcode);
VB: DestroySilentMonitorManager (ByVal pSilentMonitor As CTIOSCLIENTLIB.
ISilentMonitorManager) As Long
Java: Not available
.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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Values

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

Remarks

Supported for use with 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++:   void DestroyWaitObject(CWaitObject * pWaitObject)
Java:  void DestroyWaitObject(WaitObject rWaitObj)
.NET:  DestroyWaitObject(WaitObject rWaitObj)
```

Parameters

WaitObject

A pointer to the CWaitObject to be destroyed.

Return Values

None.

Remarks

For more information about CWaitObject see Chapter 12, "Helper Classes."

DisableSkillGroupStatistics (C++ , Java, and .NET)

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

Syntax

```
C++, Java:int DisableSkillGroupStatistics(Arguments & args)
.NET:     CilError DisableSkillGroupStatistics(Arguments rArgs)
```

Parameters

args

This parameter has two required values for PeripheralId and SkillGroupNumber. See the Remarks section for a code example.

Return Value

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

Remarks

C++ code example:

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

```
Java:  int Disconnect(Arguments args)
.NET:  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. This keyword should be added to the array if the session mode has not yet been set by SetAgent or SetSessionMode at the time of the disconnect.

errorcode

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

Return Values

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

DumpProperties

See Chapter 7, "CtiOs Object" for a description of the DumpProperties method.

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 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++/Java:int EnableSkillGroupStatistics(Arguments & args)
.NET:CilError EnableSkillGroupStatistics(Arguments rArgs)
```

Parameters

args

This parameter has two required values for PeripheralId and SkillGroupNumber. See the Remarks section for a code example.

Return Value

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

Remarks

C++ code example:

```
Arguments & argsStatBroadcast = Arguments::CreateInstance();
argsStatBroadcast.AddItem(CTIOS_SKILLGROUPNUMBER, intSG);
argsStatBroadcast.AddItem(CTIOS_PERIPHERALID, m_periphID);
m_pSkGrStatSession->EnableSkillGroupStatistics ( argsStatBroadcast );
argsStatBroadcast.Release();
```

GetAllAgents

The GetAllAgents method 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 has 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. See Chapter 3, "CIL Coding Conventions."

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.

```
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
args = 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

            Console.Out.WriteLine("--")

        End If

    End If
Next
```

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 has discovered through call events. For example, a CIL used in an agent desktop application will return 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. See Chapter 3, "CIL Coding Conventions."

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.

```
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 ;
            }
            cout << "--" << 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.

```
Dim args As Arguments
args = 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

    End If
Next
```

GetAllProperties

See Chapter 7, "CtiOs Object" for a description of the GetAllProperties method.

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++: `Arguments & GetAllSkillGroups()`

COM: `HRESULT GetAllSkillGroups(/*[out, retval]*/ VARIANT *args)`
VB: `GetAllSkillGroups (args As VARIANT)`
Java , .NET: `Arguments GetAllSkillGroups()`

Parameters

args

C++, Java, and .NET: A pointer or a fereence 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. See Chapter 3, "CIL Coding Conventions."

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. See Chapter 3, "CIL Coding Conventions."

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. This method can be used as a way for controls to communicate between each other which call is selected and therefore should be acted upon. Use this method rather than GetValue("CurrentCall").

Syntax

```
C++:   CCall * GetCurrentCall()
COM:   HRESULT GetCurrentCall(/*[out, retval]*/ ICall ** call)
VB:    GetCurrentCall () As CTIOSCLIENTLib.ICall
Java/.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 CTI OS return values. See Chapter 3, "CIL Coding Conventions."

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

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

GetNumProperties

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

GetObjectFromObjectID

Given a string containing the UniqueObjectID of a call, an agent, or a skill group, the GetObjectFromObjectID method returns a pointer to the associated object.

Syntax

```
C++:   bool GetObjectFromObjectID (string& uniqueObjectID,
           CCtiosObject ** object);
COM:   HRESULT GetObjectFromObjectID (/*[in]*/ BSTR uniqueObjectID,
/*[out]*/ IDispatch ** object, /*[out, retval]*/ VARIANT_BOOL * errorcode);
VB:    GetObjectFromObjectID(uniqueObjectID As String, object as IDispatch) As Boolean
Java:  CtiOsObject GetObjectFromObjectID(java.lang.String sUID)
.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 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

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

C++, VB, .NET: A boolean indicating success or failure of the method.

The Java version of this method return NULL if the value requested is not found or if there is an error.

Remarks

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

GetPropertyName

See Chapter 7, "CtiOs Object" for a description of the GetPropertyName method.

GetPropertyType

See Chapter 7, "CtiOs Object" for a description of the GetPropertyType method.

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. Refer to "OnQueryAgentStateConf" in Chapter 6, "Event Interfaces and Events" for a description of the SystemStatus.

GetValue Methods

See Chapter 7, "CtiOs Object" for a description of the GetValue, GetValueArray, GetValueInt, and GetValueString methods.

IsAgent

The IsAgent method determines whether the current agent is an agent rather than a supervisor.

Syntax

C++: `bool IsAgent()`
COM: `HRESULT IsAgent (VARIANT_BOOL *bIsAgent)`
VB: `IsAgent () As Boolean`
Java: `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 has been 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 has been 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

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

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 CtiOs_Enums.CilError code indicating success or failure.

RemoveListener Methods (C++ only)

The RemoveListener methods unregisters the subscriber from a specified event listener.

Syntax

```
int RemoveEventListener(Arguments & rArguments);
int RemoveSessionEventListener(ISessionEvents * pSessionEvents);
int RemoveCallEventListener(ICallEvents * pCallEvents);
int RemoveAgentEventListener(IAgentEvents * pAgentEvents);
int RemoveSkillGroupEventListener(ISkillGroupEvents *
    pSkillGroupEvents);
int RemoveButtonEnablementEventListener(IButtonEnablementEvents *
    pButtonEvents);
int RemoveAllInOneEventListener(IAllInOne * pAllInOneEvents);
int  RemoveSilentMonitorEventListener(ISilentMonitorEvents *
    pSilentMonitorEvents);
int RemoveSessionEventGenericListener(IGenericEvents *
    pSessionEvents);
int RemoveCallEventGenericListener(IGenericEvents * pCallEvents);
int RemoveAgentEventGenericListener(IGenericEvents * pAgentEvents);
int RemoveSkillGroupEventGenericListener(IGenericEvents *
    pSkillGroupEvents);
int RemoveButtonEnablementEventGenericListener(IGenericEvents *
    pButtonEvents);
int RemoveAllInOneEventGenericListener(IGenericEvents *
    pAllInOneEvents);
int  RemoveSilentMonitorEventGenericListener(IGenericEvents * pSilentMonitorEvents);
```

Remarks

Also, see the section "Subscribing for Events in C++" in Chapter 4, "Building Your Application."

RequestDesktopSettings

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

Syntax

C++: `int RequestDesktopSettings(Arguments& args)`
COM: `HRESULT RequestDesktopSettings(/* [in] */ IArguments *args, /*[out]*/ int * errorcode)`
VB: `RequestDesktopSettings (args As CTIOSCLIENTLib.IArguments) As Long`
Java: `int RequestDesktopSettings(int desktopType)`
.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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Values

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

Remarks

A successful RequestDesktopSettings request results in an OnGlobalSettingsDownloadConf event. For detailed information about the OnGlobalSettingsDownloadConf event, see "OnFailure Event" section on page 6-6.

SetAgent

The SetAgent method assigns an agent to this Session object. The agent object used as a parameter in this method should have the following properties set:

- CTIOS_AGENTID
- CTIOS_PERIPHERALID

To sign on a mobile agent, the following parameters must be set:

- CTIOS_AGENTCALLMODE
- CTIOS_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: CilError 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 3-2 in Chapter 3, "CIL Coding Conventions."

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 will not be sent.

- E_CTIOS_SET_AGENT_SESSION_DISCONNECT_REQUIRED - You attempted to execute SetAgent for a session in monitor mode. The SetAgent request will not be 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 will not be sent.

Note In the above error cases, the SetAgent request will not be sent to the CTI OS server, and the client application will 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 CtiOs_Enums.CilError codes:

- E_CTIOS_INVALID_SESSION -- if session is not connected.
- E_CTIOS_PROP_ATTRIBUTES_ACCESS_FAILED -- if unable to get the connection mode property
- E_CTIOS_SET_AGENT_SESSION_DISCONNECT_REQUIRED -- if SetAgent request was during a Monitor Mode session. The client application will need to call Disconnect first to clean up the connection mode and then call Connect again.
- E_CTIOS_AGENT_ALREADY_IN_SESSION -- if the agent is already assigned to the session object. The client application will need to call Disconnect first to clean up the connection mode and then call Connect again.
- E_CTIOS_ARGUMENT_ALLOCATION_FAILED -- if the application is unable to allocate memory.
- E_CTIOS_PROP_ATTRIBUTES_ACCESS_FAILED -- if an error occurred while accessing agent properties.

SetCurrentCall

The SetCurrentCall method assigns a call as the session's current call.

Syntax

C++:	`int SetCurrentCall(CCall& call)`
COM:	`HRESULT SetCurrentCall (/*{in]*/ICall *call, /*[out, retval]*/ errorcode`
VB:	`SetCurrentCall (call As CTIOSCLIENTLib.ICall)`
Java:	`int SetCurrentCall(Call callObject)`
.NET:	`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 3-2 in Chapter 3, "CIL Coding Conventions.".

Return Values

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

Remarks

A successful request results in an OnCurrentCallChanged event.

In Java 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++:   int SetCurrentSilentMonitor(CSilentMonitorManager * pSilentMonitor);
COM:   HRESULT SetCurrentSilentMonitor (/*[in]*/ ISilentMonitorManager * pSilentMonitor,
/*[out,retval]*/ int * errorcode);
VB:    SetCurrentSilentMonitor (ByVal pSilentMonitor As CTIOSCLIENTLIB.
ISilentMonitorManager) As Long
Java:  Not available
.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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Values

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

Remarks

Supported for use with Unified CCE only.

SetMessageFilter

The SetMessageFilter method specifies a filter for CTI OS Server to use to determine which events are sent to a monitor mode client.

Syntax

```
C++:  int SetMessageFilter(string filter)
COM:  HRESULT SetMessageFilter(/*{in]*/ BSTR filter, /*[out, retval]*/ int* errorcode)
VB:   SetMessageFilter (filter As String, retVal As Long)
Java: int SetMessageFilter(Arguments messageFilter)
.NET: CilError SetMessageFilter(Arguments rArgs)
```

Parameters

filter

A string containing the message filter, as explained in the section "Notes On Message Filters".

errorcode

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

Return Values

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

Remarks

The Session will receive an OnMonitorModeEstablished event when the filter is set on the server.

SetSupervisorMonitorMode

The SetSupervisorSilentMonitorMode method can be used 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++:  int SetSupervisorSilentMonitorMode (Arguments & args);
COM:  HRESULT SetSupervisorSilentMonitorMode (/*[in]*/ IArguments * args,
/*[out,retval]*/ int * errorcode);
VB:   SetSupervisorSilentMonitorMode (args As CTIOSCLIENTLib.IArguments);
Java/.NET: Not available
```

Parameters

args

Arguments array that contains the following parameters.

Table 8-5 SetSupervisorSilentMonitorMode Arguments Array Parameters

Keyword	Type	Description
CTIOS_SILENTMONITOR FORCEMONITOREDMODE	INT	One of the following values: 1 -- supervisors can be monitored 0 (default) -- supervisors are put in monitoring mode and cannot be monitored

errorcode

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

Return Values

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

Notes On Message Filters

A message filter is a condition that an event must meet in order to be sent to the client. It consists of a keyword/value pair, as explained in the following sections.

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`.

A Simple Example

The most basic event filter is for all events for a specific agent. CTI OS uniquely identifies an agent object by it's UniqueObjectID (refer to CIL architecture chapter for explanation of the UniqueObjectID). To establish an event stream for a unique agent, the following syntax would be used:

```
sFilterExpression = "UniqueObjectID=agent.5000.22866"
```

In this example, the key is the UniqueObjectID, and the value is agent.5000.22866. This is 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 can be expressed in the following general form:

```
key1=value1 [,value2, …] [; key2=valueA [,valueB, …] …]
```

In this form, the *filter expression* must start with a key name (`key`). Following the key must be an equal sign (=), and at least one value (`value1`) must be specified. Optionally, additional values (e.g. `value2, …`) for the same key might be specified (optional parts of the expression are indicated with square brackets `[]`). This will be interpreted as a logical OR among all of the values specified for that key, e.g. if any of those values is found, the condition will be satisfied.

For example, a *filter expression* with one key and multiple values might look like the following:

```
sFilterExpression = "AgentID=22866, 22867, 22868"
```

The interpretation of this filter is to match on any event with AgentID of 22866, 22867, or 22868.

Combining Filters

Multiple *filters expressions* (as described above) can be combined to create more complex expressions. The general form allows for any number of filters to be concatenated using the semicolon (`;`), which produces a logical AND effect.

For example, a *filter expression* with multiple keys, each with multiple values might look like the following:

```
sFilterExpression =
"AgentID=22866, 22867, 22868; SkillGroupNumber=20, 21"
```

The interpretation of this filter is to match on any event with AgentID of 22866, 22867, or 22868 *and* with SkillGroupNumber of 20 or 21.

Filtering for Specific Events

One of the most powerful types of event filters for custom applications are filters that work on specific events.

An example of such an application would be an "all agents" real time display, listing the agent states of all known agents at the call center, using the `eAgentStateEvent` to receive agent updates. To request specific events, use the `MessageID` keyword, and the numeric value for the event ID that you wish to receive:

```
' register for all eAgentStateEvents
sFilterExpression = "MessageID = 30"
```

It is also possible to obtain multiple specific events. For example, consider an all calls real-time display application, using `eCallBeginEvent` and `eCallEndEvent` to add or remove calls from a view:

```
' register for all eCallBeginEvents, eCallEndEvents
sFilterExpression = "MessageID = 23, 24"
```

Events Not Allowed In Filter Expressions

The following events cannot be used in filter expressions:

- ePreLogoutEvent
- ePostLogoutEvent
- eOnConnection
- eOnConnectionClosed
- eOnConnectionFailure
- eOnHeartbeat
- eOnMissingHeartbeat
- eOnCurrentCallChanged
- eOnCurrentAgentReset
- Events that are part of the IMonitoredAgentEvents interface or the IMonitoredCallsInterface. This includes the following events:
  - eOnMonitoredAgentStateChange
  - OnMonitoredAgentInfoEvent
  - OnMonitoredCallDeliveredEvent
  - OnMonitoredCallEstablishedEvent
  - OnMonitoredCallHeldEvent
  - OnMonitoredCallRetrievedEvent
  - OnMonitoredCallClearedEvent
  - OnMonitoredCallConnectionClearedEvent
  - OnMonitoredCallOriginatedEvent
  - OnMonitoredCallFailedEvent
  - OnMonitoredCallConferencedEvent
  - OnMonitoredCallTransferredEvent

– OnMonitoredCallDivertedEvent
– OnMonitoredCallServiceInitiatedEvent
– OnMonitoredCallQueuedEvent
– OnMonitoredCallTranslationRouteEvent
– OnMonitoredCallBeginEvent
– OnMonitoredCallEndEvent
– OnMonitoredCallDataUpdateEvent
– OnMonitoredCallReachedNetworkEvent
– OnMonitoredCallDequeuedEvent
– OnMonitoredAgentPrecallEvent
– OnMonitoredAgentPrecallAbortEvent

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.

```
void CMyEventSink::OnAgentStateEvent(Arguments & argParams)

{

     if (argParams.IsValid(CTIOS_MONITORED) &&
argParams.GetValueBoolean(CTIOS_MONITORED))

    {

        //Do process the event

    }

}
```

Table 8-6 Special Filter Keywords

Keyword	Description
FilterTarget=SkillGroupStats (see Filtering Skillgroup Statistics, page 8-32)	When set, this filter indicates that the CTI OS server should forward skill group statistics to the client application, whether or not any agents are logged in.
HideSilentMonitorCallEvents (see Filtering CCM Based Silent Monitor Calls, page 8-33)	This keyword is used to block call events for silent monitor calls in monitor mode applications.

Filtering Skillgroup Statistics

One of the most common applications for a filter mode application is the processing of only skill group statistics. For this purpose, the specialized filter *FilterTarget=SkillGroupStats* is defined. When set, this filter indicates that the CTI OS server should forward skill group statistics to the client application, whether or not any agents are logged in.

After the filter is set, the client application needs to invoke the EnableSkillGroupStatistics(...) method for each skill group that it is expecting to receive statistics. To stop receiving statistics for a given skill group, the application must invoke the DisableSkillGroupStatistics method.

```
'register to receive skill group statistics
sFilterExpression=”FilterTarget=SkillGroupStats"
'call SetMessageFilter
m_session.SetMessageFilter sFilterExpression
'Enable statistics for skills 78,89 and 90 in peripheral 5004
Private Sub m_Session_OnMonitorModeEstablished(ByVal pArguments As Arguments)
    Dim m_Args = new  Arguments
       'For Skill 78
        m_Args.AddItem "SkillGroupNumber",78
        m_Args.AddItem "PeripheralID",5004
        m_session.EnableSkillGroupStatistics m_Args
       'For Skill 89
        m_Args.Clear
        m_Args.AddItem "SkillGroupNumber",89
        m_Args.AddItem "PeripheralID",5004
        m_session.EnableSkillGroupStatistics m_Args
        'For Skill 90
        m_Args.Clear
        m_Args.AddItem "SkillGroupNumber",90
        m_Args.AddItem "PeripheralID",5004
        m_session.EnableSkillGroupStatistics m_Args
       'Don't need arguments any more
       Set m_Arg = Nothing
End Sub
Private Sub MyObjectClass_OnCleanupApplication()
        Dim m_Args = new  Arguments
       'For Skill 78
        m_Args.AddItem "SkillGroupNumber",78
        m_Args.AddItem "PeripheralID",5004
        m_session.DisableSkillGroupStatistics m_Args
       'For Skill 89
        m_Args.Clear
        m_Args.AddItem "SkillGroupNumber",89
        m_Args.AddItem "PeripheralID",5004
        m_session.DisableSkillGroupStatistics m_Args
        'For Skill 90
        m_Args.Clear
        m_Args.AddItem "SkillGroupNumber",90
        m_Args.AddItem "PeripheralID",5004
        m_session.DisableSkillGroupStatistics m_Args
       'Don't need arguments any more
       Set m_Arg = Nothing
End Sub
```

Filtering 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. Please see All Calls Sample.NET for an example of the use of this filter.

CHAPTER 7

CtiOs Object

All of the interface objects in the CTI OS 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 CTI OS client interface library. It is implemented 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 CTI OS 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

Table 7-1 lists the available CCtiOsObject class methods.

Table 7-1 CCtiOsObject Class Methods

Method	Description
DumpProperties	Returns a string listing all of an object's properties' names and values.
GetAllProperties	Returns all of the object's properties as Args (name/value pairs).
GetElement	Returns the value of an element.

Table 7-1 CCtiOsObject Class Methods

GetLastError	Returns the last error that occurred on the calling thread.
GetNumProperties	Returns the number of properties of an object.
GetPropertyName	Returns a property name in a string format.
GetPropertyType	Returns the data type of the specified property.
GetValue, GetValueInt, GetValueString, GetValueArray	Returns the value of a specified property.
IsValid	Checks to see if the property of an object is valid.

DumpProperties

The DumpProperties method returns all the properties of the object. This method builds a string showing all of the properties in the form "key1 = value1; key2 = value2;...".

Syntax

C++: `string DumpProperties ()`
COM:`HRESULT DumpProperties (/*[out,retval]*/ BSTR* bstrValue)`
VB: `DumpProperties() As String`
Java:`String DumpProperties()`
.NET:`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. See Chapter 3, "CIL Coding Conventions."

All Others: The string listing the names of all the object's properties.

GetAllProperties

The GetAllProperties method returns all of the object's properties and their values. For the properties that are calls, agents, or skillgroups, their string UniqueObjectIDs are returned, not the objects themselves. To get the objects themselves use GetObjectFromObjectID, explained in Chapter 8, "Session Object."

Syntax

C++: `bool GetAllProperties (Arguments** arguments)`

COM:`HRESULT GetAllProperties (/*[out]*/ IArguments** arguments, /*[out,retval]*/ VARIANT_BOOL* errorcode)`
VB:`GetAllProperties arguments As (CTIOSCLIENTLib.IArguments) As Bool`
Java, .NET:`Arguments GetAllProperties()`

Parameters

C++, COM, VB: arguments

Output parameter in the form of an arguments array that has all of the property names and values of the object.

errorcode

An output parameter (return parameter in VB) that contains a boolean indicating success or lack thereof.

Return Value

C++ , VB: True upon success and false upon failure.

COM: Always returns S_OK. The errorcode parameter should be used 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 of 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++: Arg& GetElement (string& key, int element)
    Arg& GetElement (int key, int element)
    Arg& GetElement (char* key, int element)
COM: HRESULT GetElement /*[in]*/ VARIANT* key, /*[in]*/ int element, /*[out,retval]*/
IArg** pIArg)
VB: GetElement (key As VARIANT) As CTIOSCLIENTLib.IArg
Java: Arg GetElement(String key, int element)
    Arg GetElement(int key, int element)
.NET: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.

COM, VB:pIArg

An output parameter (return parameter in VB) containing an IArg with the value of the desired element.

.NET: 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:Integer GetLastError()
.NET: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.

```
// 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++: `int GetNumProperties ()`
COM: `HRESULT GetNumProperties (/*[out,retval]*/ int * num`
VB: `GetNumProperties () As Long`
Java, .NET: `int GetNumProperties()`

Parameters

num

In the COM version, an output parameter (return value in VB, C++, Java, and .NET) that contains an integer that is the number of properties in the object.

Return Value

COM: Default CTI OS return values. See Chapter 3, "CIL Coding Conventions."

All Others: An integer that is the number of properties currently a part of the object.

GetPropertyName

The GetPropertyName method returns the name of a property in a string format.

Syntax

C++: `string GetPropertyName (int index)`
COM: `HRESULT GetPropertyName (/* [in] index, /*[out,retval]*/ name)`
VB: `GetPropertyName (index As Integer) As String`
Java: `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. See Chapter 3, "CIL Coding Conventions."

.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++: `int GetPropertyType (string& key)`
```
int GetPropertyType (int key)
int GetPropertyType (char* key)
```
COM:`HRESULT GetPropertyType (/*[in]*/ VARIANT* key, /*[out,retval]*/ int* value)`
VB:`GetPropertyType (key As VARIANT) As Int`
Java:`int GetPropertyType(string sPropName)`
```
int GetPropertyType(int key)
```
.NET:`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. See Chapter 3, "CIL Coding Conventions."

Others: An integer indicating the property's type with the following possible values:

Table 7-2 GetPropertyType Values

Argument Type	Description
ARG_NOTSET	Argument type not determined
ARG_INT	Signed integer
ARG_SHORT	2 bytes signed integer
ARG_BOOL	1 byte integer
ARG_STRING	**C++, COM:** STL character string **VB .NET:** String object
ARG_ARGARRAY	Variable length array of Arg
ARG_UINT	32 bit unsigned int
ARG_USHORT	16 bit unsigned short int
ARG_ARGUMENT	Arguments array
ARG_REFERENCE	Contains a reference to an object of a CtiOsObject derived class

GetValue

The GetValue method returns the value of the specified property. Use this method if you don't know the type of the property. Otherwise, use the more specific GetValue methods discussed later in this chapter. 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 Chapter 8, "Session Object."

Syntax

```
C++:Arg& GetValue (string& key)
    Arg& GetValue (int key)
    Arg& GetValue (char* key)
COM:HRESULT GetValue (/*[in]*/ VARIANT* key, /*[out,retval]*/ IArg** value)
VB: GetValue (key As VARIANT) As CTIOSCLIENTLib.IArg
Java:Arg  GetValue( String key )
    Arg& GetValue (int key)
.NETvirtual 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. See Chapter 3, "CIL Coding Conventions."

.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++:    Arg& GetValueArray (string& key)
        Arg& GetValueArray (enum_Keywords key)
        Arg& 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 )
        Arguments GetValueArray (int 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 arArguments** 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. See Chapter 3, "CIL Coding Conventions."

.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)
```

Parameters

Key

Key ID for the object to be retrieved.

Returns

A Boolean object representation of the contained value or null if error.

GetValueInt

The GetValueInt method returns the integer value of the specified property. Use this method when you know that the property has an integer type.

Syntax

```
C++:   int GetValueInt (string& key)
       int GetValueInt (int key)
       int GetValueInt (char* key)
COM:   HRESULT GetValueInt /*[in]*/ VARIANT* key, /*[out,retval]*/ int* value)
VB:    GetValueInt (key As VARIANT) As Integer
Java:  Not implemented, use GetValueIntObj
.NET:  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 prerty 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. See Chapter 3, “CIL Coding Conventions.”

.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.

Return Value

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++:string GetValueString (string& key)
    string GetValueString (int key)
    string GetValueString (char* key)
COM:HRESULT GetValueString (/*[in]*/ VARIANT* key, /*[out,retval]*/ BSTR* value)
VB: GetValueString (key As VARIANT) As String
Java:String GetValueString( String key )
    String GetValueString (int key)
.NET: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.

value

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. See Chapter 3, "CIL Coding Conventions."

.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( int iKey)
Integer GetValueUShortObj( String sKey)
```

Parameters

key

Key ID of the value to be retrieved.

Returns

An Integer object containing the 16 bit unsigned short value or `null` if error.

IsValid

The IsValid method tests to see if the object includes the specified property.

Syntax

```
C++:   bool IsValid (string& key)
       bool IsValid (char* key)
       bool IsValid (int key)
COM:   HRESULT IsValid (/*[in]*/ VARIANT* key, /*[out,retval]*/ VARIANT_BOOL* value)
VB:    IsValid (key As VARIANT)as Bool
Java:  boolean IsValid( String key)
       boolean IsValid (int key)
.NET:  virtual bool IsValid(Enum_CtiOs eKeyID)
       virtual bool IsValid(System.String sKey)
```

Parameters

C++, Java: key

A key containing the name or ID of the property whose validity you are testing.

COM, VB: key

VARIANT containing the name or ID of the property to retrieve.

.NET: eKeyID

The ID of the property whose validity you are testing.

.NET: sKey

The name of the property whose validity you are testing.

COM: value

An output parameter (return parameter in VB) containing a VARIANT_BOOL pointer indicating whether or not a property with the specified name exists for the object.

Return Value

COM: Default HRESULT return value. See Chapter 3, "CIL Coding Conventions."

Others: A boolean indicating whether or not a property with the specified name exists for the object.

ReportError (Java 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

```
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

```
boolean SetValue(int iKeyID, Arg rArg
boolean SetValue(int iKeyID, Arguments rArgs)boolean
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

C++:
```
bool SetValue( string& key, string& value)
    bool SetValue( string& keyValuePair)
    bool SetValue( string& key, int value)
    bool SetValue( const char * key, const char * value)
    bool SetValue( const char * keyValuePair)
    bool SetValue( const char * key, int value)
```
COM:
```
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 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

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

All Others: A boolean indicating the success or failure of the method.

Remarks

This method should only be used when creating a new Agent in preparation for logging in. Therefore, it should be used to set the AgentID, AgentInstrument, AgentPassword, PeripheralID, and AutoLogin *only*.

CHAPTER 9

Agent Object

The Agent object provides developers using the CTI OS Client Interface Library with an interface to Agent behavior. The Agent object exposes methods to perform all agent behaviors, such as logging in and setting the agent's state.

The object stores specific agent information as properties, including the AgentID, AgentPassword, AgentInstrument, AgentExtension, and SkillGroups. When the agent is logged into an ACD, the agent object receives updates through AgentStateEvents and Agent Statistics updates.

The Agent object can be used in two different modes:

- In Agent Mode, the application should create an Agent object and inform the Session about the agent using *Session.SetAgent()*.
- In Monitor Mode, the client application sets a message filter, and if the event stream involves events for Agent objects, those objects will be dynamically created at the CIL as needed.

Agent Object Properties

Table 9-1 lists the agent object properties.

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

Table 9-1 Agent Object Properties

Keyword	Type	Description
AgentAvailability Status	INT	One of the following values: UNKNOWN (-1), NOT AVAILABLE (0), ICM AVAILABLE (1), or APPLICATION AVAILABLE (2).
Agent CallMode	INT	A value that indicates the agent's call mode. Valid values are call-by-call (3) and nailed-up (4).
AgentExtension	STRING*	Extension associated by ACD to agent.
AgentID	STRING*	Can be set prior to Login or after Logout.
AgentInstrument	STRING*	Instrument associated by ACD to agent.
AgentRemote Number	STRING	The phone number that the agent uses for remote login.

Keyword	Type	Description
AgentState	SHORT	One of the values in Table 6-56 representing the current state of the associated agent.
ClassIdentifier	INT	Identifies the type of this object.
SilentMonitorCallUID	STRING	The unique object ID of the silent monitor call. This is the call that results from calling SuperviseCall() with the SupervisorAction set to eSupervisorMonitor. **Note** Only applies to Cisco Unified Communications Manager based silent monitor.
SilentMonitorTargetAgentUID	STRING	This property contains the unique object ID of the agent who the supervisor is currently silent monitoring. **Note** Only applies to Cisco Unified Communications Manager based silent monitor.
Extension		Extension associated by ACD to agent.
CurrentConnection Profile	STRING	The last selected agent connection profile.
IsSupervisor	INT	Indicates whether this agent is a supervisor.
LastError	INT	Last error code, if any. Otherwise this value is 0.
PeripheralID	INT	ID of peripheral.
PeripheralType	INT	The type of the peripheral.
SavedAgentID	STRING	On Spectrum, contains the user provided data for the agent specified by the AgentID property. On all other switches, this property is identical to the AgentID property.
Statistics	ARGUMENTS	An arguments array containing the statistics listed in Table 9-2.

*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.

Agent Statistics

Statistics can be accessed by first using GetValueArray on the Agent object to obtain the "Statistics" arguments array and then using GetValueInt on the "Statistics" arguments array to obtain the desired value:

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

' Then get the desired statistics
Dim availTimeSession As Integer
Dim loggedOnTimeSession As Integer
```

```
availTimeSession = args.GetValueInt("AvailTimeSession")
bargeInCallsToday = args.GetValueInt("BargeInCallsToday")
```

Note Not all the statistics values listed in Table 9-2 are present in every system configuration. Whether or not a particular statistic value is available depends both on the protocol version of CTI Server with which CTI OS connects and on the peripheral on which the agent resides.

Table 9-2 Agent Statistics

Statistic	Definition
AvailTime Session	Total time, in seconds, the agent was in the Available state for any skill group.
LoggedOnTime Session	Total time, in seconds, the agent has been logged on.
NotReadyTime Session	Total time, in seconds, the agent was in the Not Ready state for all skill groups.
ICMAvailable TimeSession	Total time, in seconds, the agent was in the Unified ICM Available state.
RoutableTime Session	Total time, in seconds, the agent was in the Routable state for all skill groups.
AgentOutCalls Session	Total number of completed outbound ACD calls made by agent.
AgentOutCalls TalkTimeSession	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.
AgentOutCalls Time Session	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.
AgentOutCalls Held Session	The total number of completed outbound ACD calls the agent has placed on hold at least once.
AgentOutCalls HeldTime Session	Total number of seconds outbound ACD calls were placed on hold.
HandledCalls Session	The number of inbound ACD calls handled by the agent.

Statistic	Definition
HandledCalls TalkTime Session	Total talk time in seconds for Inbound ACD calls counted as handled by the agent. Includes hold time associated with the call.
HandledCalls AfterCall TimeSession	Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.
HandledCalls Time Session	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.
IncomingCalls Held Session	The total number of completed inbound ACD calls the agent placed on hold at least once.
IncomingCalls HeldTime Session	Total number of seconds completed inbound ACD calls were placed on hold.
InternalCallsSession	Number of internal calls initiated by the agent.
InternalCalls TimeSession	Number of seconds spent on internal calls initiated by the agent.
InternalCalls RcvdSession	Number of internal calls received by the agent.
InternalCalls RcvdTime Session	Number of seconds spent on internal calls received by the agent.
InternalCalls Held Session	The total number of internal calls the agent placed on hold at least once.
InternalCalls HeldTime Session	Total number of seconds completed internal calls were placed on hold.
AutoOutCalls Session	Total number of AutoOut (predictive) calls completed by the agent.
AutoOutCalls TalkTime Session	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.

Statistic	Definition
AutoOutCalls Time Session	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.
AutoOutCalls Held Session	The total number of completed AutoOut (predictive) calls the agent has placed on hold at least once.
AutoOutCalls HeldTime Session	Total number of seconds AutoOut (predictive) calls were placed on hold.
PreviewCalls Session	Total number of outbound Preview calls completed by the agent.
PreviewCalls TalkTime Session	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.
PreviewCalls TimeSession	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.
PreviewCalls HeldSession	The total number of completed outbound Preview calls the agent has placed on hold at least once.
PreviewCalls HeldTime Session	Total number of seconds outbound Preview calls were placed on hold.
Reservation CallsSession	Total number of agent reservation calls completed by the agent.
Reservation CallsTalk TimeSession	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.

Statistic	Definition
Reservation CallsTime Session	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.
Reservation CallsHeld Session	The total number of completed agent reservation calls the agent has placed on hold at least once.
Reservation CallsHeld TimeSession	Total number of seconds agent reservation calls were placed on hold.
BargeInCalls Session	Total number of supervisor call barge-ins completed.
InterceptCalls Session	Total number of supervisor call intercepts completed.
MonitorCalls Session	Total number of supervisor call monitors completed.
WhisperCalls Session	Total number of supervisor whisper calls completed.
EmergencyCallsSession	Total number of emergency calls .
AvailTimeToday	Total time, in seconds, the agent was in the Available state for any skill group.
LoggedOnTime Today	Total time, in seconds, the agent has been logged on.
NotReadyTime Today	Total time, in seconds, the agent was in the Not Ready state for all skill groups.
ICMAvailable TimeToday	Total time, in seconds, the agent was in the Unified ICM Available state.
RoutableTime Today	Total time, in seconds, the agent was in the Routable state for all skill groups.
AgentOutCalls Today	Total number of completed outbound ACD calls made by agent.
AgentOutCalls TalkTime Today	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.

Statistic	Definition
AgentOutCalls Time Today	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.
AgentOutCalls HeldToday	The total number of completed outbound ACD calls the agent has placed on hold at least once.
AgentOutCalls HeldTime Today	Total number of seconds outbound ACD calls were placed on hold.
HandledCalls Today	The number of inbound ACD calls handled by the agent.
HandledCalls TalkTime Today	Total talk time in seconds for Inbound ACD calls counted as handled by the agent. Includes hold time associated with the call.
HandledCalls AfterCall TimeToday	Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.
HandledCalls TimeToday	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.
IncomingCalls HeldToday	The total number of completed inbound ACD calls the agent placed on hold at least once.
IncomingCalls HeldTime Today	Total number of seconds completed inbound ACD calls were placed on hold.
InternalCalls Today	Number of internal calls initiated by the agent.
InternalCalls TimeToday	Number of seconds spent on internal calls initiated by the agent.
InternalCalls RcvdToday	Number of internal calls received by the agent.
InternalCalls RcvdTime Today	Number of seconds spent on internal calls received by the agent.
InternalCalls HeldToday	The total number of internal calls the agent placed on hold at least once.
InternalCalls HeldTime Today	Total number of seconds completed internal calls were placed on hold.

Statistic	Definition
AutoOutCalls Today	Total number of AutoOut (predictive) calls completed by the agent.
AutoOutCalls TalkTime Today	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.
AutoOutCalls TimeToday	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.
AutoOutCalls HeldToday	The total number of completed AutoOut (predictive) calls the agent has placed on hold at least once.
AutoOutCalls HeldTime Today	Total number of seconds AutoOut (predictive) calls were placed on hold.
PreviewCalls Today	Total number of outbound Preview calls completed by the agent.
PreviewCalls TalkTimeToday	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.
PreviewCalls TimeToday	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.
PreviewCalls HeldToday	The total number of completed outbound Preview calls the agent has placed on hold at least once.
PreviewCalls HeldTimeToday	Total number of seconds outbound Preview calls were placed on hold.
Reservation CallsToday	Total number of agent reservation calls completed by the agent.

Statistic	Definition
Reservation CallsTalk TimeToday	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.
Reservation CallsTimeToday	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.
Reservation CallsHeldToday	The total number of completed agent reservation calls the agent has placed on hold at least once.
Reservation CallsHeld TimeToday	Total number of seconds agent reservation calls were placed on hold.
BargeInCalls Today	Total number of supervisor call barge-ins completed.
InterceptCalls Today	Total number of supervisor call intercepts completed.
MonitorCalls Today	Total number of supervisor call monitors completed.
WhisperCalls Today	Total number of supervisor whisper calls completed.
EmergencyCalls Today	Total number of emergency calls .
AvailTime Session	Total time, in seconds, the agent was in the Available state for any skill group.
LoggedOnTime Session	Total time, in seconds, the agent has been logged on.
NotReadyTime Session	Total time, in seconds, the agent was in the Not Ready state for all skill groups.
ICMAvailable TimeSession	Total time, in seconds, the agent was in the Unified ICM Available state.
RoutableTime Session	Total time, in seconds, the agent was in the Routable state for all skill groups.
AgentOutCalls Session	Total number of completed outbound ACD calls made by agent.

Methods

Table 9-3 lists the Agent object methods.

Table 9-3 Agent Object Methods

Method	Description
DisableAgentStatistics	Disables agent statistic messages.
DisableSkillGroupStatistics	Disables skill group statistic messages.
DumpProperties	See Chapter 7, "CtiOs Object."
EnableAgentStatistics	Enables agent statistic messages.
EnableSkillGroupStatistics	Enables skill group statistic messages.
GetAgentState	Returns the current agent state.
GetAllProperties	See Chapter 7, "CtiOs Object."
GetElement	See Chapter 7, "CtiOs Object."
GetMonitoredAgent	Returns the agent object that is currently being monitored.
GetMonitoredCall	Returns the call object that is currently being monitored.
GetNumProperties	See Chapter 7, "CtiOs Object."
GetPropertyName	See Chapter 7, "CtiOs Object."
GetPropertyType	See Chapter 7, "CtiOs Object."
GetSkillGroups	Returns an array of SkillGroups objects
GetValue	See Chapter 7, "CtiOs Object."
GetValueArray	See Chapter 7, "CtiOs Object."
GetValueInt	See Chapter 7, "CtiOs Object."
GetValueString	See Chapter 7, "CtiOs Object."
IsAgent	Checks the current mode and returns true if agent mode.
IsSupervisor	Checks the current mode and returns true if supervisor mode.
IsValid	See Chapter 7, "CtiOs Object."
Login	Logs an agent into the ACD.
Logout	Logs an agent out of the ACD.
MakeCall	Initiates a call to a device or agent.
MakeEmergencyCall	Lets an agent makes an emergency call to the supervisor.
QueryAgentState	Gets the current agent state from CTI Server and retrieves it.
ReportBadCallLine	Informs the CTI OS Server of a bad line.
RequestAgentTeamList	Retrieves the current agent team list.
RequestSupervisorAssist	Allows the agent to call an available supervisor for assistance.
SendChatMessage	Send asynchronous messages between CTI clients.

Method	Description
SetAgentState	Requests a new agent state.
SetValue	Sets the value of the property whose name is specified.
StartMonitoringAgent	Enables monitoring of a specified agent.
StartMonitoringAgentTeam	Enables monitoring of a specified agent team.
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:

  ```
  Answer([reserved_args As IArguments]) As Long
  ```

  as follows:

  ```
  Answer()
  ```

- To ignore these parameters in COM you must send a NULL, as shown:

  ```
  Answer (NULL)
  ```

DisableAgentStatistics

The DisableAgentStatistics method is sent by an agent to request that real-time statistics stop being sent to that agent.

Syntax

C++:`int DisableAgentStatistics (Arguments& reserved_args)`
COM:`HRESULT DisableAgentStatistics (/*[in]*/  IArguments reserved_args, /* [out, retval]*/ int * errorcode)`
VB:`DisableAgentStatistics (reserved_args As CTIOSCLIENTLib.IArguments) As Long`
Java:`int DisableAgentStatistics (Arguments reservedargs)`
.NET:`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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

DisableSkillGroupStatistics

The DisableSkillGroupStatistics method is sent by an agent to request that real-time statistics stop being sent to that agent.

Syntax

```
C++:int DisableSkillGroupStatistics (Arguments& optional_args)
COM:HRESULT DisableSkillGroupStatistics (/* [in, optional]*/ IArguments * optional_args,
/* [out, retval]*/ int * errorcode)
VB:DisableSkillGroupStatistics (optional_args As CTIOSCLIENTLib.IArguments) As Long
Java:int DisableSkillGroupStatistics (Arguments optional_args
.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, each number has a string key of an integer starting with "1" and an integer value that is a SkillGroupNumber to be disabled. If the parameter is NULL or missing, statistics will be disabled for all skill groups to which the agent belongs.

errorcode

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

Return Value

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

EnableAgentStatistics

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

Syntax

C++: `int EnableAgentStatistics(Arguments& reserved_args)`
COM: `HRESULT EnableAgentStatistics (/*[in]*/ IArguments* reserved_args, /* [out, retval]*/ int * errorcode)`
VB: `EnableAgentStatistics (reserved_args As CTIOSCLIENTLib.IArguments) As Long`
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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

Remarks

The CTI OS server sends agent statistics in an OnAgentStatistics event. See theOnAgentStatistics section in Chapter 6, "Event Interfaces and Events"for an explanation of the PollingIntervalSec and PollForAgentStatsAtEndCall registry settings and how these settings affect the refresh rate of agent statistics.

EnableSkillGroupStatistics

The EnableSkillGroupStatistics method is sent by an agent to request that real-time statistics be sent to that agent. If the argument array is empty, then statistics for all skillgroups 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++:`int EnableSkillGroupStatistics (Arguments& optional_args)`
COM: `HRESULT EnableSkillGroupStatistics (/*[in]*/ IArguments * optional_args, /* [out, retval]*/ int * errorcode)`

VB: `EnableSkillGroupStatistics (optional_args As CTIOSCLIENTLib.IArguments) As Long`
Java:`int EnableSkillGroupStatistics(Arguments optional_args)`
.NET:`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. If the parameter is NULL or missing, statistics will be enabled for all skill groups to which the agent belongs.

.NET:args

Refer to the description for optional_args above.

errorcode

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

Return Value

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

Remarks

The CTI OS server sends 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 6-56.

Return Value

For C++, VB, Java, and .NET, this method returns the current state of the agent.

GetAllProperties

See Chapter 7, "CtiOs Object" for a description of the GetAllProperties method.

GetElement

See Chapter 7, "CtiOs Object" for a description of the GetElement method.

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

See Chapter 7, "CtiOs Object" for a description of the GetNumProperties method.

GetPropertyName

See Chapter 7, "CtiOs Object" for a description of the GetNumProperties method.

GetPropertyType

See Chapter 7, "CtiOs Object" for a description of the GetNumProperties method.

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 without enabling skill group statistics, 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>\CTIOS1\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:

```
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 Object Id of the skill group object and the value is a reference to a CILRefArg object instance that contains the actual reference to a CSkillGroup object. To retrieve a reference to a skill group object, you need to do something similar to what is shown in the following code example.

```
Arguments & arSkills = m_Agent->GetSkillGroups();

if(Arguments::IsValidReference(arSkills)){
     for(int nI = 1; nI <= arSkills.NumElements(); nI ++){
         string strUOID = arSkills.GetElementKey(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.

```
HRESULT hr = S_OK;
VARIANT varSkills;

VariantInit(&varSkills)

hr  = m_Agent->GetSkillGroups(&varSkills);

if(SUCCEDED(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(SUCCEDED(hr)){
                  int nSkillGrpNumber = 0;
                   VARIANT vPropKey;
                   VariantInit(&vPropKey);
                   vPropKey.vt = VT_BSTR;
                   vPropKey.bstr =  OLESTR("SkillGroupNumber");
                   pSkill->GetValueInt(vPropKey,&nSkillGrpNumber);
                  pSkill->Release();
                  VariantClear(&vPropKey);
             }
         }
     }
}
```

VB

In VB, the GetSkillGroups method returns a variant array where each element is a reference to a CTIOSClientLib.SkillGroup object.

To retrieve references to skill group objects you need to do something similar to what is shown in the following code example.

```
Dim  obSkill As CTIOSClientLib.SkillGroup
Dim arSkills As Variant
              Dim lNumElements as Long

              arSkills = m_Agent.GetSkillGroups()
              lNumElements = UBound(arSkills,1)
             For nI = 0 to lNumElements
                  Set obSkill = arSkills(nI)
                  Print "SkillGroup" & obSkill.GetValueString(CStr("UniqueObjectId")) &  _
                            "Skill Group Number: " &
obSkill.GetValueInt(CStr("SkillGroupNumber"))
            Next
            End For
```

GetValue Methods

See Chapter 7, "CtiOs Object." for descriptions of the GetValue, GetValueInt, GetValueArray, and GetValueString methods.

IsAgent

The IsAgent method determines whether the AgentMode connection is for an agent rather than a supervisor.

Syntax

C++: `bool IsAgent()`
COM: `HRESULT IsAgent (VARIANT_BOOL *bIsAgent)`
VB: `IsAgent () As Boolean`
Java: `boolean IsAgent()`
.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 into an ACD are AgentID and AgentInstrument. Often, based on customer configuration, the minimum requirements include an ACD password (AgentPassword). Some switches require PositionID in place of (or in addition to) AgentInstrument. Optional arguments include Extension or AgentWorkMode.

To sign on a mobile agent, the following parameters must be set:

- 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 listed in Table 9-4.

Table 9-4 Login Parameters

Keyword	Type	Description
AgentID (required)**	STRING*	The agent's login ID.
AgentInstrument	STRING*	The agent's instrument number.
LoginName (required)**	STRING	The agent's login name.
PositionID	STRING*	Required for Alcatel only.
AgentExtension	STRING*	The agent's teleset extension. Optional if AgentInstrument is provided.
AgentPassword (optional)	STRING*	The agent's password.

Keyword	Type	Description
AgentWorkMode (optional)	INT	A value representing the desired work mode of the agent. Used by Avaya Communications Manager (ACM) ECS with default value of ManualIn.
NumSkillGroups (optional)	INT	The number of Skill Groups that the agent is currently associated with, up to a maximum of 20.
PeripheralID (optional)	INT	The Unified ICM Peripheral ID of the ACD the agent is attached to.
SkillGroupNumber (optional)	INT	The number of an agent skill group associated with the agent.
SkillGroupPriority (optional)	INT	The priority of an agent skill group associated with the agent.
Agent CallMode	INT	A value that indicates the agent's call mode. Valid values are call-by-call (3) and nailed-up (4).
AgentRemote Number	STRING	The phone number that the agent uses for remote login.
RemoteLogin	INT	A value that indicates the agent is configured for remote login as a remote agent.

*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 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

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

Remarks

If the Login request is successful, it returns a CIL_OK CtiOs_Enums.CilError code In addition, the requesting client should expect an AgentStateChange event if the request is successful with an Arguments member with keyword “AgentState” and value of the agent’s current state. (See GetAgentState for possible values.)

If the Login request is unsuccessful, the client will receive an OnControlFailureConf event and the request will return one of the following CtiOs_Enums.CilError codes:

- E_CTIOS_INVALID_SESSION -- either the agent is not associated with the session or the session is not connected.

- E_CTIOS_INVALID_ARGUMENT -- null or invalid arguments were provided.
- E_CTIOS_LOGIN_INCONSISTENT_ARGUMENTS -- Login request argument values for AgentId and/or PeripheralID do not match the values that were set by SetAgent() prior to the Login request.

Logout

The Logout method logs the agent out of the ACD. If the ACD configuration requires or supports other parameters, these can be passed in as logout parameters. Examples are AgentPassword (required by Alcatel for Logout) or logout reason codes (supported on ACM ECS, Unified CCE).

Syntax

```
C++:   int Logout (Arguments& args)
COM:   HRESULT Logout (/*[in]*/ IArguments args, /*[out,retval]*/ int * errorcode)
VB:    Logout (args As CTIOSCLIENTLib.IArguments) As Long
Java:  int Logout(Arguments args)
.NET:  CilError Logout(Arguments args)
```

Input Parameters

args

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

Table 9-5 Logout Parameters

Keyword	Type	Description
EventReasonCode	INT	Reason for logging out. Required for Unified CCE, optional for all other switches.
AgentPassword (optional)	STRING*	The agent's password.
NumSkillGroups (optional)	INT	The number of Skill Groups that the agent is currently associated with, up to a maximum of 20.
SkillGroupNumber (optional)	INT	The number of an agent skill group associated with the agent.
SkillGroupPriority (optional)	INT	The priority of an agent skill group associated with the agent.
AgentID (optional)	STRING*	The agent's login ID.
AgentInstrument	STRING*	The agent's instrument number.
PeripheralID (optional)	INT	The Unified ICM Peripheral ID of the ACD the agent is attached to.

*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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Values

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

Remarks

If the request is successful, the client should receive an OnAgentStateChange event with an Arguments member with keyword "AgentState" and value eLogout. If it is unsuccessful, the client should receive an OnControlFailureConf event. The client should also receive an OnPreLogout event before the OnAgentStateChange event, and an OnPostLogout event afterwards.

MakeCall

The MakeCall method initiates a call to a device or agent. The simplest form of the request requires only a DialedNumber.

Note You can select and make the call against the skillgroup. Do not set the value if the default skillgroup is desired.

Syntax

```
C++:   int MakeCall (Arguments& args)
COM:   HRESULT MakeCall (/*[in]*/ IArguments *args, /*[out,retval]*/ int * errorcode)
VB:    MakeCall (args As CTIOSCLIENTLib.IArguments) As Long
Java:  int MakeCall(Arguments args)
.NET:  CilError MakeCall(Arguments args)
```

Input Parameters

args

Input parameter in the form of an Arguments array that contains the MakeCall parameters listed in Table 9-6.

Table 9-6 MakeCall Parameters

Keyword	Type	Description
DialedNumber (required)	STRING, maximum length 40	The number to be dialed to establish the new call.
PeripheralID (optional)	INT	The Unified ICM Peripheral ID of the ACD the agent is attached to.
AgentInstrument (optional)	STRING*	The agent's instrument number.
CallPlacementType (optional)	STRING, maximum length 40	A value specifying how the call is to be placed identified in Table 9-7.
CallMannerType (optional)	INT	A value specifying additional call processing options identified in Table 9-8.
AlertRings (optional)	INT	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.
CallOption (optional)	INT	A value from Table 9-9 specifying additional peripheral-specific call options.
FacilityType (optional)	INT	A value from Table 9-10 indicating the type of facility to be used.
AnsweringMachine (optional)	INT	A value from Table 9-11 specifying the action to be taken if the call is answered by an answering machine.
Priority (optional)	BOOL	This field should be set to TRUE if the call should receive priority handling.
PostRoute (optional)	BOOL	When this field is set to TRUE, the Post-Routing capabilities of the Unified ICM are to be used to determine the new call destination.
UserToUserInfo (optional)	STRING, maximum length 40	The ISDN user-to-user information.
CallVariable1 (optional)	STRING, maximum length 40	Call variable data that should be set in the new call in place of the corresponding data in the active call.

Keyword	Type	Description
...	...	...
CallVariable10 (optional)		
ECC (optional)	ARGUMENTS	ECC data that should be set in the new call in place of the corresponding data in the active call.
CallWrapupData (optional)	STRING, maximum length 40	Call-related wrapup data.
FacilityCode (optional)	STRING, maximum length 40	A trunk access code, split extension, or other data needed to access the chosen facility.
AuthorizationCode (optional)	STRING, maximum length 40	An authorization code needed to access the resources required to initiate the call.
AccountCode (optional)	STRING, maximum length 40	A cost-accounting or client number used by the peripheral for charge-back purposes.
SkillGroupNumber	INT	The number of an agent skill group associated with the agent.

Table 9-7 Call Placement Types

CallPlacementType	Description	Value
CPT_UNSPECIFIED	Use default call placement.	0
CPT_LINE_CALL	An inside line call.	1
CPT_OUTBOUND	An outbound call.	2
CPT_OUTBOUND_NO_ ACCESS_CODE	An outbound call that will not require an access code.	3
CPT_DIRECT_POSITION	A call placed directly to a specific position.	4
CPT_DIRECT_AGENT	A call placed directly to a specific agent.	5
CPT_SUPERVISOR_ASSIST	A call placed to a supervisor for call handling assistance.	6

*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 9-8 CallManager Type

Table 9-9 Peripheral-specific Call Options

CallOption	Description	Value
COPT_UNSPECIFIED	No call options specified, use defaults.	0
COPT_CALLING_ AGENT_ONLINE	Attempt the call only if the calling agent is “online” (available to interact with the destination party).	1
COPT_CALLING_ AGENT_RESERVED	Attempt the call only if ACDNR on the calling agent’s set is activated (DMS-100).	2
COPT_CALLING_ AGENT_NOT_ RESERVED	Attempt the call only if ACDNR on the calling agent’s set is not activated (DMS-100).	3
COPT_CALLING_ AGENT_BUZZ_BASE	Causes a buzz to be applied to the base of the telephone set as the call is initiated (DMS-100).	4
COPT_CALLING_ AGENT_BEEP_HSET	Causes a tone to be applied to the agent headset as the call is initiated (DMS-100).	5
COPT_SERVICE_ CIRCUIT_ON	Causes a call classifier to be applied to the call (ACM ECS)	6

Table 9-10 Facility Types

FacilityType	Description	Value
FT_UNSPECIFIED	Use default facility type.	0
FT_TRUNK_GROUP	Facility is a trunk group.	1
FT_SKILL_GROUP	Facility is a skill group or split.	2

Table 9-11 Answering Machine Actions

AnsweringMachine	Description	Value
AM_UNSPECIFIED	Use default behavior.	0
AM_CONNECT	Connect call to agent when call is answered by an answering machine.	1
AM_DISCONNECT	Disconnect call when call is answered by an answering machine.	2
AM_NONE	Do not use answering machine detection.	3
AM_NONE_NO_ MODEM	Do not use answering machine detection, but disconnect call if answered by a modem.	4
AM_CONNECT_NO_ MODEM	Connect call when call is answered by an answering machine, disconnect call if answered by a modem.	5

errorcode

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

Return Value

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

Remarks

If the request is successful, the client should receive one or more of the following call related events:

- OnCallBegin
- OnCallDelivered
- OnServiceInitiated
- OnCallOriginated
- OnCallReachedNetwork

If the request is unsuccessful, the client should receive an OnControlFailureConf event.

MakeEmergencyCall

The MakeEmergencyCall method makes an emergency call to the Agent's supervisor.

Syntax

```
C++:    int MakeEmergencyCall ()
        int MakeEmergencyCall (Arguments& reserved_args)
COM:    HRESULT MakeEmergencyCall (/*[in, optional]*/  IArguments reserved_args, /* [out,
retval]*/ int * errorcode)
VB:     MakeEmergencyCall () As Long
    MakeEmergencyCall (reserved_args As CTIOSCLIENTLib.IArguments) As Long
Java:   int MakeEmergencyCall (Arguments args)
.NET:   CilError MakeEmergencyCall(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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

Remarks

The MakeEmergencyCall request is very similar to the RequestSupervisorAssist request in the following two ways:

- Both requests place a call from the requesting agent to a supervisor and are routed employing the same script. A typical script might attempt to route the call to the primary supervisor first (if logged in and in available state) and, failing that, to route the call to a skillgroup that all supervisors belong to.
- Both call requests can be configured through Unified ICM Agent Desk Settings to be performed via a single step conference or consult call. If the consult method is chosen, the agent can complete the established consult call as a transfer or conference.

However, these two requests have the following important differences:

- Only Emergency calls are able to be recorded, if so configured in the 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 MakeEmercenyCall method for higher priority calls than calls made with the

RequestSupervisorAssist method. For example, agents can be trained to click the Emergency button if the customer has more than $1,000,000 in an account, and otherwise to click the Supervisor Assist button. The Supervisor will be able to differentiate the agent's request by noting the CallType.

The MakeEmergencyCall 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 client issuing the request receives an OnEmergencyCall event when the request reaches an available supervisor. If the request is unsuccessful the client receives an OnControlFailureConf event.

QueryAgentState

The QueryAgentState method lets a client retrieve the current state of the agent.

Syntax

C++: `int QueryAgentState (Arguments & args );`
COM: `HRESULT QueryAgentState ( /*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode );`
VB: `QueryAgentState (ByVal args as CTIOSCLIENTLIB.IArguments) As Long`
Java: `int QueryAgentState (Arguments args)`
.NET: `CilError QueryAgentState(Arguments args)`

Input Parameters

args

Arguments array that contains the parameters listed in the following table.

Table 9-12 QueryAgentState Parameters

Keyword	Type	Description
Agent ID	STRING	Agent's login ID.
AgentInstrument	STRING	Agent's instrument number.

Return Values

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

Remarks

If the request is successful, the client should receive an OnQueryAgentStateConf event. If it is unsuccessful, the client should receive an OnControlFailureConf event.

ReportBadCallLine

The ReportBadCallLine method informs the CTI OS server of the poor quality of the agent's line. A note of this is recorded in the database.

Syntax

C++:
```
int ReportBadCallLine ()
int ReportBadCallLine (Arguments& reserved_args)
```
COM:
```
HRESULT ReportBadCallLine (/*[in, optional]*/  IArguments reserved_args, /* [out,
retval]*/ int * errorcode)
```
VB:
```
ReportBadCallLine () As Long
```
Java:
```
int ReportBadCallLine (Arguments args)
```
.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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Values

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

RequestAgentTeamList

The RequestAgentTeamList method is called by a supervisor to make a request to the CTI OS server for a list of agents in the supervisor's team.

Syntax

C++:
```
int RequestAgentTeamList ()
int RequestAgentTeamList (Arguments& reserved_args)
```
COM:
```
HRESULT RequestAgentTeamList (/*[in, optional]*/  IArguments reserved_args, /*
[out, retval]*/ int * errorcode)
```
VB:
```
RequestAgentTeamList () As Long
```
Java:
```
int RequestAgentTeamList ()
int RequestAgentTeamList (Arguments args)
```
.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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Values

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

Remarks

Supported for use with Unified CCE only. See "MakeEmergencyCall" for more information.

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++: `int SendChatMessage (Arguments& args)`
COM: `HRESULT SendChatMessage (/*[in]*/ IArguments *args, /*[out,retval]*/ int * errorcode)`
VB: `SendChatMessage (args As CTIOSCLIENTLib.IArguments) As Long)`
Java: `int SendChatMessage(Arguments args)`
.NET: `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 9-13 SendChatMessage Parameters

Keyword	Type	Description
Distribution (required)	STRING	Currently the only supported value is “agent”.
Target (optional)	STRING	When the Distribution is set to DistributeToAgent, this field must be included with the AgentID of the intended recipient. When the LoginName is set to the LoginName of the agent to receive the chat message, this field must also be set to the login name of the agent to which to chat.
Message (optional)	STRING	The text of the user message. Maximum message size is 255 bytes.
LoginName (optional)	STRING	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.

errorcode

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

Return Values

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

Remarks

The recipient receives the message via the OnChatMessage event.

SetAgentState

The SetAgentState method requests a new agent state. Login and Logout are valid agent states and can be set using the SetAgentState method as well as 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 9-14 SetAgentState Parameters

Keyword	Type	Description
AgentState (required)	INT	The state to which to set the specified agent. The value of this field must be one of the values in Table 6-56.
AgentID (required)	STRING*	The agent's login ID.
AgentInstrument	STRING*	The agent's instrument number. Optional if Agent Extension is provided.
PositionID	STRING*	Required for Alcatel only.
AgentPassword (optional)	STRING*	The agent's password.
AgentWorkMode (optional)	INT	A value representing the desired work mode of the agent. Used by ACM ECS with default value of ManualIn.
NumSkillGroups (optional)	INT	The number of Skill Groups that the agent is currently associated with, up to a maximum of 20.
EventReasonCode (optional)	INT	Reason for logging out. Required for Unified CCE, optional for all other switches.

Keyword	Type	Description
PeripheralID (optional)	INT	The Unified ICM Peripheral ID of the ACD the agent is attached to.
SkillGroupNumber (optional)	INT	The number of an agent skill group associated with the agent.
SkillGroupPriority (optional)	INT	The priority of an agent skill group associated with the agent.

*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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Values

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

Remarks

A successful request will result in an OnAgentStateChanged event. It may also result in OnPreLogout, OnPostLogout, and/or OnLogoutFailed events. If this request is unsuccessful, the client receives an OnControlFailureConf event.

StartMonitoringAgent

The StartMonitoringAgent method allows the client, which must be a supervisor, to start monitoring the specified Agent object. This call will cause the supervisor to receive all of the monitored call events (See "IMonitoredCallEvents Interface" in Chapter 6, "Event Interfaces and Events") for this agent until the supervisor calls StopMonitoringAgent.

Syntax

C++: `int StartMonitoringAgent(Arguments& args)`
COM: `HRESULT StartMonitoringAgent (/*[in]*/ IArguments * args, /*[out,retval]*/ int * errorcode)`
VB: `StartMonitoringAgent (args As CTIOSCLIENTLib.IArguments) As Long`
Java: `int StartMonitoringAgent(Arguments args)`
.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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, 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 OnMonitorAgentStateChange events for every agent on the specified team.

Syntax

```
C++:   int StartMonitoringAgentTeam (Arguments& args)
COM:   HRESULT StartMonitoringAgentTeam (/*[in]*/ IArguments args, /*[out,retval]*/ int *
errorcode)
VB:    StartMonitoringAgentTeam (args as CTIOSCLIENTLib.IArguments) As Long
Java:  int StartMonitoringAgentTeam (Arguments args)
.NET:  CilError StartMonitoringAgentTeam(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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Unified CCE only).

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 will cause the supervisor to receive monitored agent events for all of the agents in the supervisor's team (see"IMonitoredAgentEvents Interface" in Chapter 6, "Event Interfaces and Events").

Syntax

C++: `int StartMonitoringAllAgentTeams (Arguments& reserved_args)`
COM: `HRESULT StartMonitoringAllAgentTeams (/*[in, optional]*/ IArguments reserved_args, /*[out,retval]*/ int * errorcode)`
VB: `StartMonitoringAllAgentTeams ([reserved_args as CTIOSCLIENTLib.IArguments]) As Long`
Java: `int StartMonitoringAllAgentTeams (Arguments args)`
.NET: `CilError StartMonitoringAllAgentTeams(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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, 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, to clear the value of the currently monitored call, call this method with an empty args parameter.

Syntax

```
C++:   int StartMonitoringCall(Arguments& args)
COM:   HRESULT StartMonitoringCall (/*[in]*/ IArguments * args, /*[out,retval]*/ int *
errorcode)
VB:    StartMonitoringCall (args As CTIOSCLIENTLib.IArguments) As Long
Java:  int StartMonitoringCall(Arguments args)
.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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Unified CCE only).

StopMonitoringAgent

The StopMonitoringAgent method allows the client, which must be a supervisor, to stop monitoring the specified Agent object. This will stop all Monitored Call events from being sent to the supervisor.

Syntax

```
C++:   int StopMonitoringAgent(Arguments& args)
COM:   HRESULT StopMonitoringAgent (/*[in]*/ IArguments * args, /*[out,retval]*/ int *
errorcode)
VB:    StopMonitoringAgent (args As CTIOSCLIENTLib.IArguments) As Long
Java:  int StopMonitoringAgent(Arguments args)
.NET:  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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Unified CCE only).

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++:int StopMonitoringAgentTeam (Arguments& args)
COM:  HRESULT StopMonitoringAgentTeam (/*[in]*/ IArguments args, /*[out,retval]*/ int *
errorcode)
VB: StopMonitoringAgentTeam (args as CTIOSCLIENTLib.IArguments) As Long
Java:int StopMonitoringAgentTeam(Arguments args)
.NET: 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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Unified CCE only).

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++: `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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Unified CCE only).

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. Refer to "Enabling Unified CM Based Silent Monitoring in your Application" on page 4-57 for additional Information.

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). These should be packaged with the keywords "AgentReference" and "CallReference" respectively.

The third required parameter is one of the following integers representing the desired supervisory action.

Table 9-15 SuperviseCall Parameters

Value	Enum	Description
3	eSupervisorBargeIn	BargeIn to the specified call of the specified agent.
4	eSupervisorIntercept	Intercept the specified call of the specified agent.
1	eSupervisorMonitor	Used to silently monitor the call of the specified agent.
0	eSupervisorClear	Used to clear the silent monitor call.

Note 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".

Return Values

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

Remarks

This request is specific to the Supervisor feature and should only be used on switches or configurations that have the necessary support (currently, Unified CCE only).

A BargeIn action is very similar to a Single Step Conference where the agent is the conference controller. As such, only this agent is able to add other parties to the conference; the supervisor will not be able to do this.

An Intercept can only be performed by a supervisor who has already performed a BargeIn. The Intercept simply hangs up the original agent, leaving only the customer and the supervisor talking.

E_CTIOS_INVALID_SILENT_MONITOR_MODE is returned when Agent.SuperviseCall() is called when CTI OS Based silent monitor is configured.

CHAPTER 10

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 would be two distinct CIL Call objects.

The object stores specific call information as properties, including the ICMEnterpriseUniqueID, ANI, DNIS, Call variables, and ExpandedCallContext variables. The Call object is created in response to call events received at the CIL. The Call object's properties and state will be updated throughout the lifetime of the call connection.

See Chapter 3, "CIL Coding Conventions" for an explanation of accessing Call and ECC variables via the *GetValue* mechanism.

Current Call Concept

The Client Interface Library uses the concept of a Current Call. The Current Call concept is used by the CTI OS Toolkit as a way for the controls and the application to communicate with each other regarding which call is currently selected and should be the one to act upon. For example, if an agent has a call and receives a new Ringing call, he might select the Talking call on the grid. At this click, CallAppearanceMgr control calls SetCurrentCall() to make this call the Current Call. When the agent clicks the Hold control, this control would call GetCurrentCall() to obtain a call pointer through which to call the Hold() method. The agent might then select the Ringing call, which would again cause the CallAppearanceMgr control to call SetCurrentCall() to make this new call the current call. Then, when the agent clicks the Answer control, this control would again call GetCurrentCall() to obtain a call pointer through which to call the Answer() method.

If your application uses Cisco's out-of-the-box button controls (see Chapter 5, "CTI OS ActiveX Controls"), but not the CallAppearanceMgr grid control, you will need to use SetCurrentCall() and GetCurrentCall() in order for the button controls to enable and disable correctly when switching between multiple calls.

Note The CurrentCall concept does not place any limitations on call control of non-current calls. All of the call behaviors implemented by method calls on the Call object will work on any call object that is available at the CIL, even if it is not the CurrentCall.

Accessing ECC Variables

The 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. ECC variables are defined in the Unified ICM Configuration Manager, and are sent between Unified ICM servers as a key-value structure. The mechanism for accessing ECC variables from CTI OS is similar to accessing all other call variables.

To simplify the organization of properties on the Call object, the ECC variables are stored in their own Arguments structure which is nested in the Call object Arguments structure.

Considerations for 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.

Retrieving ECC Variable Values

To retrieve an ECC variable from the Call object, first retrieve the ECC (Arguments) structure from the Call object using GetValueArray with keyword ECC. Then, retrieve the specific ECC variable required by using its name as the keyword to GetValueInt, GetValueArray, or GetValueString, depending on its type. The following is some sample code for C++ without COM:

```
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 would be 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]");
    }
}
```

Adding ECC Values

If you want to add ECC values to a call without deleting ones that are already set in the call, retrieve the ECC variables and then add the new ones as shown in C++ without COM:

```
    Arguments & RequestArgs = Arguments::CreateInstance();
    Arguments * pECCData = NULL;

    // presumes that we have a Call object pointer in pCall
    if (pCall->IsValid (CTIOS_ECC))
        pCall->GetValueArray(CTIOS_ECC, &pECCData);

    else
        Arguments::CreateInstance(&pECCData);


    pECCData->AddItem("user.MyECC", "FirstECCVariable");
    pECCData->AddItem("user.MyArray[2]", 2222);
    RequestArgs.AddItem(CTIOS_ECC, *pECCData);
    pCall->SetCallData(RequestArgs);

    RequestArgs.Release();
    pECCData->Release();
```

The same thing in VB would be 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 would be 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

Table 10-1 lists the available call object properties.

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

Table 10-1 Call Object Properties

Keyword	Type	Description
ANI	STRING	The calling line ID of the caller.
CallerEnteredDigits	STRING	The digits entered by the caller in response to IVR prompting.
CallStatus	SHORT	The current status of the call.
CallType	SHORT	The general classification of the call type.
CallVariable1	STRING	Call-related variable data.
CallVariable2	STRING	Call-related variable data.
CallVariable3	STRING	Call-related variable data.
CallVariable4	STRING	Call-related variable data.
CallVariable5	STRING	Call-related variable data.
CallVariable6	STRING	Call-related variable data.
CallVariable7	STRING	Call-related variable data.
CallVariable8	STRING	Call-related variable data.
CallVariable9	STRING	Call-related variable data.
CallVariable10	STRING	Call-related variable data.
CallWrapupData	STRING	Call-related variable data.
ClassIdentifier	INT	Private; for internal use only.
DialedNumber	STRING	The number dialed.
DNIS	STRING	The DNIS provided with the call.
ECC	ARGUMENTS	Arguments structure of key-value pairs of ECC variables.
ICMEnterpriseUniqueID	STRING	Required only when the call is pre-routed.
LineType	SHORT	Indicates the type of the teleset line.
MeasuredCallQTime	INT	Number of seconds this call was in a local queue before being delivered to the agent.
PeripheralID	INT	The Unified ICM PeripheralID of the ACD where the call activity occurred.
RouterCallKeyCallID	INT	The call key created by the Unified ICM. The Unified ICM resets this counter at midnight.

Table 10-1 Call Object Properties (continued)

Keyword	Type	Description
Router CallKeyDay	INT	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.
ServiceID	INT	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.
ServiceNumber	INT	The service that the call is attributed to, as known to the peripheral. May contain the special value NULL_SERVICE when not applicable or not available.
SkillGroupID	INT	The Unified ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_ GROUP when not applicable or not available.
SkillGroupNumber	INT	The number of the agent SkillGroup the call is attributed to, as known to the peripheral. May contain the special value NULL_ SKILL_GROUP when not applicable or not available.
UniqueObjectID	STRING	An object ID that uniquely identifies the call object.
UserToUserInfo	STRING	The ISDN user-to-user information element.

Methods

Table 10-2 lists the available call object methods.

Table 10-2 Call Object Methods

Method	Description
Alternate	Places the current call on hold and retrieves a previously held call.
Answer	Answers a call that is in the alerting or ringing state.

Table 10-2 Call Object Methods (continued)

Method	Description
Clear	Clears a call, dropping all parties to the call.
ClearConnection	Hangs up a call, leaving other parties in a conference call. If there are only two parties on the call it clears the call.
Conference	Either establishes a three party conference call or adds a new party to an existing conference call.
DumpProperties	See Chapter 7, "CtiOs Object."
GetAllProperties	See Chapter 7, "CtiOs Object."
GetCallContext	Gets data associated with the call other than call and expanded call context (ECC) variables.
GetCallData	Obtains call and expanded call context (ECC) variables.
GetElement	See Chapter 7, "CtiOs Object."
GetLastError (.NET only)	Returns the last error that occurred on the calling thread.
GetNumProperties	See Chapter 7, "CtiOs Object."
GetPropertyName	See Chapter 7, "CtiOs Object."
GetPropertyType	See Chapter 7, "CtiOs Object."
GetValue methods	Retrieve a property from the Call object based on the property's name key.
Hold	Places a current call on hold.
IsValid	See Chapter 7, "CtiOs Object."
MakeConsultCall	Places a current call on hold and makes a new call.
Reconnect	Clears the current call and then retrieves a held call.
Retrieve	Retrieves a held call.
SetCallData	Sets call and expanded call context (ECC) variables.
SendDTMFSignal	Requests the ACD to send a sequence of DTMF tones.
SingleStepConference	Performs a single step conference.
SingleStepTransfer	Performs a single step transfer.
Snapshot	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.
StartRecord	Starts recording of a call.
StopRecord	Stops recording of a call.
Transfer	Transfers a call to a third party.

Arguments Parameters

The following rules apply to the optional_args and reserved_args parameters in Call Object methods:

- In VB, you can ignore these parameters altogether. For example, you can treat the line:

  ```
  Answer([reserved_args As IArguments]) As Long
  ```

  as follows:

  ```
  Answer()
  ```

- To ignore these parameters in COM you must send a NULL, as shown:

  ```
  Answer (NULL)
  ```

Alternate

The Alternate method combines the action of placing a talking call on hold and then retrieving a previously held call at the same device. If there are only two calls at the device, this method may be called via either the current or the held call.

Syntax

```
C++:   int Alternate()
       int Alternate(Arguments & reserved_args);
COM:   HRESULT Alternate (/*[in,optional]*/  IArguments *reserved_args,    (/*[out,
retval]*/ int * errorcode );
VB:    Alternate([reserved_args As IArguments]) As Long
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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Values

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

Remarks

For switches that allow more than two calls at a device (for example G3), it is recommended that this request only be made through the desired held call, because of the ambiguity caused by multiple held calls at the device.

The Alternate request must be made via a call whose status is either LCS_CONNECT or LCS_HELD or it will fail.

The following events will be received if this request is successful.

For the call making the Alternate request:

- OnAlternateCallConf event

For the originally current call:

- OnCallHeld event

For the originally held call:

- OnCallRetrieved event

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

- OnControlFailureConf event

Answer

The Answer method answers a call that is in the alerting or ringing state (i.e., call status of LCS_ALERTING).

Syntax

```
C++: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 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

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

Remarks

A call may be answered after the OnCallDelivered event has been received. The Answer request must be made via a call whose call status LCS_ALERTING or it will fail.

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

- OnAnswerCallConf event
- OnCallEstablished event

The following events will be received if this request fails:

- OnControlFailureConf event

Clear

The Clear method clears the call and drops all parties to the call.

Syntax

C++:`int Clear()`
```
    int Clear(Arguments & reserved_args);
```
COM:`HRESULT Clear (/*[in,optional]*/  IArguments *reserved_args,    (/*[out, retval]*/ int * 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 3-2 in Chapter 3, "CIL Coding Conventions.".

Return Value

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

Remarks

In the case of a multi-party Conference call, calling Clear() will result in all of the parties to the call being hung up. (If this is not the desired behavior, see the ClearConnection method.) Under certain switches the Clear request will be made via a call whose status is LCS_CONNECT or LCS_INITIATE or it will fail. Many other switches will allow the Clear method to be called via a call whose status is LCS_ALERTING or LCS_HOLD. It may never be made via a call whose status is LCS_NULL indicating that it has already been cleared.

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

- OnClearCallConf event
- OnCallCleared event

The following events will be received if this request fails:

- OnControlFailureConf event

Note The Clear method is not supported on Unified CCE. Use of the Clear method with Unified CCE will result in loss of third-party call control. To avoid this error, applications should use the ClearConnection method instead of Clear to hang up a call.

ClearConnection

The ClearConnection method clears a single connection from a call. If there are only two parties to the call, this effectively clears the call, however for a multi-party conference call, only the one connection is dropped.

Syntax

C++:`int ClearConnection()`
`int ClearConnection(Arguments & reserved_args);`
COM:`HRESULT ClearConnection (/*[in,optional]*/ IArguments *reserved_args, (/*[out, retval]*/ int * errorcode)`
VB: `ClearConnection([reserved_args As IArguments]) As Long`
Java: `int ClearConnection(Arguments rArgs);`
.NET: `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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

Remarks

As with the Clear method, under certain switches the ClearConnection request must be made via a call whose status is LCS_CONNECT or LCS_INITIATE or it will fail. Many other switches allow the Clear method to be called via a call whose status is LCS_ALERTING or LCS_HOLD. It may never be made via a call whose status is LCS_NULL indicating that it has already been cleared.

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

- OnClearConnectionConf event
- OnCallConnectionlCleared event

If this is a two party call, these events will be followed by

- OnCallCleared event

The following events will be received if this request fails:

- OnControlFailureConf event

Conference

The Conference method either begins a new conference call or adds an additional call to an existing conference call. When it begins a new conference call, it combines an original two-party call with a two-party consult call (where the two calls have a common party) into a single three party call. Only the common party (which is called the "Conference Controller") can call this method to make the new conference call. This method may be called on either of the Conference Controller's calls.

Syntax

C++:`int Conference();`
`int Conference(Arguments& optional_args)`
COM:`HRESULT Conference ( /*[in, optional]*/ IArguments *optional_args, (/*[out, retval]*/ int * errorcode )`
VB: `Conference([optional_args As IArguments]) As Long`
Java:`int Conference(Arguments optional_args)`
.NET:`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 should be conferenced. If this argument is used, it should be added to the Arguments parameter with the keyword of "CallReferenceObjectID". This would only be necessary in an environment where there are multiple held calls and the request is being made through the talking call. If the request is being made through a specific held call in this scenario, or if there are only two calls at the device, this parameter is unnecessary.

errorcode

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

Return Value

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

Remarks

Before making this request, it is necessary for the original call to be in the held state and the consult call to be in the talking state or the request will fail. Therefore, if the calls are alternated (see Alternate), they must be alternated again to return the two calls to their appropriate states.

If there are only two calls at the device, this method may be called 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.

The Conference request must be made via a call whose call status is LCS_CONNECT or LCS_HELD or it will fail.

On certain switches (notably Unified CCE), only the Conference Controller (the party that first initiated the conference call) may add additional parties to an existing conference call.

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

- OnConferenceCallConf event
- OnCallConferenced event

The following events will be received if this request fails:

- OnControlFailureConf event

GetCallContext

The GetCallContext method returns an Arguments array containing the values for call properties other than CallVariables and ECC Variables, such as ANI, DNIS, and the other properties listed in Table 10-3.

Syntax

```
C++:   int GetCallContext(Arguments& args)
COM:   HRESULT GetCallContext (/*[out,retval]*/ IArguments ** args)
VB:    GetCallContext (CTIOSCLIENTLib.IArguments args)
Java:  Arguments GetCallContext()
.NET:  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 Table 10-3 that are present in the call.

Return Value

C++, COM, and VB: Default HRESULT return values. See Chapter 3, "CIL Coding Conventions."

Java/.NET: A reference to an Arguments array that, on return, holds name/value pairs from Table 10-3. Any of these parameters included may be accessed from the Arguments array using the associated keyword.

Table 10-3 GetCallContext Arguments Array Contents

Keyword	Type	Description
ANI	STRING	The calling line ID of the caller.
CallerEnteredDigits	STRING	The digits entered by the caller in response to IVR prompting.
CallType	SHORT	The general classification of the call type.
CallWrapupData	STRING	Call-related wrapup data.
ConnectionCallID	UINT	The Call ID value assigned to this call by the peripheral or the Unified ICM.
DialedNumber	STRING	The number dialed.
DNIS	STRING	The DNIS provided with the call.

Table 10-3 GetCallContext Arguments Array Contents (continued)

Keyword	Type	Description
ICMEnterpriseUniqueID	STRING	A unique identifier for this contact throughout the enterprise. This can track a single customer contact across multiple sites, e.g., when a call is transferred between agents.
ServiceID	INT	The Unified ICM identifier for the Service to which this call was routed.
ServiceNumber	INT	The ACD number of the Service to which this call was routed.
SkillGroupID	INT	The Unified ICM identifier for the SkillGroup to which this call was routed.
SkillGroupNumber	INT	The number of the SkillGroup at the ACD to which this call was routed.
UniqueObjectID	STRING	A unique object ID for the call.
UserToUserInfo	STRING	The ISDN user-to-user information element.

Remarks

This is simply a convenience method to be called to get all of a call's non-CallVariable data at one time. If only certain data members are desired, call the appropriate GetValue method for each instead.

GetCallData

The GetCallData method returns the values ofCallVariable1 through CallVariable10 and all of the ECC (Extended CallContext) variables.

Syntax

C++: `int GetCallData(Arguments& args)`
COM: `HRESULT GetCallData (/*[out,retval]*/ IArguments ** args)`
VB: `GetCallData (CTIOSCLIENTLib.IArguments args)`
Java: `Arguments GetCallData()`
.NET: `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. See Chapter 3, "CIL Coding Conventions."

Java/.NET: A reference to an Arguments array that, on return, holds parameters described under Remarks.

Remarks

This is simply a convenience method to be called to get all of a call's CallVariables (1 through 10) and ECC Call Variables at one time. If only certain call variables are desired, call the appropriate GetValue method for each instead.

Access the data in the following way:

- To access the values for individual CallVariables from the arguments parameter, use GetValueString with either the keywords of "CallVariable1" through "CallVariable10".

To access ECC call data, use the following procedure:

- First, get the ECC variables as a whole from the arguments parameter, using GetValueArray with the keyword "ECC'. This will return another Arguments array that is nested in the Arguments array returned from GetCallData.
- To access an individual ECC scalar variable from this Arguments array, use the appropriate GetValueString, GetValueInt, etc. depending on the variable's type, using the string keyword "user.VariableName".
- To access an individual ECC array variable from this Arguments array, use the appropriate GetValueString, GetValueInt, etc. depending on the variable's type, using the string keyword "user.ArrayName[n] where n is a zero based integer that notes the offset in the array.

Hold

The Hold method holds a current call.

Syntax

C++:
```
int Hold()
int Hold(Arguments & reserved_args);
```
COM:
```
HRESULT Hold (/*[in,optional]*/  IArguments *reserved_args,    (/*[out, retval]*/
int * errorcode )
```
VB:
```
Hold([reserved_args As IArguments]) As Long
```
Java:
```
Arguments Hold(Arguments rArgs)
```
NET:
```
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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

Remarks

The Hold request must be made via a call whose call status is LCS_CONNECT or it will fail.

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

- OnHoldCallConf event
- OnCallHeld event

The following events will be received if this request fails:

- OnControlFailureConf event

MakeConsultCall

The MakeConsultCall method initiates the combined action of placing the associated current call on hold and then making a new call. By default, the call context data (including call variables) of the current call is used to initialize the context data of the new consultation call. The application may override some or all of the original call context in the consultation call by providing the desired values in this request.

The simplest form of the request only requires a dialed number and a consult type. The request may also include optional parameters, as listed in Table 10-4.

Syntax

C++:`int MakeConsultCall (Arguments& args))`
COM:`HRESULT MakeConsultCall (/*[in]*/ IArguments *args, /*[out, retval]*/ int * errorcode)`
VB: `MakeConsultCall (args As CTIOSCLIENTLib.IArguments) 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 Table 10-4. Any of these parameters included should be added to the Arguments array using the associated key word.

Table 10-4 MakeConsultCall Parameters

Parameter	Type	Description
DialedNumber (required)	STRING, maximum length 40	Dialed number; the number to be dialed to establish the new call.
ConsultType (required)	INT	A value specifying whether this consult call is in preparation for either a transfer or a conference, as specified in the ConsultType Table.
CallPlacementType (optional)	STRING, maximum length 40	A value specifying how the call is to be placed identified in Table 10-5.
CallMannerType (optional)	INT	A value specifying additional call processing options identified in Table 10-6.
AlertRings (optional)	INT	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.
CallOption (optional)	INT	A value from Table 10-7 specifying additional peripheral-specific call options.
FacilityType (optional)	INT	A value from Table 10-8 indicating the type of facility to be used.
AnsweringMachine (optional)	INT	A value from Table 10-9 specifying the action to be taken if the call is answered by an answering machine.
Priority (optional)	BOOL	This field should be set to TRUE if the call should receive priority handling.
PostRoute (optional)	BOOL	When this field is set to TRUE, the Post-Routing capabilities of the Unified ICM will determine the new call destination.
UserToUserInfo (optional)	STRING, maximum length 40	The ISDN user-to-user information.
CallVariable1 (optional)	STRING, maximum length 40	Call variable data that should be set in the new call in place of the corresponding data in the current call.

Table 10-4 MakeConsultCall Parameters (continued)

Parameter	Type	Description
...	...	...
CallVariable10 (optional)		
ECC	ARGUMENTS	ECC data that should be set in the new call in place of the corresponding data in the current call.
CallWrapupData (optional)	STRING, maximum length 40	Call-related wrapup data.
FacilityCode (optional)	STRING, maximum length 40	A trunk access code, split extension, or other data needed to access the chosen facility.
AuthorizationCode (optional)	STRING, maximum length 40	An authorization code needed to access the resources required to initiate the call.
AccountCode (optional)	STRING, maximum length 40	A cost-accounting or client number used by the peripheral for charge-back purposes.

Table 10-5 CallPlacementType Values

CallPlacementType	Description	Value
CPT_UNSPECIFIED	Use default call placement.	0
CPT_LINE_CALL	An inside line call.	1
CPT_OUTBOUND	An outbound call.	2
CPT_OUTBOUND_NO_ ACCESS_CODE	An outbound call that will not require an access code.	3
CPT_DIRECT_POSITION	A call placed directly to a specific position.	4
CPT_DIRECT_AGENT	A call placed directly to a specific agent.	5
CPT_SUPERVISOR_ASSIST	A call placed to a supervisor for call handling assistance.	6

Table 10-6 CallMannerType Values

CallMannerType	Description	Value
CMT_UNSPECIFIED	Use default call manner.	0
CMT_POLITE	Attempt the call only if the originating device is idle.	1

Table 10-6 CallMannerType Values (continued)

CMT_BELLIGERENT	The call should always be attempted, disconnecting any currently active call.	2
CMT_SEMI_POLITE	Attempt the call only if the originating device is idle or is receiving dial tone.	3

Table 10-7 CallOption Values

CallOption	Description	Value
COPT_UNSPECIFIED	No call options specified, use defaults.	0
COPT_CALLING_ AGENT_ONLINE	Attempt the call only if the calling agent is “online” (available to interact with the destination party).	1
COPT_CALLING_ AGENT_RESERVED	Attempt the call only if ACDNR on the calling agent’s set is activated (DMS-100).	2
COPT_CALLING_ AGENT_NOT_ RESERVED	Attempt the call only if ACDNR on the calling agent’s set is not activated (DMS-100).	3
COPT_CALLING_ AGENT_BUZZ_BASE	Causes a buzz to be applied to the base of the telephone set as the call is initiated (DMS-100).	4
COPT_CALLING_ AGENT_BEEP_HSET	Causes a tone to be applied to the agent headset as the call is initiated (DMS-100).	5
COPT_SERVICE_ CIRCUIT_ON	Causes a call classifier to be applied to the call (ACM ECS)	6

Table 10-8 FacilityType Values

FacilityType	Description	Value
FT_UNSPECIFIED	Use default facility type.	0
FT_TRUNK_GROUP	Facility is a trunk group.	1
FT_SKILL_GROUP	Facility is a skill group or split.	2

Table 10-9 AnsweringMachine Values

AnsweringMachine	Description	Value
AM_UNSPECIFIED	Use default behavior.	0
AM_CONNECT	Connect call to agent when call is answered by an answering machine.	1
AM_DISCONNECT	Disconnect call when call is answered by an answering machine.	2
AM_NONE	Do not use answering machine detection.	3

Table 10-9 AnsweringMachine Values (continued)

AnsweringMachine	Description	Value
AM_NONE_NO_ MODEM	Do not use answering machine detection, but disconnect call if answered by a modem.	4
AM_CONNECT_NO_ MODEM	Connect call when call is answered by an answering machine, disconnect call if answered by a modem.	5

errorcode

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

Return Values

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

Remarks

The MakeConsultCall request must be made via a call whose call status is LCS_CONNECT or it will fail. Calling MakeConsultCall successfully will result in the same events as a successful MakeCall called on the agent.

The following events will be received if this request is successful.

For the call making the MakeConsultCallRequest:

- OnMakeConsultCallConf event
- OnCallHeld event

For the newly created outgoing consult call:

- OnBeginCall event
- OnServiceInitiated event
- OnCallOriginated event
- OnCallDelivered event

For the new connection that is ringing as a result of the consult call:

- OnBeginCall event
- OnCallDelivered event

The following events will be received if this request fails:

- OnControlFailureConf event

Reconnect

The Reconnect method combines the action of releasing a current call and then retrieving a previously held call at the same device. If there are only two calls at the device, this method may be called via either the talking or the held call.

Syntax

C++:
```
int Reconnect()
int Reconnect(Arguments & reserved_args)
```
COM:
```
HRESULT Reconnect (/*[in,optional]*/  IArguments * reserved_args,    (/*[out, retval]*/ int * errorcode )
```
VB:
```
Reconnect([reserved_args As IArguments]) As Long
```
Java:
```
int Reconnect(Arguments rArgs)
```
.NET:
```
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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Values

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

Remarks

For switches which allow more than two calls at a device (for example G3), it is recommended that this request only be made through the desired held call, because of the ambiguity caused by multiple held calls at the device.

The Alternate request must be made via a call whose status is either LCS_CONNECT or LCS_HELD or it will fail.

The following events will be received if this request is successful.

For the call making the Reconnect request:

- OnReconnectCallConf event

For the originally current call:

- OnCallConnectionCleared event
- OnCallCleared event
- OnCallEnd event

For the originally held call:

- OnCallRetrieved event

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

- OnControlFailureConf event

Retrieve

The Retrieve method unholds a held call.

Syntax

```
C++int Retrieve()
   int Retrieve(Arguments & reserved_args)
COM:HRESULT Retrieve (/*[in,optional]*/  IArguments *reserved_args,    (/*[out,
retval]*/ int * errorcode )
VB:Retrieve([reserved_args As IArguments]) As Long
Java:int Retrieve(Arguments rArgs)
.NET: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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Values

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

Remarks

The Retrieve request must be made via a call whose call status is LCS_HELD or it will fail.

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

- OnRetrieveCallConf event
- OnCallRetrieved event

The following events will be received if this request fails:

- OnControlFailureConf event

SendDTMFSignal

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

Syntax

```
C++:int SendDTMFSignal(Arguments& args)
COM:  HRESULT SendDTMFSignal (/*[in]*/ args *arguments, /*[out, retval]*/ int *
errorcode)
VB:    SendDTMFSignal (args As CTIOSCLIENTLib.IArguments, errorcode As Long)
Java:int SendDTMFSignal(Arguments rArgs)
.NET: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. Any of these parameters included should be added to the Arguments array using the associated key word.

Table 10-10 SendDTMFSignal parameters

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

errorcode

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

Return Values

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

Remarks

- The OnSendDTMFSignalConf event will be received if this request succeeds.
- The OnControlFailureConf event will be received if this request fails.

SetCallData

The SetCallData method enables any or all of a call's CallVariables (1 through 10) and ECC data to be set at one time.

Note When writing a custom application, in any language, call variables are not blanked out if it 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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Values

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

Remarks

You *must* specify the data for all elements in the Arguments array, not just those elements that you want to change. Failure to do so will cause the unchanged elements to disappear.

The following events will be sent if this request succeeds:

- OnSetCallDataConf
- OnCallDataUpdate

The OnControlFailureConf event will be 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, he will be 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++:`int SingleStepConference(Arguments& args)`
COM:`HRESULT SingleStepConference (IArguments *args, int * errorcode)`
VB: `SingleStepConference (args As CTIOSCLIENTLib.IArguments, errorcode As Long)`
Java:`int SingleStepConference(Arguments rArgs)`
.NET:`CilError SingleStepConference(Arguments args)`

Parameters

args

An output parameter of either a reference or a pointer to an Arguments array containing parameters from Table 10-11. Any of these parameters included should be added to the Arguments array using the associated keyword.

Table 10-11 SingleStepConference Parameters

Parameter	Type	Description
DialedNumber (required)	STRING, maximum length 40	Dialed number; the number to be dialed to establish the new call.
CallPlacementType (optional)	STRING, maximum length 40	A value specifying how the call is to be placed identified in Table 10-5.
CallMannerType (optional)	INT	A value specifying additional call processing options identified in Table 10-6.
AlertRings (optional)	INT	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.
CallOption (optional)	INT	A value from Table 10-7 specifying additional peripheral-specific call options.
FacilityType (optional)	INT	A value from Table 10-8 indicating the type of facility to be used.
AnsweringMachine (optional)	INT	A value from Table 10-9 specifying the action to be taken if the call is answered by an answering machine.
Priority (optional)	BOOL	This field should be set to TRUE if the call should receive priority handling.
PostRoute (optional)	BOOL	When this field is set to TRUE, the Post-Routing capabilities of the Unified ICM will determine the new call destination.
UserToUserInfo (optional)	STRING, maximum length 40	The ISDN user-to-user information.
CallVariable1 (optional)	STRING, maximum length 40	Call variable data that should be set in the new call in place of the corresponding data in the current call.
…	…	…
CallVariable10 (optional)		
ECC	ARGUMENTS	ECC data that should be set in the new call in place of the corresponding data in the current call.

Table 10-11 SingleStepConference Parameters (continued)

Parameter	Type	Description
FacilityCode (optional)	STRING, maximum length 40	A trunk access code, split extension, or other data needed to access the chosen facility.
AuthorizationCode (optional)	STRING, maximum length 40	An authorization code needed to access the resources required to initiate the call.
AccountCode (optional)	STRING, maximum length 40	A cost-accounting or client number used by the peripheral for charge-back purposes.

errorcode

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

Return Values

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

Remarks

The DialedNumber is the only required member necessary in the Arguments parameter. A SingleStepConference request will fail if the call's status is not LCS_CONNECT.

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

- OnAgentStateChange event (Hold)
- OnCallHeld event
- OnAgentStateChange event (Talking)
- OnBeginCall event
- OnServiceInitiated event
- OnCallOriginated event
- OnCallDelivered event
- OnCallConferenced event
- OnCallEnd event
- ConferenceCallConf event

The OnControlFailureConf event will be 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 will be talking to the party to be transferred and the transferring party will drop out of the call. The method requires a DialedNumber argument.

Syntax

```
C++:int SingleStepTransfer(Arguments& args)
COM:HRESULT SingleStepTransfer (/*[in]*/ IArguments * args, /*[out, retval]*/ int *
errorcode)
VB:    SingleStepTransfer (args As CTIOSCLIENTLib.IArguments, errorcode As Long)
Java:int SingleStepTransfer(Arguments rASrgs)
.NET:CilError SingleStepTransfer(Arguments args)
```

Parameters

args

An output parameter of either a reference or a pointer to an Arguments array containing parameters from Table 10-11. Any of these parameters included should be added to the Arguments array using the associated keyword.

errorcode

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

Return Values

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

Snapshot

The Snapshot method issues a server request to retrieve the current call information. If values are passed in the optional args parameter, the snapshot request will return the server's current call values for only the requested arguments. Otherwise all call information is returned, including the fields described under GetCallContext and GetCallData. See OnCallDataUpdate in Chapter 6, “Event Interfaces and Events” for more information.

Syntax

```
C++     int Snapshot()
        int Snapshot(Arguments & optional_args)
COM:    HRESULT Snapshot (/*[in,optional]*/  IArguments * optional_args,    (/*[out,
retval]*/ int * errorcode )
VB:     Snapshot([optional_args As IArguments]) As Long
Java:   int Snapshot(Arguments rArgs)
.NET:   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 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Values

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

Remarks

The current information about the call will be received in the OnCallDataUpdate event.

- The OnCallDataUpdate event will be received if this request is successful.
- The OnControlFailureConf event will be received if this request fails.

StartRecord

The StartRecord method is used to start recording a call.

Syntax

C++: `int StartRecord()`
`int StartRecord(Arguments & reserved_args);`
COM: `HRESULT StartRecord (/*[in,optional]*/ IArguments *reserved_args, (/*[out, retval]*/ int * errorcode )`
VB: `StartRecord([reserved_args As IArguments]) As Long`
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 3-2 in Chapter 3, “CIL Coding Conventions.”

Return Value

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

Remarks

Calling this method causes the CTI Server to forward the request to one or more server applications that have registered the “Cisco:CallRecording” service as described in the *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 will fail if there is no recording server available to CTIServer.

- The OnStartRecordingConf event will be received if this request is successful.
- The OnControlFailureConf event will be 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 3-2 in Chapter 3, "CIL Coding Conventions."

Return Value

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

Remarks

Calling this method causes the CTIServer to forward the request to the server application with the SessionID received in the OnStartRecordingConf event if non-zero, or if that SessionID is zero, to one or more 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 will fail if there is no recording server available to CTIServer.

- The OnStopRecordConf event will be received if this request is successful.
- The OnControlFailureConf event will be received if this request fails.

Transfer

The Transfer method transfers a call to a third party. This method may be called on either the held original call or the current consult call. If the device has only these two calls, the optional parameter is not necessary. At the end of a successful transfer, both of these calls will be gone from the device. See the Conference method for more information.

Syntax

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

COM: `HRESULT Transfer ( [in, optional] IArguments *optional_args, (/*[out, retval]*/ int * errorcode )`
VB: `Transfer([optional_args As IArguments]) As Long`
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, it should be added to the Arguments parameter with the keyword of "CallReferenceObjectID". This would only be necessary in an environment where there are multiple held calls and the request is being made through the current call. If the request is being made through a specific held call in this scenario, or if there are only two calls at the device, this parameter is unnecessary.

errorcode

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

Return Values

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

Remarks

Before making this request, it is necessary for the original call to be in the held state and the consult call to be in the talking state or the request will fail. Therefore, if the calls are alternated (See Alternate), they must be alternated again to return the two calls into their appropriate states.

If there are only two calls at the device, 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.

The Transfer request must be made via a call whose call status is LCS_CONNECT or LCS_HELD or it will fail.

The following events are received by the transfer initiator if this request is successful:

- OnCallTransferred event
- OnCallEnd event
- OnCallEnd event
- OnAgentStateChange event
- OnTransferCallConf event

The OnControlFailureConf event will be received if this request fails.

CHAPTER 11

SkillGroup Object

The SkillGroup object provides developers using the CTI OS Client Interface Library with an interface to Skill Group properties and data. The SkillGroup is mainly a representation used for accessing statistics, which can be enabled or disabled via method calls to the SkillGroup object. The SkillGroups are accessible directly from the Session object or the Agent Object.

The SkillGroup object methods can be accessed as follows:

- Via the Agent object inside the Session in Agent mode
- Via the Agent object inside the Session in Monitor mode
- In C++, Java, and .NET, via the session object inside the session in Monitor mode when the special SkillGroupStats filter is set. See the section "Filtering Skillgroup Statistics" in Chapter 8 for code examples related to the special SkillGroupStats filter.

Properties

Table 11-1 lists the available SkillGroup properties.

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

Table 11-1 Skill Group Properties

Keyword	Type	Description
SkillGroupNumber	INT	The number of the skill group from the Peripheral.
SkillGroupID	STRING	The Unified ICM SkillGroupID of the SkillGroup, if available.
SkillGroupName	STRING	The Unified ICM SkillGroupName of the SkillGroup, if available.
SkillGroupState	INT	Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.
ClassIdentifier	INT	Value represents skillgroup class.

To access statistics, first use GetValue on the Skill Group object to obtain the Statistics arguments array, then use GetValue to obtain the desired value.

Note Not all the statistics values listed in Table 11-1 are present in every system configuration. Whether a particular statistic value is available depends both on the protocol version of CTI Server with which CTI OS connects and on the peripheral on which the agent resides.The statistics listed in Table 11-2 are available in Protocol Version 8 of CTI Server.

One very important real-time skillgroup statistic is the number of calls currently in queue. Previously, this value was typically provided in CallsQNow. However, the number of calls currently in queue is now stored in RouterCallsQNow.

Statistics

Table 11-2 lists the available SkillGroup statistics.

Table 11-2 Skill Group Statistics

Statistic	Definition
AgentsLoggedOn	Number of agents that are currently logged on to the skill group.
AgentsAvail	Number of agents for the skill group in Available state ready to take calls.
AgentsNotReady	Number of agents in the Not Ready state for the skill group.
AgentsReady	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 skill group 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.
AgentsTalkingIn	Number of agents in the skill group currently talking on inbound calls.
AgentsTalkingOut	Number of agents in the skill group currently talking on outbound calls.
AgentsTalkingOther	Number of agents in the skill group currently talking on internal (not inbound or outbound) calls.
AgentsWorkNot Ready	Number of agents in the skill group in the Work Not Ready state.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
AgentsWorkReady	Number of agents in the skill group in the Work Ready state.
AgentsBusyOther	Number of agents currently busy with calls assigned to other skill groups.
AgentsReserved	Number of agents for the skill group currently in the Reserved state.
AgentsHold	Number of calls to the skill group currently on hold.
AgentsICM Available	Number of agents in the skill group currently in the ICMAvailable state.
AgentsApplication Available	Number of agents in the skillgroup currently in the Application Available state.
AgentsTalkingAutoOut	Number of calls to the skill group currently talking on AutoOut (predictive) calls.
AgentsTalking Preview	Number of calls to the skill group currently talking on outbound Preview calls.
AgentsTalking Reservation	Number of calls to the skill group currently talking on agent reservation calls.
RouterCallsQNow**	The number of calls currently queued by the CallRouter for this skill group. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
LongestRouterCallQNow**	The queue time, in seconds, of the currently Unified ICM call router queued call that has been queued to the skill group the longest. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
CallsQNow*	The number of calls currently queued to the skill group. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
CallsQTimeNow*	The total queue time, in seconds, of calls currently queued to the skill group. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
LongestCallQNow*	The queue time, in seconds, of the currently queued call that has been queued to the skill group the longest. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
AvailTimeTo5	Total seconds agents in the skill group were in the Available state.
LoggedOnTimeTo5	Total time, in seconds, agents in the skill group were logged on.
NotReadyTimeTo5	Total seconds agents in the skill group were in the Not Ready state.
AgentOutCallsTo5	Total number of completed outbound ACD calls made by agents in the skill group.
AgentOutCallsTalk TimeTo5	Total talk time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.
AgentOutCallsTimeTo5	Total handle time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
AgentOutCallsHeldTo5	The total number of completed outbound ACD calls agents in the skill group have placed on hold at least once.
AgentOutCallsHeldTimeTo5	Total number of seconds outbound ACD calls were placed on hold by agents in the skill group.
HandledCallsTo5	The number of inbound ACD calls handled by agents in the skill group.
HandledCallsTalk TimeTo5	Total talk time in seconds for Inbound ACD calls counted as handled by agents in the skill group. Includes hold time associated with the call.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
HandledCallsAfter CallTimeTo5	Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the skill group.
HandledCallsTime To5	Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the skill group. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.
IncomingCallsHeldTo5	The total number of completed inbound ACD calls agents in the skill group placed on hold at least once.
IncomingCallsHeldTimeTo5	Total number of seconds completed inbound ACD calls were placed on hold by agents in the skill group.
InternalCallsRcvdTo5	Number of internal calls received by agents in the skill group.
InternalCallsRcvd TimeTo5	Number of seconds spent on internal calls received by agents in the skill group.
InternalCallsHeldTo5	The total number of internal calls agents in the skill group placed on hold at least once.
InternalCallsHeld TimeTo5	Total number of seconds completed internal calls were placed on hold by agents in the skill group.
AutoOutCallsTo5	Total number of AutoOut (predictive) calls completed by agents in the skill group.
AutoOutCallsTalk TimeTo5	Total talk time, in seconds, for completed AutoOut (predictive) calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
AutoOutCallsTime To5	Total handle time, in seconds, for completed AutoOut (predictive) calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
AutoOutCallsHeld To5	The total number of completed AutoOut (predictive) calls that agents in the skill group have placed on hold at least once.
AutoOutCallsHeld TimeTo5	Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group.
PreviewCallsTo5	Total number of outbound Preview calls completed by agents in the skill group.
PreviewCallsTalk TimeTo5	Total talk time, in seconds, for completed outbound Preview calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.
PreviewCallsTime To5	Total handle time, in seconds, for completed outbound Preview calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
PreviewCallsHeld To5	The total number of completed outbound Preview calls that agents in the skill group have placed on hold at least once.
PreviewCallsHeld TimeTo5	Total number of seconds outbound Preview calls were placed on hold by agents in the skill group.
ReservationCallsTo5	Total number of agent reservation calls completed by agents in the skill group.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
ReservationCalls TalkTimeTo5	Total talk time, in seconds, for completed agent reservation calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.
ReservationCalls TimeTo5	Total handle time, in seconds, for completed agent reservation calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
ReservationCalls HeldTo5	The total number of agent reservation calls that agents in the skill group have placed on hold at least once.
ReservationCalls HeldTimeTo5	Total number of seconds agent reservation calls were placed on hold by agents in the skill group.
BargeInCallsTo5	Total number of supervisor call barge-ins completed in the skill group.
InterceptCallsTo5	Total number of supervisor call intercepts completed in the skill group.
MonitorCallsTo5	Total number of supervisor call monitors completed in the skill group.
WhisperCallsTo5	Total number of supervisor call whispers completed by agents in the skill group.
EmergencyCallsTo5	Total number of emergency calls completed by agents in the skill group.
CallsQ5*	The number of calls queued to the skill group during the current five-minute. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
CallsQTime5*	The total queue time, in seconds, of calls queued to the skill group during the current five-minute. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
LongestCallQ5[*]	The longest queue time, in seconds, of all calls queued to the skill group during the current five-minute. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
AvailTimeToHalf	Total seconds agents in the skill group were in the Available state.
LoggedOnTime ToHalf	Total time, in seconds, agents in the skill group were logged on.
NotReadyTime ToHalf	Total seconds agents in the skill group were in the Not Ready state.
AgentOutCallsTo Half	Total number of completed outbound ACD calls made by agents in the skill group.
AgentOutCallsTalk TimeToHalf	Total talk time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.
AgentOutCallsTimeToHalf	Total handle time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
AgentOutCallsHeldToHalf	The total number of completed outbound ACD calls agents in the skill group have placed on hold at least once.
AgentOutCallsHeldTimeToHalf	Total number of seconds outbound ACD calls were placed on hold by agents in the skill group.
HandledCallsToHalf	The number of inbound ACD calls handled by agents in the skill group.
HandledCallsTalk TimeToHalf	Total talk time in seconds for Inbound ACD calls counted as handled by agents in the skill group. Includes hold time associated with the call.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
HandledCallsAfter CallTimeToHalf	Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the skill group.
HandledCallsTime ToHalf	Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the skill group. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.
IncomingCallsHeldToHalf	The total number of completed inbound ACD calls agents in the skill group placed on hold at least once.
IncomingCallsHeldTimeToHalf	Total number of seconds completed inbound ACD calls were placed on hold by agents in the skill group.
InternalCallsRcvdToHalf	Number of internal calls received by agents in the skill group.
InternalCallsRcvd TimeToHalf	Number of seconds spent on internal calls received by agents in the skill group.
InternalCallsHeldToHalf	The total number of internal calls agents in the skill group placed on hold at least once.
InternalCallsHeld TimeToHalf	Total number of seconds completed internal calls were placed on hold by agents in the skill group.
AutoOutCallsToHalf	Total number of AutoOut (predictive) calls completed by agents in the skill group.
AutoOutCallsTalk TimeToHalf	Total talk time, in seconds, for completed AutoOut (predictive) calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
AutoOutCallsTime ToHalf	Total handle time, in seconds, for completed AutoOut (predictive) calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
AutoOutCallsHeld ToHalf	The total number of completed AutoOut (predictive) calls that agents in the skill group have placed on hold at least once.
AutoOutCallsHeld TimeToHalf	Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group.
PreviewCallsToHalf	Total number of outbound Preview calls completed by agents in the skill group.
PreviewCallsTalk TimeToHalf	Total talk time, in seconds, for completed outbound Preview calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.
PreviewCallsTime ToHalf	Total handle time, in seconds, for completed outbound Preview calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
PreviewCallsHeldToHalf	The total number of completed outbound Preview calls that agents in the skill group have placed on hold at least once.
PreviewCallsHeld TimeToHalf	Total number of seconds outbound Preview calls were placed on hold by agents in the skill group.
ReservationCallsToHalf	Total number of agent reservation calls completed by agents in the skill group.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
ReservationCalls TalkTimeToHalf	Total talk time, in seconds, for completed agent reservation calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.
ReservationCalls TimeToHalf	Total handle time, in seconds, for completed agent reservation calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
ReservationCalls HeldToHalf	The total number of agent reservation calls that agents in the skill group have placed on hold at least once.
ReservationCalls HeldTimeToHalf	Total number of seconds agent reservation calls were placed on hold by agents in the skill group.
BargeInCallsToHalf	Total number of supervisor call barge-ins completed in the skill group.
InterceptCallsTo Half	Total number of supervisor call intercepts completed in the skill group.
MonitorCallsToHalf	Total number of supervisor call monitors completed in the skill group.
WhisperCallsToHalf	Total number of supervisor call whispers completed by agents in the skill group.
EmergencyCalls ToHalf	Total number of emergency calls completed by agents in the skill group.
CallsQHalf*	The number of calls queued to the skill group during the current half hour. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
CallsQTimeHalf*	The total queue time, in seconds, of calls queued to the skill group during the current half hour. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
LongestCallQHalf*	The longest queue time, in seconds, of all calls queued to the skill group during the current half hour. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
AvailTimeToday	Total seconds agents in the skill group were in the Available state.
LoggedOnTime Today	Total time, in seconds, agents in the skill group were logged on.
NotReadyTime Today	Total seconds agents in the skill group were in the Not Ready state.
AgentOutCalls Today	Total number of completed outbound ACD calls made by agents in the skill group.
AgentOutCallsTalk TimeToday	Total talk time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.
AgentOutCallsTimeToday	Total handle time, in seconds, for completed outbound ACD calls handled by agents in the skill group. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
AgentOutCallsHeldToday	The total number of completed outbound ACD calls agents in the skill group have placed on hold at least once.
AgentOutCallsHeldTimeToday	Total number of seconds outbound ACD calls were placed on hold by agents in the skill group.
HandledCallsToday	The number of inbound ACD calls handled by agents in the skill group.
HandledCallsTalk TimeToday	Total talk time in seconds for Inbound ACD calls counted as handled by agents in the skill group. Includes hold time associated with the call.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
HandledCallsAfter CallTimeToday	Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the skill group.
HandledCallsTime Today	Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the skill group. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.
IncomingCallsHeldToday	The total number of completed inbound ACD calls agents in the skill group placed on hold at least once.
IncomingCallsHeldTimeToday	Total number of seconds completed inbound ACD calls were placed on hold by agents in the skill group.
InternalCallsRcvd Today	Number of internal calls received by agents in the skill group.
InternalCallsRcvd TimeToday	Number of seconds spent on internal calls received by agents in the skill group.
InternalCallsHeld Today	The total number of internal calls agents in the skill group placed on hold at least once.
InternalCallsHeld TimeToday	Total number of seconds completed internal calls were placed on hold by agents in the skill group.
AutoOutCallsToday	Total number of AutoOut (predictive) calls completed by agents in the skill group.
AutoOutCallsTalk TimeToday	Total talk time, in seconds, for completed AutoOut (predictive) calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
AutoOutCallsTime Today	Total handle time, in seconds, for completed AutoOut (predictive) calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
AutoOutCallsHeld Today	The total number of completed AutoOut (predictive) calls that agents in the skill group have placed on hold at least once.
AutoOutCallsHeld TimeToday	Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group.
PreviewCallsToday	Total number of outbound Preview calls completed by agents in the skill group.
PreviewCallsTalk TimeToday	Total talk time, in seconds, for completed outbound Preview calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.
PreviewCallsTime Today	Total handle time, in seconds, for completed outbound Preview calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
PreviewCallsHeld Today	The total number of completed outbound Preview calls that agents in the skill group have placed on hold at least once.
PreviewCallsHeld TimeToday	Total number of seconds outbound Preview calls were placed on hold by agents in the skill group.
ReservationCalls Today	Total number of agent reservation calls completed by agents in the skill group.

Table 11-2 Skill Group Statistics (continued)

Statistic	Definition
ReservationCalls TalkTimeToday	Total talk time, in seconds, for completed agent reservation calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent begins after call work for the call. The time includes hold time associated with the call.
ReservationCalls TimeToday	Total handle time, in seconds, for completed agent reservation calls handled by agents in the skill group. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
ReservationCalls HeldToday	The total number of agent reservation calls that agents in the skill group have placed on hold at least once.
ReservationCalls HeldTimeToday	Total number of seconds agent reservation calls were placed on hold by agents in the skill group.
BargeInCallsToday	Total number of supervisor call barge-ins completed in the skill group.
InterceptCallsToday	Total number of supervisor call intercepts completed in the skill group.
MonitorCallsToday	Total number of supervisor call monitors completed in the skill group.
WhisperCallsToday	Total number of supervisor call whispers completed by agents in the skill group.
EmergencyCalls Today	Total number of emergency calls completed by agents in the skill group.
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 skill group. 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 skill group. 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

Table 11-3 lists the SkillGroup object methods.

Table 11-3 SkillGroup Object Methods

Method	Description
GroupStatistics	Disables skill group statistic messages.
DumpProperties	See Chapter 7, "CtiOs Object."
EnableSkillGroupStatistics	Enables skill group statistic messages.
GetElement	See Chapter 7, "CtiOs Object."
GetNumProperties	See Chapter 7, "CtiOs Object."
GetPropertyName	See Chapter 7, "CtiOs Object."
GetValue	See Chapter 7, "CtiOs Object."
GetValueInt (C++) GetValueIntObj (Java)	See Chapter 7, "CtiOs Object."
GetValueString	See Chapter 7, "CtiOs Object."
IsValid	See Chapter 7, "CtiOs Object."
SetValue	See Chapter 7, "CtiOs Object."

DisableSkillGroupStatistics

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

Syntax

C++:`int DisableSkillGroupStatistics(Arguments & args)`
COM:`HRESULT DisableSkillGroupStatistics (IArguments * args, int * errorCode)`
VB: `DisableSkillGroupStatistics (args As CTIOSCLIENTLib.IArguments, errorCode As Long)`
Java:`int DisableSkillGroupStatistics(Arguments args)`
.NET:`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. See the Remarks section for a code example. Otherwise, this parameter is not used.

errorCode

An output parameter (return parameter in VB) that contains an error code, if any.

Return Value

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

Remarks

The CTI OS server sends skillgroup statistics in an OnSkillGroupStatisticsUpdated event. If this request is successful, the OnNewSkillGroupStatistics event is no longer received.

The following is a C++ code example where the args parameter contains values for PeripheralID and SkillGroupNumber.

```
Arguments & argsStatBroadcast = Arguments::CreateInstance();
argsStatBroadcast.AddItem(CTIOS_SKILLGROUPNUMBER, intSG);
argsStatBroadcast.AddItem(CTIOS_PERIPHERALID, m_periphID);
m_pSkGrStatSession->DisableSkillGroupStatistics ( argsStatBroadcast );
argsStatBroadcast.Release();
```

DumpProperties

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

EnableSkillGroupStatistics

The EnableSkillGroupStatistics method requests that real-time statistics be sent to the skillgroup object. In an agent mode application, this request is usually made through the agent object (see Chapter 10, “Call Object”). 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. See the Remarks section for a code example. Otherwise, this parameter is not used.

errorCode

An output parameter (return parameter in VB) that contains an error code, if any.

Return Value

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

Remarks

CTI OS Server sends skillgroup statistics in an OnSkillGroupStatisticsUpdated event.

The following is a C++ code example where the args parameter contains values for 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

See Chapter 7, "CtiOs Object" for a description of the GetElement method.

GetValue Methods

See Chapter 7, "CtiOs Object" for descriptions of the GetValue, GetValueInt, GetValueList, and GetValueString methods.

IsValid

See Chapter 7, "CtiOs Object" for a description of the IsValid method.

SetValue

See Chapter 7, "CtiOs Object" for a description of the SetValue method.

CHAPTER 12

Helper Classes

The CTI OS Client Interface Library usesof several custom data structures. This chapter describes the CTI OS Helper Classes (data structures). The following helper classes are distributed with the Client Interface Library:

- **Arg**. The Arg structure is the basic data type used in the CIL for any parameter included in methods or events. Objects of this type allow the CIL to be fully extensible and reusable. Arg supports many useful types including string, integer, Boolean, and Arguments array. Arg is the base class for the Arguments class. In most programming scenarios, programmers will not use Arg directly, but indirectly through the Arguments class.
- **Arguments**. The Arguments structure is used to maintain and send a set of key-value pairs between the CIL and CTI OS Server for events and requests. The Arguments array elements must all be of type Arg. The Arguments structure enables future growth of the CTI OS feature set, without requiring changes to the method call signature.
- **CilRefArg**. The CilRefArg class is a specialized subclass of Arg. It is used to store a reference to an object derived from CCtiOsObject (C++ only). For instance, it can hold reference to a CAgent, CCall, CSkillGroup, CCtiOsSession, or CWaitObject.
- **CCtiosException**. The CCtiosException class is used by CTI OS to provide detailed information when an exception occurs (C++ and Java only). When an exception is caught as CCtiosException, the programmer can query it for details such as error codes and error messages.
- **CWaitObject**. CWaitObject is a CIL object that derives from CtiOsObject. It is a utility class (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. Wait objects are created with the CreateWaitObject Session Object method and destroyed with the DestroyWaitObject Session Object method.
- **Logger**. 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.
- **LogWrapper**. 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 12-2.

Arg class methods will do conversion between types whenever possible. For example, you can do a SetValue(25) and then do a GetValueString() which will return the string "25". You can also do a SetValue("25") and then do a GetValueIntObj which will return an Integer object containing the numeric value 25. However, if you call SetValue "abc" and try to retrieve it as an int, it will fail.

Table 12-1 lists the available Arg class methods.

Table 12-1 Arg Class Methods

Method	Description
AddRef	Increments the reference count for the data item.
Clone	Creates an exact copy of the Arg object.
CreateInstance	Creates an Arg object.
DumpArg	Builds a string containing the value stored in the Arg.
GetType	Returns the type of the data stored in the argument (one of the values in Table 12-2).
GetValueInt GetValueUInt GetValueUInt GetValueUShort GetValueShort GetValueBool GetValueString	Returns the value stored in the argument.
SetValue	Sets the data in the Arg object.

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

AddRef

The AddRef method increments the reference count for the data item. It is necessary to call this if you are storing a pointer to the item for some time (for example, if you plan to store and use Arguments received in an event handler after the event thread has returned to the calling code). When you are finished with the item, you must call the Release method or a memory leak will result.

Syntax

C++: `unsigned long AddRef()`
COM: `HRESULT AddRef()`
VB, Java, .NET: `Not used`

Parameters

None.

Return Values

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

C++: The current reference count after the AddRef() call.

Clone

The Clone method allocates a new Arg in memory and copies its key, value, and type to the new instance. When using the C++ or COM CILs, it is important to release the object when it is no longer needed.

Syntax

```
C++:    Arg & Clone()
COM:    HRESULT Clone(/*[out, retval]*/ IArg** arg);
VB:     Clone() as CTIOSCLIENTLib.IArg
Java:   Arg Clone()
.NET:   Ojbect Clone()
```

Output Parameters

arg

Pointer to an IArg instance that is a copy of the object.

Return Values

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

Others: If successful, will return a reference to a new Arg object. If unsuccessful in C++ or VB, it will throw a CCtiosException with iCode set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED. If unsuccessful in Java, it returns null but does not throw an exception.

CreateInstance

The CreateInstance method creates an object of type Arg class and sets the reference count of the object to 1. It is important to release the object when it is no longer in use in the program.

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. See Chapter 3, "CIL Coding Conventions."

Others: Either a reference to the newly created Arg or a boolean indicating method success. If the methods not returning bool are unsuccessful, they will raise a CCtiosException with iCode set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED.

Remarks

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 will automatically convert an INTEGER type to a string that can be logged for debugging. In the event that a Arg object is actually an Arguments object, the string returned is the one built by Arguments.DumpArg, and thus enabled printing of nested Arguments structures.

Syntax

C++: `string DumpArg()`
COM: `HRESULT DumpArg([out,retval] BSTR* arg_string);`
VB: `DumpArg() as String`
Java, .NET:`Not available. Use the ToString method.`

Parameters

arg_string

The pointer to the string into which the contents of the Arg object will be written.

Return Values

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

Others: A string containing the contents of the structure.

GetArgType (.NET only)

The GetArgType method returns the type of the contained value. This returned value will be one of the following:

- ARG NOTSET
- ARG_BOOL
- ARG_SHORT
- ARG_USHORT
- ARG_INT
- ARG_UINT

See Table 12-2 for a list of valid types.

Syntax

COM, C++, Java: `Use GetType.`
.NET: `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. See Table 12-2 for a list of possible types.

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 receive the enumerated constant that identifies data type stored in IArg.

Return Values

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

Others: Returns the enumerated value that identifies the data type stored in the Arg (see Table 12-2),

Table 12-2 enumArgTypes

Argument Type	Description
ARG_NOTSET	Argument type not determined
ARG_INT	Signed integer
ARG_UINT	Unsigned integer
ARG_USHORT	2 bytes unsigned integer
ARG_SHORT	2 bytes signed integer
ARG_BOOL	1 byte integer
ARG_STRING	Character string
ARG_ARGARRAY	Variable length Arguments array

GetValue Methods

The GetValue method returns the value stored in the object. To extract a specific type of data you invoke the method designated for it. For more detail on GetValueArray, GetValueInt, and GetValueString, see the corresponding methods described in Chapter 7, “CtiOs Object.”

Syntax

```
C++:int GetValueInt();
    unsigned int GetValueUInt();
    unsigned shortGetValueUShort();
    short GetValueShort();
    string& GetValueString();
    bool GetValueBool();
    bool GetValueInt(int * value);
    bool GetValueUInt(unsigned int * value);
    bool GetValueUShort(unsigned short * value);
    bool GetValueShort( short * psVallue);
    bool GetValueBool( bool * value);
    bool GetValueString(string* value);
COM:  HRESULT GetValue(/*[out, retval]*/ VARIANT* value);
VB:   GetValue() as Variant
VB:   GetValue (key as String, value as Variant) as Boolean
Java: Integer GetValueIntObj()
      Long    GetValueUIntObj()
      Short   GetValueShortObj()
      Integer GetValueUShortObj()
      Boolean GetValueBoolObj()
      String  GetValueString()
.NET:System.Boolean GetValueInt(out System.Int32 nValue)
.NET:System.Boolean GetValueUInt(out System.Int64 nValue)
.NET:System.Boolean GetValueShort(out System.Int16 nValue)
.NET:System.Boolean GetValueUShort(out System.Int32 nValue)
.NET:System.Boolean GetValueBool(out System.Boolean bValue)
.NET: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 Table 12-2.

Table 12-3 Variant Types Supported by GetValue (COM)

Variant Type	Standard C++ Type
VT_INT	Int
VT_UINT	Unsigned int
VT_I2	Short
VT_UI2	Unsigned short
VT_BOOL	Bool
VT_BSTR	string, const string and char *

Return Values

C++

Methods taking no parameters, if successful, will return the value in the object; otherwise, they will raise a CCtiosException with iCode set to E_CTIOS_INVALID_ARGUMENT.

The methods taking a pointer to the variable receiving the result will return true, if the method was able to get the value, otherwise, false.

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 (i.e., value->vt) to any of the types listed in Table 12-2, it returns the value in the appropriate field of the variant. Otherwise it returns VT_EMPTY.

Release

The Release method decrements the reference count for the data item. It is necessary to call Release when you are finished with a data item that has had its reference count incremented via CreateInstance or AddRef; otherwise, a memory leak will result.

Syntax

C++: `unsigned long Release()`
COM: `HRESULT Release()`
VB, Java, .NET: `Not used`

Parameters

None.

Return Values

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

C++: The current reference count after the Release() call.

SetValue

The SetValue method sets the value in the Arg object.

Syntax

```
C++: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 Table 12-4.

Table 12-4 Supported Varient Types

Variant Type	Standard C++ Type
VT_INT	Int
VT_UINT	Unsigned int
VT_I2	Short
VT_UI2	Unsigned short
VT_BOOL	Bool
VT_BSTR	string, const string and char *
VT_DISPATCH	Pointer to an IArg interface

errorcode

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

Return Values

C++

If the method was able to set the value it returns true, otherwise it returns false.

COM, VB

If the method was able to set the value it returns VARIANT_TRUE. Otherwise, it returns VARIANT_FALSE.

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.

Table 12-5 lists the Arguments class methods.

Table 12-5 Arguments Class Methods

Method	Description
AddItem	Adds an item to an Arguments array.
AddRef	Increments the reference count for the data item.
Clear	Deletes all elements from an Arguments array.
Clone	Creates a copy of an Arguments array.
CreateInstance	Creates an Arguments array.
DumpArgs	Returns Arguments object as a string
GetElement (also GetElementInt, GetElementUInt, GetElementUShort, GetElementShort, GetElementBool, GetElementString, GetElementArg, GetElementKey GetElementArgType)	Returns the value stored under a specified index.
GetValue (also GetValueInt, GetValueUShort, GetValueShort, GetValueBool, GetValueUInt, GetValueString, GetValueArray, GetValueArg)	Returns the value stored under a specified key.
IsValid	Tests if a key is present in the current Arguments array.
NumElements	Returns the number of arguments in the current Arguments array,.
Release	Decrements the reference count for the data item.
RemoveItem	Removes an item from an Arguments array.
SetElement	Sets the value of an index.
SetValue	Sets the value of a key.

Usage Notes

When writing an application using the CTI OS SDK, the following sequence of steps in the program may produce a problem:

- Programmer passes an Arguments array into a CTI OS SDK method (methodA)
- MethodA returns
- Programmer modifies the same Arguments array
- Programmer passes the modified Arguments array into another CTI OS SDK method (methodB)

When running the application, the call to methodA may behave as if it was passed the modified Arguments array. This is because many CTI OS methods simply place a pointer to the Arguments array on a queue of items to be sent to CTI OS server. When the same Arguments array is later modified, as in the preceding example, the pointer on the queue now points to the modified array and the modified array is sent to CTI OS server. A problem may occur depending on timing, as there are multiple threads involved: the thread pulling items off the queue and the thread modifying the Arguments array. If the queued message is sent to CTI OS before the Arguments array is modified, the problem will not occur.

To avoid this problem, call the Clone method on the initial Arguments array and modify the copy rather than modifying the original. For example, the preceding example would change as follows:

- Programmer passes an Arguments array (initialArray) into a CTI OS SDK method (methodA)
- MethodA returns
- modifiedArray = initialArray.Clone()
- Programmer modifies modifiedArray
- Programmer passes the modifiedArray into another CTI OS SDK method (methodB)

AddItem (C++, COM, VB only)

The AddItem method expects a key-value pair. The key value may be a string or an integer. The value may be a string, an integer, or an object reference. If there is an entry with the same key, it will be replaced with this entry, otherwise the new key-value pair will be added to the arguments array. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Syntax

C++:

```
bool AddItem( std::string& key, int value );
bool AddItem( std::string& key, unsigned int value );
bool AddItem( std::string& key, unsigned short value );
bool AddItem( std::string& key, short value );
bool AddItem( std::string& key, bool value );

bool AddItem( std::string& key, char * pchar );
bool AddItem( std::string& key, std::string& value );
bool AddItem( std::string& key, Arg& value );
bool AddItem( std::string& key, const Arg& value );
bool AddItem( std::string& key, Arguments& value );
bool AddItem( std::string& key, const Arguments& value);

bool AddItem( char * key, int value );
bool AddItem( char * key, unsigned int value );
bool AddItem( char * key, unsigned short value );
bool AddItem( char * key, short value );
bool AddItem( char * key, bool value );

bool AddItem( char * key, char * value   );
bool AddItem( char * key, std::string& value );
bool AddItem( char * key, Arg& cArg );
bool AddItem( char * key, const Arg& value );
bool AddItem( char * key, Arguments& value );
bool AddItem( char * key, const Arguments& value );

bool AddItem( enum_Keywords key, int value );
bool AddItem( enum_Keywords key, unsigned int value );
bool AddItem( enum_Keywords key, unsigned short value );
bool AddItem( enum_Keywords key, short value );
```

```
        bool AddItem( enum_Keywords key, bool value );

        bool AddItem( enum_Keywords key, char * value );
        bool AddItem( enum_Keywords key, std::string& value );
        bool AddItem( enum_Keywords key, Arg& cArg );
        bool AddItem( enum_Keywords key, const Arg& value );
        bool AddItem( enum_Keywords key, Arguments& value );
        bool AddItem( enum_Keywords key, const Arguments& value)
COM:    HRESULT AddItem(/*[in]*/ VARIANT *key, /*[in]*/ VARIANT *value,
            /*[out,retval]*/ VARIANT_BOOL success) As Boolean;
VB:     AddItem( Key as Variant, Value as Variant)
Java, .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; see Chapter 3, "CIL Coding Conventions."

AddRef (C++ and COM only)

The AddRef method increments the reference count for the data item. It is necessary to call this if you are storing a pointer to the item for some time. When you are finished with the item, you must call the Release method or a memory leak will result.

Syntax

```
C++:        unsigned long AddRef()
COM:        HRESULT AddRef()
VB, Java, .NET: Not used
```

Parameters

None.

Return Values

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

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:

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

To cast in VB, do the following:

```
Dim Args As CTIOSCLIENTLib.IArguments
    Set Args = Orig.Clone()
```

Syntax

C++: `Arg & Clone()`
COM: `HRESULT Clone(/*[out, retval]*/ IArguments ** args);`
VB: `Clone() as CTIOSCLIENTLib.IArguments`
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. See Chapter 3, "CIL Coding Conventions."

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. See Chapter 3, "CIL Coding Conventions."

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() after you are finished with the Arg.

DumpArgs

The DumpArgs method builds a string showing all of the members of the Arguments structure in the form "key1 = value1; key2 = value2;...". It is primarily used for debugging.

Syntax

C++: `string DumpArgs()`
COM: `HRESULT DumpArgs([out,retval] BSTR* arg_string);`
VB: `DumpArgs() as String`
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. See Chapter 3, "CIL Coding Conventions."

Others: A string containing the contents of the Arguments array listing all key/value pairs

GetElement Methods

The GetElement method is similar to GetValue, except that it uses an index value instead of a key. The index value is not related to the order in which items are added or removed. The order of items in Arguments is never guaranteed. This method is useful for sequentially iterating over all items in Arguments. The Index is 1-based. The Index should never be less than one or greater than NumElements. see also NumElements method. The GetElementKey returns the key of a given index.

Syntax

C++:
```
Arg& GetElement( int index );
bool GetElement( int index, Arg ** value);
int GetElementInt( int index );
bool GetElementInt( int index, int * value);
unsigned int GetElementUInt( int index );
bool GetElementUInt( int index, unsigned int * value);
unsigned short GetElementUShort( int index );
bool GetElementUShort( int index, unsigned short * value );
short GetElementShort( int index );
bool GetElementShort( int index, short * value);
bool GetElementBool( int index );
bool GetElementBool( int index, bool * value);
std::string GetElementString( int index );
bool GetElementString( int index, std::string * value);
Arguments& GetElementArg( int index );
bool GetElementArg( int index, Arguments ** key);
std::string GetElementKey( int index );
bool GetElementKey( int nIndex, std::string * key);
bool GetElementKey( int nIndex, int * key);
```
COM:
```
HRESULT GetElementKey(/*[in]*/ int index, /*[out]*/ BSTR *
    key);
HRESULT GetElement(/*[in]*/ int index, /*[out]*/ VARIANT *
    value);
```
VB:
```
GetElement (Integer index, Variant value)
GetElement (Integer index, String key)
```
Java:
```
Arg GetElement(int iIndex)
Arguments GetElementArguments(int iIndex)
Integer GetElementIntObj(int iIndex)
Long GetElementUIntObj(int iIndex)
Short GetElementShortObj(int iIndex)
Integer GetElementUShortObj(int iIndex)
Boolean GetElementBoolObj(int iIndex)
String GetElementString(int iIndex)
String GetElementKey(int iIndex)
```
.NET:
```
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
    strValue)
Boolean GetElementArguments(System.Int32 iIndex, out Arguments
    argArguments)
Boolean GetElementKey(System.Int32 iIndex, out System.String
    strKey)
```

Parameters

value

An output parameter containing the value of the member at the specified index.

key

An output parameter containing the key of the member at the specified index.

index

An input parameter containing an index into the Arguments array.

Return Value

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

Others: Returns either the value at the index specified independently from its key, or a boolean indicating success or failure.

GetValue Methods

The GetValue methos return the value stored under a key. The existence of a key can be tested using IsValid. Keys are not case sensitive. Leading and trailing spaces are always removed from the key. For more detail on GetValueArray, GetValueInt, and GetValueString, see the corresponding methods described in Chapter 7, "CtiOs Object."

Syntax

C++:

```
Arg& GetValue( enum_Keywords eKey );
bool GetValue( enum_Keywords key, Arg ** value );
Arg& GetValue( std::string& key);
bool GetValue( std::string& key, Arg ** value);
Arg& GetValueArg( std::string& key);
bool GetValueArg( std::string& key, Arg ** value);
int GetValueInt( enum_Keywords key); /*throws exception*/
bool GetValueInt( enum_Keywords key, int * value);
unsigned int GetValueUInt( enum_Keywords key);
bool 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);
Arguments& GetValueArray( std::string& key  );
bool GetValueArray( std::string& key , Arguments ** value);
Arguments& GetValueArray( enum_Keywords key  );
bool GetValueArray( enum_Keywords key , Arguments ** value);
Arg& GetValue( char * key );
bool GetValue( char * key, Arg ** value);
Arguments& GetValueArray( char * key );
bool GetValueArray( char * key, Arguments ** value);
int GetValueInt( char * key );
bool GetValueInt( char * key, int * value);
unsigned int GetValueUInt( char * key );
bool GetValueUInt( char * key, unsigned int * value);
unsigned short GetValueUShort( char * key );
bool GetValueUShort( char * key, unsigned short * value);
short GetValueShort( char * key );
bool GetValueShort( char * key, short * value);
bool GetValueBool( char * key );
bool GetValueBool( char * key, bool * value);
std::string GetValueString( char * key );
bool GetValueString( char * key, std::string * value);
Arg& GetValueArg( char * key );
bool GetValueArg( char * key, Arg ** value);
```

COM:

```
HRESULT GetValue(/*[in]*/ BSTR key, /*[out, retval]*/ VARIANT *
    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 GetValueUInt(System.String sKey, out System.Int64
             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 (see Appendix A, "CTI OS Keywords and Enumerated Types") or a string specifying the keyword of the value to be retrieved.

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. See also Chapter 3, "CIL Coding Conventions."

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 will occur. To resolve this error, it is recommended that you use a 32-bit integer (Long).

Those methods that do not return a bool indicating success or failure will throw a CtiosException if the method fails. The most common reasons for failure are NULL key or element with specified key not found.

IsValid

The IsValid method returns True if the specified key exists in the current Arguments array, otherwise it returns False.

Syntax

```
C++:    bool IsValid( std::string& key );
        bool IsValid( char * key );
        bool IsValid( Arg& arg );
        bool IsValid( enum_Keywords key );
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 string or enumerated (see Appendix A, “CTI OS Keywords and Enumerated Types”); all others expect the key as a string.

Return Values

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

Others: True if key exists in the current Arguments array, otherwise False.

NumElements

The NumElements method returns number of elements stored in the current arguments array. This method is useful in combination with GetElement to implement a “for” loop to iterate over all values of an arguments array without knowing the keywords (those can be retrieved at the same time using GetElementKey).

Syntax

```
C++:    int  NumElements();
COM:    HRESULT NumElements(/*[out, retval]*/ int * num_elements);
VB:     NumElements as Integer
Java:   int NumElements()
.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. See Chapter 3, "CIL Coding Conventions."

Others: Number of elements in arguments array.

Release (C++ and COM only)

The Release method decrements the reference count for the data item. It is necessary to call Release when you are finished with a data item that has had its reference count incremented via CreateInstance or AddRef; otherwise, a memory leak will result.

Syntax

```
C++:    unsigned long Release()
COM:    HRESULT Release()
VB, Java, .NET: Not used
```

Parameters

None.

Return Values

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

C++: The current reference count after the Release() call.

RemoveItem

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

Syntax

```
C++:bool RemoveItem( std::string& key );
    bool RemoveItem( char * key );
    bool RemoveItem( enum_Keywords key );
COM:  HRESULT RemoveItem(/*[in]*/ VARIANT* key, /*[out, retval]*/
      VARIANT_BOOL* bSuccess);
VB:   RemoveItem ( key as Variant) as Boolean
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. See Chapter 3, "CIL Coding Conventions."

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. See Chapter 3, "CIL Coding Conventions."

Others: A boolean indicating success or failure.

SetValue

The SetValue method sets a value for a key. Keys are *not* case sensitive. Leading and trailing spaces are always removed from the key.

Syntax

C++:
```
bool SetValue( std::string& key, int value );
bool SetValue( std::string& key, unsigned int value );
bool SetValue( std::string& key, unsigned short value );
bool SetValue( std::string& key, short value );
bool SetValue( std::string& key, bool value );
bool SetValue( std::string& key, std::string& value );
bool SetValue( std::string& key, char * pchar );
bool SetValue( std::string& key, Arg& value );
bool SetValue( std::string& key, Arguments& value );
bool SetValue( std::string& key, const Arguments& value);
bool SetValue( char * key, int value );
bool SetValue( char * key, unsigned int value );
bool SetValue( char * key, unsigned short value );
bool SetValue( char * key, short value );
bool SetValue( char * key, bool value );
bool SetValue( char * key, std::string& value );
bool SetValue( char * key, char * value   );
bool SetValue( char * key, Arg& value );
bool SetValue( char * key, Arguments& value );
bool SetValue( char * key, const Arguments& value );
bool SetValue( enum_Keywords key, int value );
bool SetValue( enum_Keywords key, unsigned int value );
bool SetValue( enum_Keywords key, unsigned short value );
bool SetValue( enum_Keywords key, short value );
bool SetValue( enum_Keywords key, bool value );
bool SetValue( enum_Keywords key, std::string& value );
bool SetValue( enum_Keywords key, Arg& value );
bool SetValue( enum_Keywords key, const Arg& value );
bool SetValue( enum_Keywords key, Arguments& value );
bool SetValue( enum_Keywords key, const Arguments&
   cArguments);
bool SetValue( enum_Keywords key, char * value );
```
COM:
```
HRESULT SetValue(/*[in]*/ VARIANT* key, /*[in]*/ VARIANT*
    value,/*[out, retval]*/ VARIANT_BOOL* success);
```
VB:
```
SetValue (key as String, value as Variant) as Boolean
```
Java:
```
boolean SetValue(Arguments rArguments)
boolean SetValue(int iKey, Arg rArg)
boolean SetValue(String sKey, Arg rArg)
boolean SetValue(int iKey, int iVal)
boolean SetValue(String sKey, int iVal)
boolean SetValue(int iKey, short nValue)
```

```
         boolean SetValue(String sKey, short nValue)
         boolean SetValue(int iKey, String sValue)
         boolean SetValue(String sKey, String sValue)
         boolean SetValueUInt(int iKey, long lValue)
         boolean SetValueUInt(String sKey, long lValue)
         boolean SetValueUShort(int iKey, int iValue)
         boolean SetValueUShort(String sKey, int iValue)
         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, 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. See Chapter 3, “CIL Coding Conventions.”

Others: A boolean indicating success or failure.

Remarks

The C++ methods overload several implementations for different value types and allow to specify a key via enumerated keywords (see Appendix A, “CTI OS Keywords and Enumerated Types”) as well as string. COM and VB use String keywords and Variants as values.

CILRefArg Class (C++, 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 (see Chapter 7, "CtiOs Object"). This class allows object references to be included in argument structure. The object types that can be used are any of the following: CAgent, CCall, CSkillGroup, CWaitObject or CCtiOsSession.

In addition to the methods inherited from the Arg class, the CILRefArg class contains the methods listed in Table 12-6.

Table 12-6 CILRefArg Class Methods

Method	Description
GetType	Returns the ARG_REFERENCE.
GetUniqueObjectID	Returns the UID of the contained CtiOsObject
GetValue	Returns the encapsulated pointer in the object.
SetValue	Encapsulates the pointer to CTI OS object into the CILRefArg object.

GetType

The GetType method returns the type of the data stored by the Arg. For a CilRefArg, this will always be ARG_REFERENCE.

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 the data type stored in Arg. In this case, that data type will be ARG_REFERENCE.

Return Values

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

Others: Returns the enumerated value that identifies the data type stored in the Arg (see Table 12-2). For CilRefArg, this will always be ARG_REFERENCE.

GetUniqueObjectID (Java and .NET only)

The GetUniqueObjectID method returns the unique objectID of the contained CtiOsObject.

Syntax

```
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 CTI OS object encapsulated in the CILRefArg.

Syntax

```
C++:  CCtiOsObject * GetValue():
Java: CCtiOsObject GetValue();
.NET: 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

```
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.

Table 12-7 lists the available CCtiOsException class methods.

Table 12-7 CCtiOsException Class Methods

Method	Description
CCtiosException	Class constructor.
GetCode	Returns the error code that generated the exception.
GetStatus	Returns the error status that generated the exception.
GetString	Returns a text string containing the description of the exception.
What	Returns a text string containing the description of the exception, the code of an error and the status.

CCtiosException Constructor

The CCtiosException constructor initializes an object of type CCtiosException.

Syntax

```
C++, Java, .NET: CCtiosException(const char *pMsg, int iCode, int iStatus);
C++: CCtiosException(const string& rstrMsg, int iCode, int iStatus);
```

Input Parameters

pMsg

Pointer to string that holds a description of an error.

iCode

Number that identifies an error.

iStatus

Status of an error.

rstrMsg

An STL string that holds a description of an error.

Return Values

None.

GetCode

The GetCode method returns the error code that generated the exception.

Syntax

C++, 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 – see also Appendix A, "CTI OS Keywords and Enumerated Types."

GetStatus

The GetStatus method returns the error status that generated the exception.

Syntax

C++, Java, and .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 will deadlock and the event handler will not receive any more events.**

Methods

Table 12-8 list the CWaitObject class methods.

Table 12-8 CWaitObject Class Methods

Method	Description
CreateWaitObject	See Chapter 8, "Session Object."
DestroyWaitObject	See Chapter 8, "Session Object."
DumpEventMask	Returns a printable string listing the events in the CWaitObject's mask
GetMask	Sets a user provided pointer to an Arguments object that contains the list of events that the object will wait for.
GetTriggerEvent	Gets the ID of the event that triggered the WaitOnMultipleEvents method to wake.
InMask	Returns true if the specified event ID is in the list of events that the object will wait for.
SetMask	Set the list of events that the object will wait for.
WaitOnMultipleEvents	Waits for the events in the object's event mask for the specified time period or until one of the events occurs.

CreateWaitObject

See Chapter 8, "Session Object."

DestroyWaitObject

See Chapter 8, "Session Object."

DumpEventMask

The DumpEventMask method returns a printable string listing the events in the CWaitObject's mask.

Syntax

C++ , Java, .NET: `string DumpEventMask();`

Parameters

None.

Return Values

A printable string object listing the events in the wait mask.

GetMask

The GetMask method gets the list of events that the CWaitObject will wait for.

Syntax

C++: `bool GetMask(Arguments ** pMask);`
Java, .NET: `Arguments GetMask();`

Parameters

pMask

A pointer to an Arguments object pointer. GetMask will set the value of pMask to a pointer to an Arguments object that contains the event mask.

Return Values

If the method was able to get the mask it returns true; otherwise, it returns false. 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 will wait for.

Syntax

C++: `bool InMask(int iEventId);`
Java, .NET: `boolean InMask(int iEventId);`

Parameters

iEventId

The enumerated event ID of the event to check for.

Return Values

If the event is in the mask it returns true. Otherwise, it returns false.

SetMask

The SetMask method sets the list of events that the CWaitObject will wait for.

Syntax

C++: `bool SetMask(Arguments & args);`
Java, .NET: `boolean SetMask(Arguments rArgs);`

Parameters

args

A reference to an Arguments object containing the list of events to wait for. The Arguments should contain values where the keys are “Event1” through “EventN” and the values are the enumerated event IDs.

Return Values

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 which 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 will cause this method to wait infinitely.

Return Values

The WaitOnMultipleEvents method returns one of the values listed in Table 12-9.

Table 12-9 WaitOnMultipleEvents Return Values

Value	When Returned
`EVENT_SIGNALED`	If one of the events in the mask occurred.
`EVENT_WAIT_TIMEDOUT`	If the timeout period elapsed.
`WAIT_FAILED`	If unable to wait on the events in the mask.

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

Table 12-8 list the methods in the Logger class.

Table 12-10 CWaitObject Class Methods

Method	Description
AddLogListener	Registers a listener with the LogManager.
GetTraceMask	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

Table 12-10 CWaitObject Class Methods (continued)

Method	Description
Logger (Constructor)	Creates a Logger object and also a LogManager object if one does not already exist. If one has already been created, it just gets a reference to the existing singleton LogManager object.
RemoveLogListener	Unregisters a listener from the LogManager.
SetTraceMask	Sets the current trace mask.
Trace	Send a trace message to the central LogManager with the specified trace mask.

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 gets a reference to the existing singleton LogManager object.

Syntax

```
void Logger()
```

Parameters

None.

Return Values

None.

GetTraceMask

The GetTraceMask method gets the current trace mask.

Syntax

```
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 Refer to Table B-5 for additional information on trace masks available in the .NET CIL.

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(ILogListener 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 will send 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

Table 12-8 list the LogWrapper class methods.

Table 12-11 LogWrapper Class Methods

Method	Description
Dispose	Releases system resources used by the LogWrapper. (.NET only)
GetMaxDaysBeforeExpire (.NET Only)	Obtains the current log file age threshold beyond which the active log file will be rolled over into a new file regardless of file size
GetMaxFileSize(.NET only)	Obtains the current log file size threshold beyond which a new file will be created.
GetMaxNumberOfFiles (.NET Only)	Obtains the current number of log files threshold beyond which older files will begin to be deleted.
GetTraceMask	Gets the current trace mask.
LogWrapper() Constructor	Creates a new LogWrapper object that writes tracing messages to System.Console.Out.
LogWrapper (String fileName) Constructor	Creates a new LogWrapper object that traces to the file specified in the fileName parameter.
LogWrapper (string, int, int, int)	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 will be used.
SetMaxDaysBeforeExpire	Changes the current log file age threshold beyond which the active log file will be rolled over into a new file regardless of file size
SetMaxFileSize	Changes the current log file size threshold beyond which a new file will be created.
SetMaxNumberOfFiles	Changes the current number of log files threshold beyond which older files will begin to be deleted.
SetTraceMask	Sets the current trace mask.
UpdateTraceSettings	Parses TraceConfig.cfg and imports the settings contained within.
WriteTraceLine	Prints a string to the active trace file or to System.Console.Out if no active trace file exists.

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 the LogWrapper is no longer needed.

Syntax

```
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 the LogWrapper is no longer needed.

Syntax

```
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 will be used. If you are using the .NET CIL, call the Dispose method to release system resources when the LogWrapper is no longer needed.

Syntax

```
JAVA:  void LogWrapper (string sFileName, long iMaxSize, int iArchives, int
iFlushIntervalMs)
.NET:  void LogWrapper (string sFileName, int maxSizeKB, int maxFiles, int
daysUntilExpiration)
```

Parameters

.NET and Java:sFfileName

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

```
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 will be rolled over into a new file regardless of file size.

Syntax

```
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 will begin to be deleted. If the provided value is not greater than zero, the default value of 4 is used.

Syntax

```
.NET:void SetMaxNumberFiles (int maxFiles)
Java:void SetMaxNumberFiles (int iArchives)
```

Parameters

maxTraceFiles

New number of files threshold. If 0 is specified, the default value will be used

Return Values

None.

GetMaxNumberFiles (.NET Only)

The GetMaxNumberFiles method gets the current number of log files threshold beyond which older files will begin to be deleted.

Syntax

```
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 will be rolled over into a new file regardless of file size.

Syntax

```
Java:void SetMaxDaysBeforeExpire (int iExpires)
.NET: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

```
boolean ProcessConfigFile()
```

Parameters

None.

Return Values

Returns true if operation succeeded and false if unable to open theTraceConfig.cfg file.

CHAPTER 13

SilentMonitorManager Object

The SilentMonitorManager Object provides developers with an interface to silent monitor behavior. The SilentMonitorManager object exposes methods to perform all silent monitor tasks, such as starting, stopping and managing silent monitor sessions. The SilentMonitorManager object stores specific silent monitor session information as properties.

The SilentMonitorManager object can be used in two different modes:

- In Monitoring Mode, an application that wants to silent monitor conversation without being noticed by the calling parties must create a SilentMonitorManager object and set it mode to eSMMonitoringMode using the StartSMMonitoringMode method.
- In Monitored Mode, an application will accept requests to initiate silent monitor sessions to forward the voice conversations to the remote monitoring application. The application will create a SilentMonitorManager object and set the mode to eSMMonitoredMode using the StartSMMonitoredMode method.

For more information on these modes see the section "Initiating and Ending a Silent Monitor Session" in Chapter 4, "Building Your Application."

Note SilentMonitorManager Object methods and properties are not available in the Java or .NET CILs. SilentMonitorManager Object methods and properties are supported for use with Unified CCE only. SilentMonitorManager Object methods and properties are only supported for CTI OS based silent monitoring.

Properties

Table 13-1 lists the SilentMonitorManager object properties.

Table 13-1 SilentMonitorManager Object Properties

Keyword	Type	Description
HeartbeatInterval	INT	Heartbeat interval for the silent monitor session.
HeartbeatTimeout	INT	Timeout for no activity.
IPPhoneInformation	ARGUMENTS	This property is only accessible via the GetIPPhoneInfo method. It contains all the information related to the IP Phone used by the application.
MediaTerminationPort	INT	TCP/IP port where monitored conversation will be sent for playback on system sound card.
SessionInformation	ARGUMENTS	This property is only accessible via the GetSessionInfo method. It contains all the information related to the current active silent monitor session.
SMManagerMode	SHORT	Mode in which the manager object will operate (Table 13-2). If SetIPPhoneInfo is used, SMManagerMode attempts to use the information provided. **Note** Only applies to CTI OS based Silent Monitor.

Table 13-2 SMManagerMode Values

enum Value	Description	Numeric Value
eSMModeNotSet	Mode not set.	-1
eSMMonitoredMode	The manager accept request for silent monitor sessions and forward voice to the monitoring application.	0
eSMMonitoringMode	The manager can make requests to remote client to start a silent monitor session and send voice.	1

Methods

Table 13-3 lists the SilentMonitorManager object methods.

Table 13-3 SilentMonitorManager Object Methods

Method	Description
AcceptSilentMonitoring	Establishes a silent monitor session and immediately starts sending audio.
GetIPPhoneInfo[1]	Retrieves the information of the IP Phone used by the client application. Gets its information from the RTP events that occur when RTP streams are created and modified.
GetSessionInfo	Retrieves the information related to the current silent monitor session.
GetSMSessionList	Retrieves a list of all active Silent Monitor sessions.
IsMonitoredTarget	Determines if the device/agent is a target being monitored.
SetIPPhoneInfo[1]	Saves the information of the IP Phone used by the client application.
StartSilentMonitorRequest	Sends a silent monitor session start request to a targeted client.
StartSMMonitoredMode	Puts the SilentMonitorManager in Monitored mode.
StartSMMonitoringMode	Puts the SilentMonitorManager in Monitoring mode.
StopSilentMonitorMode	Sets the SilentMonitorManager mode to eSMModeNotSet. If a silent monitor session is active at this time, the session is terminated.
StartSilentMonitorRequest	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.
StopSilentMonitorRequest	Stops the active silent monitor session.

1. GetIPPhoneInfo and SetIPPhone Info are used by SilentMonitorManager in the following manner. The RTPStartedEvent arrives and SilentMonitorManager uses SetIPPhoneInfo 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. SetIPPhoneInfo is used internally by the SilentMonitorManager to populate IP phone information carried in RTPStartedEvents.

Argument Parameter Rules

The following rules apply to the optional_args and reserved_args parameters in SilentMonitorManager Object methods:

- In VB, you can ignore these parameters altogether. For example, you can treat the line:

  ```
  StartSMMonitoringMode([reserved_args As IArguments]) As Long
  ```

  as follows:

  ```
  StartSMMonitoringMode()
  ```

- To ignore these parameters in COM you must send a NULL, as shown:

  ```
  StartSMMonitoringMode(NULL)
  ```

AcceptSilentMonitoring

The AcceptSilentMonitoring method establishes the silent monitor session requested by the OnSilentMonitorRequestedEvent and immediately starts sending audio to the monitoring client. This method should only be used if the parameter DoDefaultMessageHandling was set to False when the subscriber handled the OnSilentMonitorRequestedEvent event.

Syntax

```
C++: int AcceptSilentMonitoring(Arguments & args );
COM:  HRESULT AcceptSilentMonitoring ( /*[in]*/ IArguments * args, /*[out,retval]*/ int *
errorcode );
VB: AcceptSilentMonitoring (ByVal args as CTIOSCLIENTLIB.IArguments) As Long
```

Parameters

args

Arguments array that contains the parameters listed in Table 13-4:

Table 13-4 AcceptSilentMonitoring Arguments Array Parameters

Keyword	Type	Description
MonitoredUniqueObject ID	STRING	Unique Object ID of the object being monitored.
MonitoringIPAddress	STRING	TCP/IP address of the monitoring application.
MonitoringIPPort	INT	TCP/IP port of the monitoring application.
SMSessionKey	UNSIGNED SHORT	Unique identifier for the Silent Monitor Session.
HeartbeatInterval	INT	Heartbeat interval for the silent monitor session.
HeartbeatTimeout	INT	Timeout for no activity.

Table 13-4 AcceptSilentMonitoring Arguments Array Parameters (continued)

Keyword	Type	Description
OriginatingServerID	STRING	TCP/IP Address:Port of the CTI OS server from which the request originated.
OriginatingClientID	STRING	Client Identification of the monitoring application.
DoDefaultMessage Handling	BOOLEAN	When this parameter is set to True, it instructs the SilentMonitorManager to immediately start sending audio and establish the silent monitor session. If this value is set to False, it instructs the SilentMonitorManager to not send voice and not to establish the silent monitor session. It is then the responsibility of the subscriber to report this status accordingly.

errorcode

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

Return Values

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

GetIPPhoneInfo

The GetIPPhoneInfo method gets the information about the client application IP Phone.

Note This method is not necesary to use and defaults to figuring out the information to sniff packets from.

Syntax

```
C++: Arguments *  GetIPPhoneInfo(void);
COM:  HRESULT GetIPPhoneInfo ( /*[out,retval]*/ IArguments ** IPPhoneInfo);
VB: GetIPPhoneInfo () as CTIOSCLIENTLIB.IArguments
```

Parameters

None.

Return Value

This method returns an arguments array that contain the parameters listed in Table 13-5.

GetSessionInfo

The GetSessionInfo method retrieves the information related to the current silent monitor session.

Syntax

```
C++: Arguments * GetSessionInfo(Arguments & args) ;
COM:  HRESULT GetSessionInfo ( /*[in]*/ IArguments * args, /*[out,retval]*/ IArguments *
SMSessionInfo );
VB: GetSessionInfo (ByVal args as CTIOSCLIENTLIB.IArguments) As CTIOSCLIENTLIB.IArguments
```

Parameters

args

Arguments array that contains one of the parameters listed in Table 13-6:

Return Values

This method returns an arguments array containing the key/value pairs listed in Table 13-7:

Table 13-7 GetSessionInfo Return Arguments Array Parameters

Keyword	Type	Description
SMSessionKey	UNSIGNED SHORT	Unique silent monitor session Object ID of the target object that is being monitored.
SMSessionStatus	SHORT	One of the ISilentMonitorEvent status codes in Table 6-87.
AudioMode	INT	Reserved. Specifies the audio mode bitmask.
AgentID/DeviceID	STRING	Agent ID or DeviceID of the target being monitored.
MonitoredUniqueObjectID	STRING	Unique Object ID of the target object being monitored.
MonitoredDeviceIPAddress	STRING	TCP/IP Address of the monitored IP Phone.
PeripheralID	INT	ID of the peripheral associated with the agent and IP phone.
MonitoringIPAddress	STRING	TCP/IP Address of the system receiving audio.
MonitoringIPPort	INT	TCP/IP port on which receiving system is listening for audio.

GetSMSessionList

The GetSMSessionList method returns an Arguments array that contains the parameters listed in Table 13-10. 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++: bool IsMonitoredTarget (Arguments & args);
COM:  HRESULT IsMonitoredTarget ( /*[in]*/ IArguments * args, /*[out,retval]*/
VARIANT_BOOL * bMonitored );
VB: IsMonitoredTarget () As Boolean
```

Parameters

args

Arguments array that contains the parameter listed in Table 13-8:

Table 13-8 IsMonitoredTarget Arguments Array Parameter

Keyword	Type	Description
MonitoredUniqueObjectID	STRING	Unique Object ID of the target object being monitored.

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.

The SetIPPhoneInfo() function is used 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.

Note AgentState control only uses SetIPPhoneInfo in a Citrix environment. Since the IP address/port that the phone is connected to is not local to the Citrix server, the AgentState control uses terminal services APIs to figure out the IP address of the real network interface.

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 Table 13-9:

Table 13-9 SetIPPhoneInfo Arguments Array Parameters

Keyword	Type	Description
ClientAddress (required)	STRING	IP Address of the IP Phone to be used by the client application.
BitRate (optional)	INT	Audio transmission bit rate.
PacketSize (optional)	INT	Number of milliseconds of audio stored in a packet.
Direction (optional)	SHORT	One of the following values that indicates the direction of voice flow between the calling party and the called party: 0: Input 1: Output 2: Bidirectional
RTPType (optional)	SHORT	One of the following values that indicates the type of RTP messages between the calling party and the called party: 0: audio 1: video 2: data
EchoCancelation (optional)	SHORT	One of the following values that indicates whether the echo cancellation feature is enabled on this IP Phone: 0: Off 1: On
PayLoadType (optional)	SHORT	Audio codec type.

errorcode

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

Return Values

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

StartSilentMonitorRequest

The StartSilentMonitorRequest method sends a silent monitor session start request to a targeted client

Syntax

```
C++: int StartSilentMonitorRequest (Arguments & args, unsigned short * SMSessionKey );
COM:  HRESULT StartSilentMonitorRequest ( /*[in]*/ IArguments * args, /*/[out]*/ unsigned
short * SMSessionKey, /*[out,retval]*/ int * errorcode );
VB: StartSilentMonitorRequestInt (ByVal args as CTIOSCLIENTLIB.IArguments, ByRef
SMSessionKey AsLong) As Long
```

Parameters

args

Arguments array that contains the parameters listed in Table 13-10. All parameters are required.

Table 13-10 StartSilentMonitorRequest Arguments Array Parameters

Keyword	Type	Description
AgentID or DeviceID	STRING	AgentID or DeviceID of the target to monitor. Specify either an AgentID or a DeviceID, not both,
PeripheralID	INT	ID of the peripheral associated with the agent or device.
MonitoringIPAddress	STRING	TCP/IP address of the system receiving audio.
MonitoringIPPort (Optional)	INT	TCP/IP port where the monitoring application is listening for audio.
HeartbeatInterval	INT	Interval in seconds between heartbeats.
HeartbeatTimeout	INT	Seconds elapsing before a Silent Monitor session is aborted because of no heartbeats received from the monitored peer.

SMSessionKey

An output parameter that contains the unique key to the started silent monitor session. This key must be used to perform any action on the currently active silent monitor session.

errorcode

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

Return Values

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

Remarks

If the DeviceID is used, there must be an agent associated with the device. The session will timeout if there is no agent logged into the device. An established silent monitor session will end if the associated agent logs out of the device.

E_CTIOS_INVALID_SILENT_MONITOR_MODE is returned when SilentMonitorManager.Start SilentMonitorRequest() 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.StartSilentMonitor
Request(), the application receives an OnSilentMonitorStatusReportEvent carrying a status code of eSMStatusCCMSilentMonitor.

StartSMMonitoredMode

The StartSMMonitoredMode method puts the SilentMonitorManager in Monitored mode.

Syntax

```
C++: int StartSMMonitoredMode (Arguments & args );
COM:  HRESULT StartSMMonitoredMode ( /*[in]*/ IArguments * args, /*[out,retval]*/ int *
errorcode );
VB: StartSMMonitoredMode (ByVal args as CTIOSCLIENTLIB.IArguments) As Long
```

Parameters

args

Arguments array that contains the following parameters listed in Table 13-11:

errorcode

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

Return Values

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

StartSMMonitoringMode

The StartSMMonitoringMode method puts the SilentMonitorManager in Monitoring mode.

Syntax

```
C++: int StartSMMonitoringMode (Arguments & args );
COM:  HRESULT StartSMMonitoringMode ( /*[in]*/ IArguments * args, /*[out,retval]*/ int *
errorcode );
VB: StartSMMonitoringMode (ByVal args as CTIOSCLIENTLIB.IArguments) As Long
```

Parameters

Table 13-12 StartSMMonitoringMode Arguments Array Parameters

Keyword	Type	Description
SMSAddr	STRING	A string that contains the address of the silent monitor service used to decode and play back the agent’s phone call.
SMSListenport	INT	The port on which the silent monitor services listen for connections.
SMSTOS	INT	The QoS setting for the connection.
SMSHeartbeats	INT	The interval in milliseconds between heartbeat packets.
SMSRetries	INT	The number of heartbeats that can be missed before the connection is aborted.

errorcode

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

Return Values

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

StopSilentMonitorMode

The StopSilentMonitorMode method sets the SilentMonitorManager mode to eSMModeNotSet. If a silent monitor session is active at the time, the session will be terminated.

Syntax

```
C++: int StopSilentMonitorMode (Arguments & reserved_args );
COM:  HRESULT StopSilentMonitorMode ( /*[in]*/ IArguments * reserved_args,
/*[out,retval]*/ int * errorcode );
VB: StopSilentMonitorMode (ByVal reserved_args as CTIOSCLIENTLIB.IArguments) As Long
```

Parameters

reserved_args

Not currently used, reserved for future use.

errorcode

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

Return Values

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

StopSilentMonitorRequest

The StopSilentMonitorRequest method stops the Active silent monitor session.

Syntax

```
C++: int StopSilentMonitorRequest (Arguments & args);
COM:  HRESULT StartSilentMonitorRequest ( /*[in]*/ IArguments * args, /*[out,retval]*/
int * errorcode );
VB: StopSilentMonitorRequest (ByVal args as CTIOSCLIENTLIB.IArguments) As Long
```

Parameters

args

Arguments array that contains the parameter listed in Table 13-13:

Table 13-13 StopSilentMonitorRequest Arguments Array Parameters

Keyword	Type	Description
SMSessionKey	UNSIGNED SHORT	Unique key of the current active silent monitor session

errorcode

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

Return Values

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

APPENDIX A

CTI OS Keywords and Enumerated Types

Keywords

The CTI OS Client Interface Library uses the Arguments structure to pass key-value pairs between the client and the server (see Chapter 12, “Helper Classes” for a detailed explanation of Arguments). Throughout this document all event and method parameter lists, as well as object properties, are listed with the keywords and the types associated with those keywords.

The expected (required and optional) keywords are referred to in this document by string name. For example, the Agent’s property for agent identifier is referred to as *AgentID*.

In addition to being able to use the string name for a keyword, programmers can take advantage of an enumeration of keywords as well.

Note The enumeration of keywords is presently only available in the C++ CIL.

For each string keyword, a corresponding enumerated keyword exists. The enumerated keyword is the same name, preceded by the prefix ‘ekw’. For example, the *AgentID* string keyword is mapped to the enumerated keyword *ekwAgentID*.

Usage Example in C++:

```
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 can be found in the C++ header file "*ctioskeywords.h*", located in the *\Distribution\cpp\Include* directory on the CTI OS toolkit media.

Java CIL Keywords

For Java CIL, the CtiOs_IKeywordIDs interface contains a list of known Java CIL CTI OS keywords. See the Java CIL Javadoc file for more information.

.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. See the next section, "Java Interfaces".

The complete set of enumerated types and their values can be found in the following locations:

- For C++ CIL using static libraries: the complete set of enumerated types is located in the C++ header file "*cilmessages.h*", located in the *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. See the Java CIL Javadoc file for more information.

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 is 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

```
enum enumCTIOS_AgentState
{
    eLogin = 0,
    eLogout = 1,
    eNotReady = 2,
    eAvailable = 3,
    eTalking = 4,
    eWorkNotReady = 5,
    eWorkReady = 6,
    eBusyOther = 7,
    eReserved = 8,
    eUnknown = 9,
    eHold=10

 };
```

is implemented in the Java CIL as follows:

```
public interface CtiOs_IAgentState
{
    public static final inteLogin = 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:

```
public class MyAgent extends CtiOsObject implements CtiOs_IAgentState
{

      ................................................


      public int MyLogin(Arguments rArguments)
     {
    .................................
                //Access eLogin directly
rArguments.AddItemInt( "agentstate", eLogin );
    .................................

     }
}
```

or by referencing as follows:

```
public class MyAgent extends CtiOsObject
{

      ..................................................
     public int MyLogin(Arguments rArguments)
     {
    ..................................
rArguments.AddItemInt( "agentstate", CtiOs_IAgentState.eLogin );
    ..................................

     }
}
```

APPENDIX B

CTI OS Logging

This appendix discusses a few issues related to CTI OS logging.

Creating 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 can be found under the following Windows registry key:

```
HKEY_LOCAL_MACHINE\Software\Cisco Systems, Inc.\CTI Desktop\CTIOS Tracing\<TraceFileName>
```

The default filename is CtiosClientLog. Log files are created using the convention <TraceFileName>.<Windows user name>.mmdd.hhmmss.log. The files are created in the current directory of the executing program, such as the directory into which the Agent Desktop is installed. You can provide a fully qualified path for the TraceFileName if you wish to store the files in a different location. For example, setting the following value causes the log files to be stored in the directory `C:`\Temp, using the naming convention CtiosClientLog.<Windows user name>.mmdd.hhmmss.log.

```
C:\Temp\CtiosClientLog
```

Client trace files are formatted in ASCII text that you can open with a conventional text editor such as Notepad.

How to Install the Tracing Mechanism (COM and C++)

To install the tracing mechanism:

Step 1 Copy the tracing executable, ctiostracetext.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:

ctiostracetext.exe /regserver

Setting Trace Levels (COM and C++)

You must set the tracel 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
                     "MaxDaysBeforeExpire"=dword:00000007
                     "MaxFiles"=dword:00000005
                     "MaxFileSizeKb"=dword:00000800
                     "FlushIntervalSeconds"=dword:0000001e
                     "TraceServer"="C:\\Program Files\\Cisco Systems\\CTIOS Client\\CTIOS
                      Toolkit\\Win32 CIL\\Trace\\CTIOSTraceText.exe"
```

Note CTI OS Tracing can be configured globally for the entire machine (using the TraceMask setting on HKLM) and per user (using the TraceMask setting on HKCU).

Warning **If the TraceMask not set or if it is set incorrectly, the application's performance may be negatively affected. The preferred setting for normal operation is 0x40000307.**

Configuring Tracing (COM and C++)

You can set C++ and COM client trace configuration parameters in the Windows registry at the following key. See Java CIL Logging Utilities, page B-3, for instructions on configuring tracing for the Java CIL. See Logging and Tracing (.NET), page B-5, for instructions on configuring tracing for the .NET CIL.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems\CTI Desktop\Ctios\Logging
```

These settings are defined as follows:

Table B-1   Configuring Tracing Settings

Parameter	Description	Recommended Value
FlushIntervalSeconds	Maximum number of seconds before the trace mechanism transfers data to the log file.	30
MaxDaysBeforeExpire	Maximum number of days before a log file is rolled over into a new log file regardless of the size of the file.	7

Parameter	Description	Recommended Value
MaxFiles	Maximum number of log files that may exist in the log file directory before the logging mechanism starts overwriting old files.	5
MaxFileSizeKb	Maximum size of a log file in kilobytes. When a log file reaches the maximum size, a new log file is created.	2048
TraceMask	Bit mask that determines the categories of events that are traced.	0x40000307

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 will only need 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 only has one method.

```
void processLogEvent(LogEvent event)
```

LogEvent

A custom application that is interested in receiving LogEvents will receive an object of this type whenever a log message is generated. This class extends the Java “EventObject”, and has one public method:

Table B-2 LogEvent Method Description

Method	Description
getDescription	Returns the text description to write somewhere.

```
String getDescription()
```

Logger

A custom application that is interested in firing or handling its own LogEvents, can create an instance of this class.

Table B-3 Logger Method Description

Method	Description
Logger	Public constructor of the Logger object.
Trace	Lets the custom app fire a LogEvent
GetTraceMask	Gets the trace mask.
IsTraceMaskEnabled	Determines if a certain trace mask is set.
addLogListener	Subscribe to receive LogEvents.
removeLogListener	Unsubscribe from receiving LogEvents.

Syntax

```
Logger()
int Trace(long nMsgTraceMask, String message)
long GetTraceMask()
boolean IsTraceMaskEnabled(long nMsgTraceMask)
void addLogListener(ILogEvents logEvents
```

where logEvents implements the ILogEvents interface.

```
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 doesn't want to implement its own ILogEvents interface can create an instance of this class. The adapter class provides two constructors, a default one that will automatically log to the Java console and one that takes in an output filename.

Table B-4 LogEventsAdaptor Descriptions

Method	Description
LogEventsAdapter	Public constructor
startLogging	Start receiving LogEvents
stopLogging	Stop receiving LogEvents
processLogEvent	Handles a LogEvent
finalize	Does some cleanup

Syntax

```
LogEventsAdapter()
LogEventsAdapter(String fileName)
void startLogging()
void stopLogging()
void processLogEvent(LogEvent e)
void finalize()
```

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 sFileName) - Creates a new LogWrapper object that writes trace messages to the file specified in sFileName.
- LogWrapper(string sFileName, long iMaxSize, int iArchives, int iExpires, int iFlushIntervalMs) - Creates a new LogWrapper object that traces to the file specified in sFileName 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.

Note You will also need to provide a way to adjust the trace mask at runtime. If you encounter problems, Cisco personnel will need to see this log file in order to assist you with your problem.

See the Java CIL Javadoc file for information on the LogWrapper class and its associated methods.

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.

Using the 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.

How to Log Trace Events Using the LogWrapper Class

To log trace events using the LogWrapper class:

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 may 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.

```
// 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. See Table B-1 for a list of available trace mask values.

```
    protected internal static void MyTrace (
        int traceLevel,
        string methodName,
        string msg)
        {
            if ( m_logger.IsTraceMaskEnabled(traceLevel) )
            {
                string tracsMsg = string.Format("{0}: {1}", methodName,
                    msg) ;
                m_logger.Trace(traceLevel, msg) ;
            }
        }
```

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.

Table B-1 lists the trace masks available in the .NET CIL.

Table B-5 Trace Masks in .NET CIL

Creating a 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 must be implemented by custom applications:

```
public delegate void LogEventHandler(object eventSender, LogEventArgs args);
```

How to Log Trace Events Using the Logger Class

To log trace events from a custom application to a file, perform the following steps:

Step 1 Create a Logger object. For example:

```
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:

```
public virtual void ProcessLogEvent(Object eventSender, LogEventArgs Evt)
{
    // Output the trace
    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 will send trace events to the method that you pass to the LogEventHandler.

In the following example, the LogManager will send trace events to the ProcessLogEvent method created in Step 2.

```
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.

Configuring Tracing (Java and .NET)

For the Java and .NET CILs, you can configure tracing either programmatically by using the LogWrapper class or by editing the TraceConfig.cfg file. Settings in TraceConfig.cfg will 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 how to process that file.

Log file configuration settings are defined as follows

Table B-6 Configuration Settings

:

Parameter	Description	Recommended Value
NumberDaysBeforeFileExpired	Maximum number of days before a log file is rolled over into a new log file regardless of the size of the file.	1
NumberArchivesFiles	Maximum number of log files that may exist in the log file directory before the logging mechanism starts overwriting old files.	5
FileMaxSize	Maximum size of a log file in kilobytes. When a log file reaches the maximum size, a new log file is created.	2048
TraceMask	Bit mask that determines the categories of events that are traced.	0x40000307

APPENDIX C

Migrating From CTI OS 7.1(x) or 7.2(x)

Introduction

Cisco CTI OS Toolkit Release 8.0(1) is the official release in which Microsoft Visual Studio .NET 2005 is supported and with which it is possible to build Win32 and.NET 2.0applications.

In Cisco CTI OS Toolkit 8.0(1), the C++ CIL static libraries are targeted to be used ONLY with Microsoft's C\C++ 8.0 compiler or equivalent. Therefore, these libraries will not work if used with an earlier version of Microsoft Visual Studio (for example, Visual Studio 6.0, .NET 2003). COM CIL 8.0(1) and the CTI OS ActiveX 8.0(1) controls are targeted for use in development environments that support OLE Automation, ActiveX and COM.

For existing applications built with earlier versions of the Cisco CTI OS Toolkit it is necessary to upgrade their build projects to include new parameters required by the new compilers and the new features in the release. The following sections describe the migration steps and the new parameters required to build the application.

The goal of the migration procedures described below is to guide the developer on porting his/her current application projects to the equivalent project model under Visual Studio .NET 2005and not a guide on how to port an unmanagedWin32 or COM application to native .NET 2.0.

Migrating a C++ CIL application

For a client application to be upgraded to use C++ CIL 8.0(1) the following tasks need to be executed in order for the application to build with the new version.

Using your development environment utilities port your current application build project to the new platform compatible with Microsoft C\C++ and made all the adjustments required by the vendor. For example, to port your Microsoft Visual C++ 7.0 *.vcproj project file to the new Microsoft Visual C++ 8.0 *.vcproj format, open the *.vcproj project selecting Open\ Project from the File command menu in Microsoft Visual Studio .NET 2005. For details refer to Microsoft's Visual Studio.NET help on Visual C++ Porting and Upgrading.

Note See Project Settings for Compiling and Linking, page 4-15.

Migrating a COM CIL Application

Migrating a C++ Application that uses COM CIL

A C++ application that uses COM CIL is considered an unmanaged C++ application under Visual Studio .NET.

Step 1 Port Build Project

Using your development environment utilities port your current application build project to the new platform compatible with Microsoft C\C++ 8.0(1) and made all the adjustments required by the vendor. For example, to port your Microsoft Visual C++ 7.0 *.vcproj project file to the new Microsoft Visual C++ 8.0 *.vcproj format, open the *.dsp project selecting Open\ Project from the File command menu in Microsoft Visual Studio .NET 2005. For details refer to Microsoft's Visual Studio.NET help on Visual C++ Porting and Upgrading (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore/html/_vc_porting_home.asp)

Step 2 Set New Compilation Parameters

Note See Project Settings for Compiling and Linking, page 4-15.

Migrate to Visual Basic .NET and use .COM CIL

For applications that the require to use COM CIL in a Visual Basic .NET application, it provides a set of Primary Interop Assemblies (PIAs) to allow the VB.NET code call interact with COM CIL objects and events. The interop assemblies for COM CIL are installed in the Global Assembly Cash (GAC) by the Cisco CTI OS 7.1(1) setup program.

Step 1 Port Build Project

Using your development environment utilities port your current application build project to the new platform compatible with Visual Basic .NET and made all the adjustments required by migration tool. For example, to port the Microsoft Visual Basic .NET2003*.vbproj project to the new Microsoft Visual Basic .NET.vbproj format, open the *.vbproj project selecting Open\ Project from the File command menu in Microsoft Visual Studio .NET 2005. For details refer to Microsoft's Visual Studio.NET help on Visual Basic Concepts.

Step 2 Verify references to COM CIL Primary Interop Assemblies

After the migration toll finishes the new ported project must include references to the COM CIL interop assemblies provided in the CTI OS 8.0(1) Toolkit. Verify that the following assemblies are listed under the project references: Cisco.CTIOSCLIENTLib.dll and Cisco.CTIOSARGUMENTSLib.dll.

If these assemblies are not in your reference list remove those included by the migration tool and set the two assemblies described before

Step 3 Apply Ported Code Changes for Visual Basic .NET

Following the migration report apply all the suggested changes such that you code complies with the new Visual Basic.NET 2005 programming language. Be aware that there are deprecated classes in .NET 2.0 that your application may have to change to accommodate the new classes.

Step 4 Porting Event Handlers to Visual Basic .NET

Migrating a Visual Basic 6.0 to Use COM CIL

As of Release 8.0(1), Visual Basic 6.0 is no longer supported.

Migrate to Visual Basic .NET and use CTI OS ActiveX Controls

For applications that use the CTI OS ActiveX Controls, it was provided a set of Runtime Callable Wrapper Assemblies (RCWs) that will allow the NET Windows Forms and Visual Basic .NET to interact with the CTI OS ActiveX controls. The RCW assemblies are installed in the Global Assembly Cash (GAC) by the Cisco CTI OS 8.0(1) setup program.

Step 1 Port Build Project

Using your development environment utilities port your current application build project to the new platform compatible with Visual Basic .NET and made all the adjustments required by migration tool. For example, to port the Microsoft Visual Basic .NET 2005*.vbproj project to the new Microsoft Visual Basic .NET 2005.vbproj format, open the *.vbproj project selecting Open\ Project from the File command menu in Microsoft Visual Studio .NET. For details refer to Microsoft's Visual Studio.NET help on Visual Basic Concepts

Step 2 Verify references to the CTI OS ActiveX Controls RCW Assemblies

After the migration toll finishes the new ported project must include references to the CTI OS ActiveX controls RCW assemblies provided in the CTI OS 8.0(1) Toolkit. Verify that the assemblies for the controls used in your application are listed under the project references. For each control in references there must be two entries for each control one prefixed with AxInterop.ControlName and Interop. ControlName. The RCW starting with AxIntero.p is responsible for hosting the ActiveX control on the .NET Windows Form and the RCW prefixed with Interop. Allows the Visual Basic .NET code to use the properties and methods exported by the control.

Step 3 Apply Ported Code Changes for Visual Basic .NET

Following the migration report applies all the suggested changes such that you code complies with the new Visual Basic.NET 2005 programming language.

Step 4 Porting Event Handlers to Visual Basic .NET

INDEX