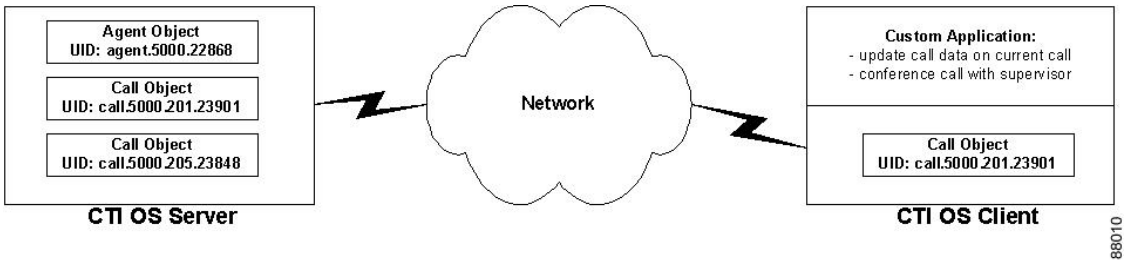
CTI OS Client Interface Library Architecture

Object Server Architecture

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

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

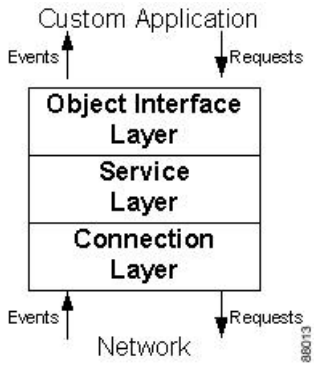
Figure 1: CTI OS Object Server and Client Object Sharing



Client Interface Library Architecture

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

Figure 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 to detect network-level failures only when the connection is silent for a period of time.

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

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

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

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

Service Layer

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

A secondary purpose of the Service Layer is to isolate the client from the network, so that network issues do not block the client and vice versa. This is done via a multi-threading model that allows user-program execution to continue without having to block on network message sending or receiving. This prevents client applications

from getting stuck when a message is not immediately dispatched across the network, and allows messages to be received from the network even if the client application is temporarily blocked.

Object Interface Layer

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

Custom Application

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

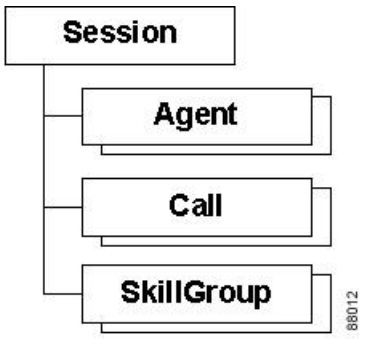
- The CIL provides the object-based interface for interacting with CTI OS, to send requests for agent and call control.
- The CIL provides an events subscription service, which the custom application takes advantage of to receive events from CTI OS.

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

CIL Object Model Object Interfaces

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

Figure 3: 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 the other objects in the CIL.

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

Session Modes

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

Agent Mode

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

Monitor Mode

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

For the complete interface specification of the Session object, see 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 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 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 SkillGroup Object

Object Creation

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

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

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

Reference Counting

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

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

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

Call Object Lifetime

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

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

Agent Object Lifetime

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

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

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

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

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

SkillGroup Object Lifetime

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

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

Methods That Call AddRef()

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

Table 1: SessionLib (C++)

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 is decremented.

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	DestroySilentMonitorManager()	This method decrements the reference count of the passed in object.
CCtiOsSession	CreateSilentMonitorManager()	The client application must call DestroySilentMonitorManager () on the returned object when it is no longer needed.
CCtiOsSession	SetCurrentSilentMonitor()	This method increments the reference count on the passed in object. The previous "current" silent monitor's reference count is decremented.
CCtiOsSession	GetCurrentCall(), GetCurrentSilentMonitorManager(), GetAllCalls(), GetAllSkillGroups(), GetAllAgents(), GetCurrentAgent(), GetValue(), GetObjectFromObjectID()	The client application must call Release() on the returned object when it is no longer needed.
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.

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

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

Object Name	Method Name	Explanation
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.
ISilentMonitorManager	GetMonitor()	The client application must call Release() on the returned object when it is no longer needed.
ISilentMonitorManager	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.

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	GetObjectFromObjectID()	The client application must call Release() on the returned object (second argument) when it is no longer needed.
ISession	CreateSilentMonitorManager()	The client application must call DestroySilentMonitorManager() on the returned object when it is no longer needed.
ISession	DestroySilentMonitorManager()	This method call decrements the reference count on the passed in object.
ISession	GetCurrentSilentMonitorManager()	The client application must call Release() on the returned object when it is no longer needed.

Table 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.

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 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 CIL Coding Conventions
- For information about building an application using the CIL, see Building Your Custom CTI Application

- For a description and syntax of the CIL programming interfaces, see Chapters 8 through 13.