CISCO SYSTEMS

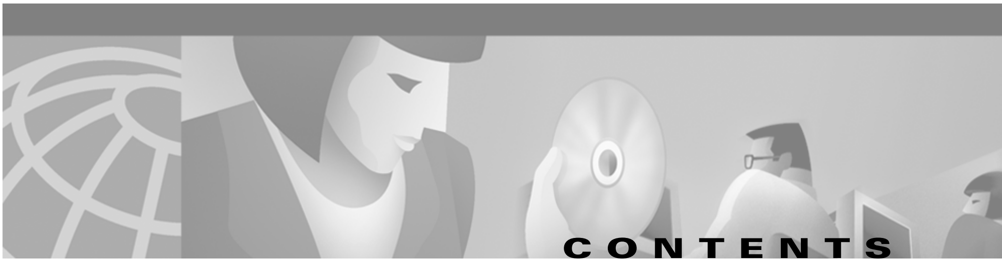
Cisco ICM Software CTI OS Developer's Guide

ICM Software Version 4.6

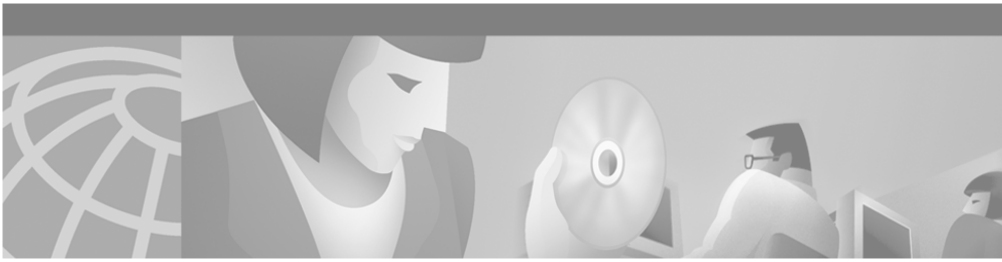
Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

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

Cisco ICM Software CTI OS Developer's Guide



CONTENTS



About This Guide

Purpose

This manual provides an overview of the Cisco CTI Object Server (CTI OS) product, describes the CTI OS Client Interface and CTI OS Architecture, and provides syntax and descritions for CTI OS methods and events.

Audience

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

Conventions

This manual uses the following conventions.

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 *Cisco ICM Software Database Schema Handbook.*
An arrow (>) indicates an item from a pull-down menu.	The Save command from the File menu is referenced as **File > Save**.

Organization

The manual is divided into the following chapters.

Chapter	Description
Chapter 1, "Introduction"	Provides an overview of the CTI OS Client Interface.
Chapter 2, "CTI OS Client Interface Library Architecture"	Discusses CTI OS architecture.
Chapter 3, "Handling Events"	Discusses CTI OS event handling.
Chapter 4, "Session Object"	Describes the methods and events associated with the CTI OS Session object.

Chapter	Description
Chapter 5, "Agent Object"	Describes the methods and events associated with the CTI OS Agent object.
Chapter 6, "Call Object"	Describes the methods and events associated with the CTI OS Call object.
Chapter 7, "SkillGroup Object"	Describes the methods and events associated with the CTI OS SkillGroup object.
Chapter 8, "Helper Classes"	Describes the methods and events associated with the CTI OS Arguments classes.
Appendix A, "CTI OS CIL Messages"	Lists the CTI OS messages.
Appendix B, "CTI OS Keywords"	Lists the CTI OS keywords.

Other Publications

For additional information about Cisco Intelligent Contact Management (ICM) software and Cisco Computer Telephony Integration (CTI) products, see the Cisco web site listing ICM and CTI documentation.

Obtaining Documentation

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

World Wide Web

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

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

Documentation CD-ROM

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

Ordering Documentation

Cisco documentation is available in the following ways:

- Registered Cisco Direct Customers can order Cisco Product documentation from the Networking Products MarketPlace:

  http://www.cisco.com/cgi-bin/order/order_root.pl

- Registered Cisco.com users can order the Documentation CD-ROM through the online Subscription Store:

  http://www.cisco.com/go/subscription

- Nonregistered Cisco.com users can order documentation through a local account representative by calling Cisco corporate headquarters (California, USA) at 408 526-7208 or, in North America, by calling 800 553-NETS(6387).

Documentation Feedback

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

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

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

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

We appreciate your comments.

Obtaining Technical Assistance

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

Cisco.com

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

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

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

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

http://www.cisco.com

Technical Assistance Center

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

Contacting TAC by Using the Cisco TAC Website

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

http://www.cisco.com/tac

P3 and P4 level problems are defined as follows:

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

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

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

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

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

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

Contacting TAC by Telephone

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

http://www.cisco.com/warp/public/687/Directory/DirTAC.shtml

P1 and P2 level problems are defined as follows:

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

Introduction

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

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

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

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

- Automatic connection failover.
- The architecture supports multiple threads and is thread safe. An application can spawn multiple threads using a single connection (session) object.
- Applications can be deployed on virtually any platform and any host container, such as VB, PowerBuilder, or Siebel.

Overview

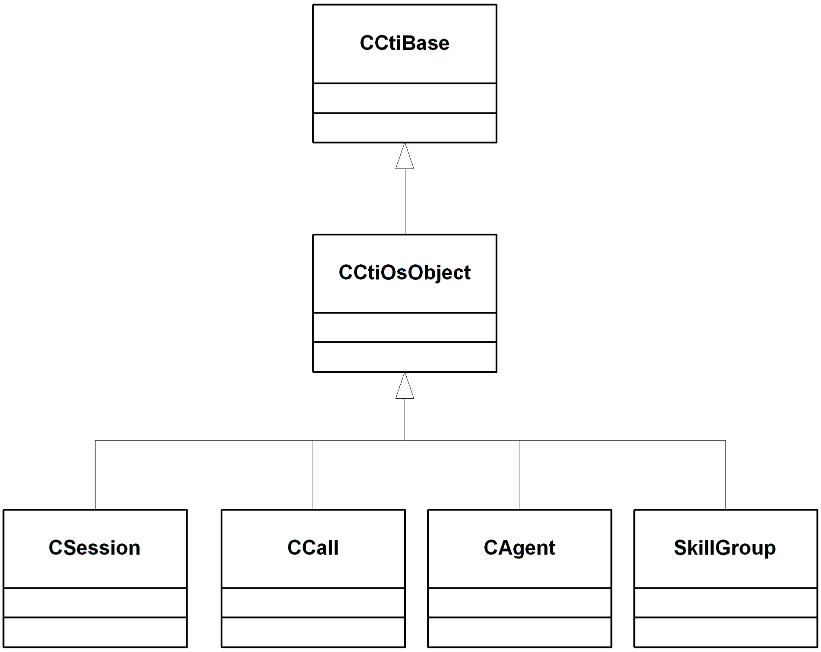
Clients connect to CTI OS through a client interface library. The library is responsible for establishing and automatically recovering connection failures, even if it must switch between CTI OS servers. It provides a set of wrapper classes to support COM, C++, and C applications.

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

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

The client application interacts with CTI OS through Session, Agent, Skill Group, and Call objects. The object model is shown in Figure 1-1.

Figure 1-1 CTI OS Client Interface Object Model



Agent and Monitor Mode connections apply regardless of the type of customer interaction. They apply to any and all type of interactions and serve to select the type of messages sent to the client application. To connect in Agent Mode, the client application must login as an agent. To connect in Monitor Mode, the application specifies a message filter.

To connect:

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

Sub ConnectToServer( )

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

       'Builds connect parameters
        Dim m_Args As New Arguments
       ‘Side A information
        m_Args.AddItem "CtiOsA", machinename_A
        m_Args.AddItem "PortA", 42028
        ‘Side B information
        m_Args.AddItem "CtiOsB", machinename_B
        m_Args.AddItem "PortB", 43028
        m_Args.AddItem "Heartbeat", 5
       ‘Get ready to submit connection request
        Dim vRequestParam As Variant
        Set vRequestParam = m_Args
        ‘sends connection request
        m_session.Connect vRequestParam

End Sub
```

To set Agent Mode:

```
Sub EstablishAgentMode()
           Dim m_Args As New Arguments

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

            ' Create a new agent
            Set m_Agent = New Agent
            ' intialize the agentobject using setvalue
            m_Agent.SetValue "Agentid", “23840”
            m_Agent.SetValue "Instrument", “23801”
            m_Agent.SetValue "PeripheralID", 5000

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

            Dim vRequestParam As Variant
            Set vRequestParam = m_Args

            ' Send login request
            m_Agent.Login vRequestParam

End Sub
```

To set Monitor Mode:

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

```
            strFilter  = "Agent=23840;Agent=23845;"

            'Tell the session about the filter to use to establish a
            'Monitor mode connection
            m_session.SetMessageFilter  strFilter

End Sub
```

Extensibility and Ease of Use

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

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

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

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

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

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

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

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

```
agent. MakeCall
"DN=" + agent_id + "; " + _
"CallVariable1=" + callvariable_item + "; " + _
"ExtDataArg=" + agent_identifier + "; " + _
"UserToUser=" + user_message

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

C++ has a slightly more flexible syntax:

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

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

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

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

Table 1-1 Client Interface Argument Types

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

Accessing Properties with GetValue

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

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

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

```
‘GetValue from Arg returns the correct value using the appropriate
data type
MsgBox “Agent Extension = “ & argValue.GetValue

Set argsAllCalls = m_session.GetValue(“Calls”)
```

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

COM interface:

```
Dim argValue As Arg
Dim argAllCalls as Arguments

Set argValue = m_Agent.GetValue(“Extension”)
Set argsAllCalls = m_session.GetValue(“Calls”)
```

C++ interface:

```
Arg & argValue = m_Agent.GetValue(EXTENSION);
argValue.Release();
Arguments & argAllCalls = (Arguments &) m_session.GetValue(“Calls”) ;
argAllCalls.Release();
```

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

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

The signature(s) for the GetValue method are:

COM interface:

```
VARIANT GetValue ([in] LPVARIANT vProperty)
```

C++ interface:

```
Arg &  GetValue(string & strProperty);
Arg &  GetValue(char * strProperty);
Arg &  GetValue(int strProperty);
```

GetElement

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

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

Property Names and Enumerated Constants

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

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

Property Attributes

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

COM Interface:

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

C++ Interface:

```
Arg&  CCtiOsObject::GetPropertyAttribute( string& strPropName,
enumCTIOS_Attribute nAttribute );
   Arg&  CCtiOsObject::GetPropertyAttribute( char * pstrzPropName,
enumCTIOS_Attribute nAttribute );
```

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


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

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

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

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

Table 1-2 Property Attributes

Attribute	Meaning
CTIOS_ATTRIBUTE_TYPE = 0	One of following values: CTIOS_DATATYPE_STRING, CTIOS_DATATYPE_INTEGER, CTIOS_DATATYPE_BOOLEAN, or CTIOS_DATATYPE_REFERENCE
CTIOS_ATTRIBUTE_LENGTH = 1	Current length of string type, otherwise 0.
CTIOS_ATTRIBUTE_SIZE = 2	Max size of string or size of data type. If string is of unlimited size, returns the value CTIOS_DATASIZE_ UNLIMITED = 32,767. Other defined types: CTIOS_DATASIZE_BOOL = 1, CTIOS_DATASIZE_INTEGER = 4

Table 1-2 Property Attributes (continued)

Attribute	Meaning
CTIOS_ATTRIBUTE_DEFINED_BY= 3	An identifier for which type of server connected to CTI OS has defined the properties of the constant. The only defined servers available at this time are: CTIOS_SERVERID_SYSTEM, CTIOS_SERVERID_CTISERVER The following names are reserved for future use: CTIOS_SERVERID_CISCOEMAIL CTIOS_SERVERID_CISCOCOLLAB CTIOS_SERVERID_UNDEF3RDPARTY
CTIOS_ATTRIBUTE_ISVALID = 4	Key. Returns true if the key is valid.
CTIOS_ATTRIBUTE_ISARRAY = 5	Returns true if value is an array or items.
CTIOS_ATTRIBUTE_NUMELEMENTS = 6	Number of elements in array or 1 if not array type.

Platform Issues

Among the different platforms supported, not all of them support unsigned integers or shorts and there exist variations in how boolean values are handled. This section addresses how these differences are handled.

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

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

CIL, boolean values are converted to (or from) VT_BOOL. COM applications such as VisualBasic, Internet Explorer, and other COM specific clients can treat TRUE as 0xFFFF.

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

Accessing Call and ECC Variables

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

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

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

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

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

   Set argsECCs =  call.GetValue(“ECC”)

   Set argECCVar = argsECCs.GetValue(“user.CustomerID”)

   Set argNamedArrayItem = argsECCs.GetValue(“user.AccountNumber[1]”)
```

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

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

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

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

Special keywords are available to retrieve an array of all call variables, all ECC variables, or a specific named ECC array. Examples:

```
Dim argsCallVars As Arguments
Dim argsECCVars As Arguments
' access arrays
Set argsCallVars = call. GetValue( eArrayCallVariables )
Set argsECCVars = call. GetValue( "ECC" )
Set argsECCVars = call. GetValue( "user. name_arrayvariable" )
```

Handling Errors

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

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

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

    int nRetcode = m_Session.Connect(arConnParams);

    if(CIL_FAILED(nRetCode))
    {
          switch(nRetCode)
          {
               case E_CTIOS_INVALID_PROPERTY:
               {
                       cout << "Invalid Parameter Received on 
Connection Parameters  " <<
                                  endl <<  arConnParams.DumpArgs();
               }
               break;
          }
    }

   //Continue working
•
•
•
}
catch(int * nExceptionCode)
{
      if(*nExceptionCode ==  E_CTIOS_ARGUMENT_ALLOCATION_FAILED)
     {
          cout << "CMyClass::MyMethod(): There is not much memory 
available " << endl
     }
}
//you need to release arConnParams after you are done with it
arConnParams.Release()
```

Event Driven Model

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

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

An event or response is not guaranteed. The operation may not result in any state change, may not create a telephony or other server response, may have failed outside in a link on the network outside of the CTI OS, or may have failed between the client application and CTI OS. The following guidelines and features will assist in diagnosing in real-time problems across servers:

Any request made that results in an error from CTI Server or other server are reported back to the client in the form of an OnError event.

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

CTI OS automatically monitors the quality and reliability of its connection to CTI Server and other servers using periodic heartbeats.

Event Cascade Model

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

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

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

Event Publication Model

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

Subscribing to an Event Interface

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

Special Values

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

Start Up, Connect, Snapshot

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

- The client connects to a CTI OS server using the Session object Connect() method.
- The client then establishes the connection mode. For Agent Mode, the client uses the Session object SetAgent() method. For Monitor Mode, the client uses the Session object SetMessageFilter() method.

The CTI OS server responds to the client's SetAgent or SetMessageFilter request by sending an exact copy of the current agent and call states, using the following events:

- **eQueryAgentStateConf**. This event contains the current state of the agent in all the skill groups it belongs to.
- **eSnapshotDeviceConf**. This event contains a list of all the calls the agent is or was working on.

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

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

CTI OS Client Interface Library Architecture

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

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

This chapter describes each layer of the CIL architecture.

Object Interface Framework

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

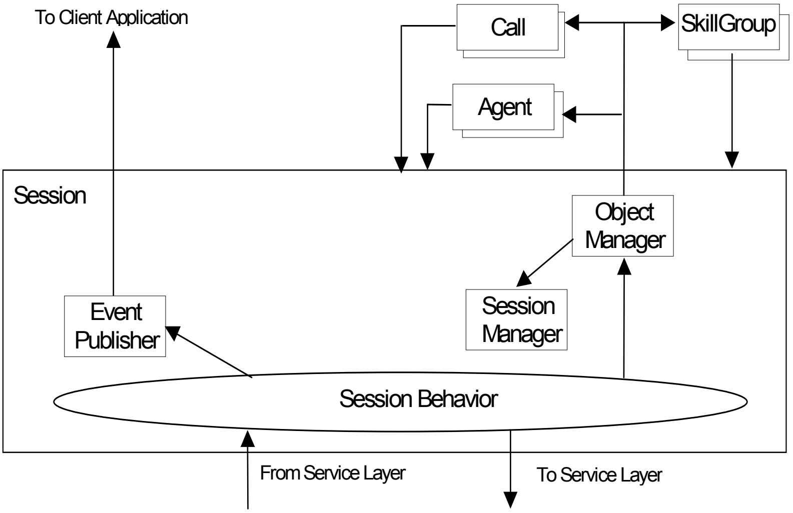
Session Object

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

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

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

Figure 2-1 Object Interface Framework Block Diagram



Session Manager

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

Session Behavior

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

Connection

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

Connection Failure and Recovery

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

Session Modes

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

Agent Mode

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

Monitor Mode

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

Object Manager and Event Passing

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

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

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

Creating Objects

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

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

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

Call Object Lifetime

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

Agent Object Lifetime

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

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

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

SkillGroup Object Lifetime

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

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

Object Factory

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

Event Publisher

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

Exposed Interfaces

The following event interfaces are exposed to the client.

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

Adding and Removing Subscribers

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

Event Distribution

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

Service Layer

The service layer sits between the OIF and the connection layer. Its main purpose is to translate between the high-level command/event message structure of the upper CIL and the low-level network message structure of the connection layer. A secondary purpose of the service layer is to isolate the client from the network such that network issues do not block the client and vice versa.

Connection Layer

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

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

Multithreaded CTI OS Client Application

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

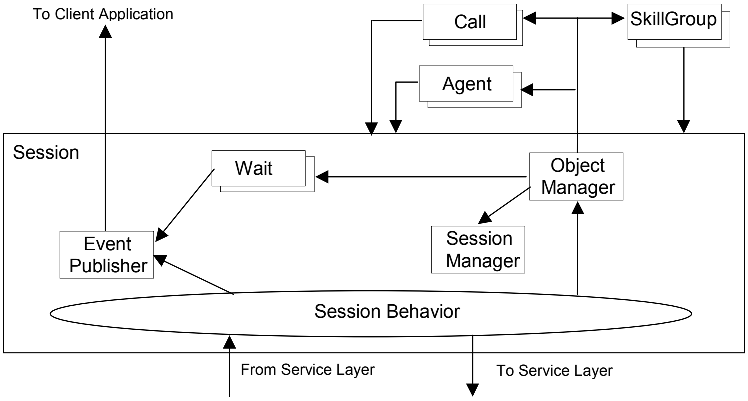
To write a multithreaded client, the application programmer will need to adhere to the programming model described in this document.

Support for Multithreaded Client in CIL

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

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

Figure 2-2 Multithreaded Application Support



Wait Object

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

Properties

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

Table 2-1 Wait Object Properties

Property	Description
ThreadId	Double word type variable to store the system assigned thread ID.
EventMask	Integer type variable to store the event mask for which the thread will be waiting on
WaitObject	Handle type variable to store the handle to the system's wait object.

Methods

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

Table 2-2 Wait Object Methods

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

Usage at the Client

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

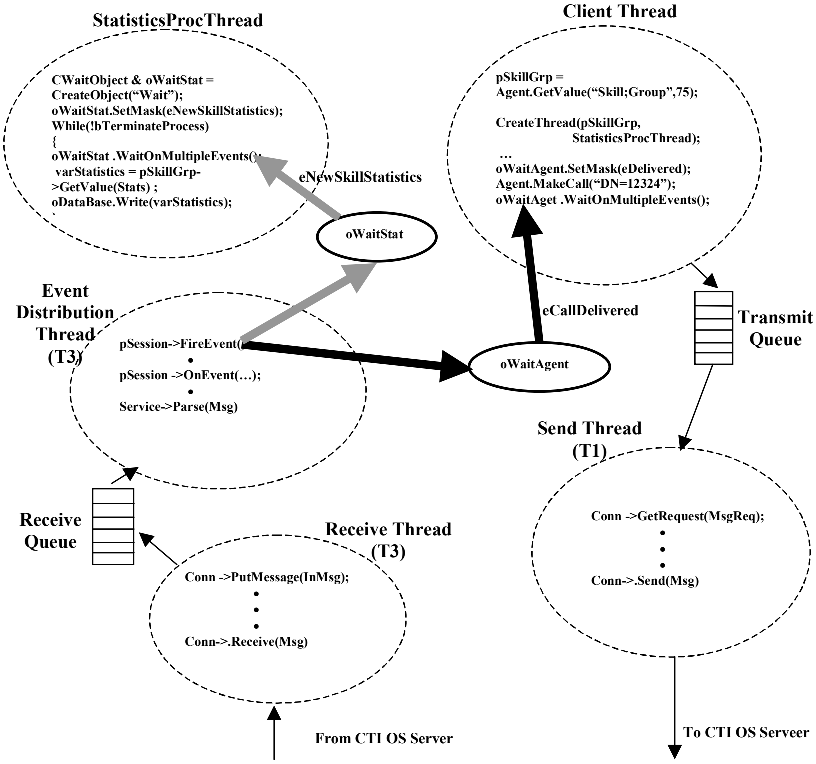
In order to create an instance, the application must call CreateWaitObject at the beginning of the thread, then call SetEventMask to specify the events to wait for. To make the thread stop for events, then call WaitOnMultipleEvents via the Wait Object. Before the thread is terminated, to destroy the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

```
{
   CAgent * pAgent = (CAgent *) lpParameter;
   Arguments & argsWaitParams = Arguments::CreateInstance();

   CWaitObject * pWaitCall = pSession->
CreateWaitObject(argsWaitParams);

   oWaitCall.SetMask(eCallDeliveredMask |
                     eCallEstablished |
                     eCallControlFailureMask );

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

   pWaitCall->WaitOnMultipleEvents();

   pSession->DestroyOject(pWaitCall);
argsWaitParams.Release().

   return TERMINATED_OK;
}


Other example:


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

   int nEventMask = eCallDeliveredMask |
                    eCallEstablishedMask |
eCallControlFailureconfMask;

   argsWaitParams.AddItem(EVENTMASK,nEventMask);
   CWaitObject * pWaitCall = pSession->
CreateWaitObject(argsWaitParams);

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

   pWaitCall->WaitOnMultipleEvents();

   pSession->DestroyOject(pWaitCall);
argsWaitParams.Release().

   return TERMINATED_OK;
}
```

Multithreaded Application Example

Figure 2-3 shows a client application with two threads. The main client's thread deals with agent activity and call management. The second, updates a corporate database used for account billing.

Figure 2-3 Multithreaded Application Example



```
CWaitObject & oWaitStat =
CreateObject(“Wait”);
oWaitStat.SetMask(eNewSkillStatistics);
While(!bTerminateProcess)
{
oWaitStat .WaitOnMultipleEvents();
 varStatistics = pSkillGrp-
>GetValue(Stats) ;
oDataBase.Write(varStatistics);
```

```
pSkillGrp =
Agent.GetValue(“Skill;Group”,75);

CreateThread(pSkillGrp,
                StatisticsProcThread);
 …
oWaitAgent.SetMask(eDelivered);
Agent.MakeCall(“DN=12324”);
oWaitAget .WaitOnMultipleEvents();
```

```
pSession->FireEvent()
        •
pSession ->OnEvent(…);
        •
Service->Parse(Msg)
```

```
Conn ->GetRequest(MsgReq);
        •
        •
        •
Conn->.Send(Msg)
```

```
Conn ->PutMessage(InMsg);
        •
        •
        •
Conn->.Receive(Msg)
```

This example shows clearly how an application can take advantage of workload distribution by using different threads to do the work. CIL’s programming model is easy to use and does not compromise extensibility. Moreover, the architecture guaranties that all CTI OS objects remain consistent during the lifetime of the application. CIL will only signal Wait Objects after the event owners (CTI OS Objects) processed the event(s).

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

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

```
CWaitObject * pWaitCall = pSession->CreatWaitObject(argsWaitParams");
```

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

```
PWaitCall->SetMask(eCallDeliveredMask);
```

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

```
pAgent->MakeCall ("DN=1234").
```

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

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

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

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

```
pSession->DestroyObject(oWaitCall);
argsWaitParams.Release()
```

Subclassing

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

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

Step 1 Create a class that derives from any of the standard OIF classes, as shown in the following example.

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

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

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

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

```
#include "CIL.h"

//Implement Class factory
IMPLEMENT_CIL_OBJECT_FACTORY(CSupportCall)

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

//////////////////////////////////////////////////////////////////////
//   Process Events Received from CTI OS Server
//////////////////////////////////////////////////////////////////////
void CSupportCall::OnEvent(int iEventID, Arguments & rEventParam)
{
     switch(iEventID)
    {
        case eCallBeginEvent:
               OnCallBegin(rEventParam);
        break;
        case eCallEndEvent:
               OnCallEnd(rEventParam);
        break;

        //other events
    }
    //Let default processing at the CIL to happen
    CCall::OnEvent(rEventParam);
}

//////////////////////////////////////////////////////////////////////
void CSupportCall::OnCallBegin(Arguments & rEventParam)
{
   CILRefArg & rCILRefArg = (CILRefArg &) GetValue(CURRENTAGENT);
   CAgent * pCurrAgent = (CAgent *) rCILRefArg.GetValue();

   m_objAuditInfo.TrackServiceRep(*pCurrAgent );
   m_objAuditInfo.TrackStartTime(GetTime());
   m_objAuditInfo.TrackPriority(GetPriority());

   rCILRefArg .Release();
   pCurrAgent->Release();
}
```

Step 3 Register the class factory for your new subclass with the instance of the Session object that will use this subclass instead of the default. It is important to remember that only the Session where the factory was registered knows about it. If the application uses more than one instance of a Session object it is required to register the new class on all the other Sessions so they can use it. It is recommended that all factories registrations be performed before any connection is open. This guarantees that CIL will use the new classes to create new object instances.

```
#include  <iostream.h>
#include ”CIL.h”
#include “SupportCall.h”

void main(void)
{
      CCtiOsSession  ctiSession;

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

     //Continue with program

}
```

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

Handling Events

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

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

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

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

Finally, the application subscribes the new interface objects with CIL such that messages can be forwarded.

Handling Events in C++

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

Creating a Subscriber Object Class

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

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

Table 3-1 CIL Adapter Classes

Adapter Class	Description
AllInOneEventsAdapter	Provide the default implementation for the message handlers in IAllEvents
AgentEventsAdapter	Provide the default implementation for the message handlers in IAgentEvents
ButtonEnablementEventsAdapter	Provide the default implementation for the message handlers in IButtonEnablementevents
CallEventsAdapter	Provide the default implementation for the message handlers in ICallEvents
SkillGroupEventsAdapter	Provide the default implementation for the message handlers in ISkillGroupEvents
SessionEventsAdapter	Provide the default implementation for the message handlers in ISessionEvents

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

```
#include "CIL.h"

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

            //We are interested only on these events
virtual void OnCallRequestFailed(Arguments & rArguments) ;
virtual void OnCallQueuedEvent(Arguments & rArguments);
virtual void OnCallDequeuedEvent(Arguments & rArguments);
virtual void OnCallReachedNetworkEvent(Arguments & rArguments);
};
//Notify the user and undo database transaction
void MyCallEventSink ::OnCallRequestFailed(Arguments & rArguments)
{
    pWallBoard->NotifyUI(rArguments);
    pDataBase->RollBack(m_CurTransaction);
}

//Registers arrival to queue and starts tracking of mean wait until
service
//For this call
void MyCallEventSink ::OnCallQueuedEvent(Arguments & rArguments)
{
        pAuditTrail->StartTrackingMeanWait(rArguments);
}
//Registers departure from queue and stops tracking of mean wait until
service
//For this call
void MyCallEventSink ::OnCallDequeuedEvent(Arguments & rArguments)
{
     pAuditTrail->StopTrackingMeanWait(rArguments);
}
//Keeps track of the calls that end up leaving the call center
 void MyCallEventSink ::OnCallReachedNetworkEvent(Arguments &
rArguments)
{
    pAuditTrail->RegisterDeparture(rArguments);
}
```

Registering a C++ Subscriber Object

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

```
int       AddAllInOneEventListener(IAllInOne * pAllInOneEvents);
 int       AddAgentEventListener(IAgentEvents * pAgentEvents);
 int       AddButtonEnablementEventListener(

IButtonEnablementEvents * pButtonEvents);
 int       AddCallEventListener (ICallEvents * pCallEvents);
 int       AddSkillGroupEventListener (ISkillGroupEvents *
pSkillGroupEvents);
 intAddSessionEventListener (ISessionEvents * pSessionEvents);
```

Input Parameters

pAllInOneEvents

Points to an IAllInOne subscriber object.

pAgentEvents

Points to an IAgentEvents subscriber object.

pButtonEvents

Points to an IButtonEnablementEvents subscriber object.

pCallEvents

Points to an ICallEvents subscriber object.

pSkillGroupEvents

Points to an ISkillGroupEvents subscriber object.

pSessionEvents

Points to an ISessionEvents subscriber object.

Return Value

If successful it returns CIL_OK, otherwise it returns E_CTIOS_INVALID_ARGUMENT.

Example

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

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

     //More things to initialize

}
```

Unregistering Subscriber

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

```
int       RemoveAllInOneEventListener(IAllInOne * pAllInOneEvents);
 int       RemoveAgentEventListener(IAgentEvents * pAgentEvents);
 int       RemoveButtonEnablementEventListener(

IButtonEnablementEvents * pButtonEvents);
 int       RemoveCallEventListener (ICallEvents * pCallEvents);
 int       RemoveSkillGroupEventListener (
 ISkillGroupEvents * pSkillGroupEvents);
 intRemoveSessionEventListener (ISessionEvents * pSessionEvents);
```

Input Parameters

pAllInOneEvents

Points to an IAllInOne subscriber object to be removed.

pAgentEvents

Points to an IAgentEvents subscriber object to be removed

pButtonEvents

Points to an IButtonEnablementEvents subscriber object to be removed pCallEvents

Points to an ICallEvents subscriber object to be removed

pSkillGroupEvents

Points to an ISkillGroupEvents subscriber object to be removed

pSessionEvents

Points to an ISessionEvents subscriber object to be removed

Return Value

If successful it returns CIL_OK, otherwise it returns E_CTIOS_INVALID_ARGUMENT.

Example

```
CMyCtiApplication::RevokeSubscriptionToEvents(void)
{

     //UnRegister Subscriber
     m_ctiSession.RemoveCallEventListener ((ICallEvents *)
m_pCallSubscriber);

     //More things to initialize

}
```

Handling Events in COM

COM CIL client applications can be classified in two different groups:

- Automation based Applications.
- COM (ATL, MFC, COM SDK) based Applications.

Automation Based Applications

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

```
' VB sample for a simple CTIOS phone
' needs references to CTIOSCLIENTLib CTIOSSESSIONRESOLVERLib
‘and CTIOSARGUMENTSLib
'
' dim CTIOS session interface
' the session interface handles connect, setagent and others

Dim WithEvents m_Session As CTIOSCLIENTLib.Session

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

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

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

End Sub
```

ATL, MFC and COM SDK Based Applications

The clients in this group are all those applications built using ATL, MFC, or Common object Model SDK. These applications are usually written in C++. Client applications are usually created this way to take advantage of the performance improvements that are obtained by using directly the v-tables and IUnknown objects. Applications of this type tend to be more complex but faster.

In this model client applications subscribe for events with a COM server object by registering an event sink instance in the client with the connection point at the event interface from which events are expected. This is the standard protocol defined in MS COM SDK. If a COM server provides more than one event interface a client is allowed to connect to any of them.

The COM Session object publishes the following interfaces:

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

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

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

```
//Call Event Sink
class ATL_NO_VTABLE CSinkCallEvents :
public CComObjectRootEx<CComSingleThreadModel>,
public CComCoClass<CSinkCallEvents, &__uuidof(_ICallEvents)>,
public IDispatchImpl<_ICallEvents,

&__uuidof(_ICallEvents),
&LIBID_CTIOSCLIENTLib>,
public IDispEventImpl<1,
                                                  CSinkCallEvents,

&__uuidof(_ICallEvents),

&LIBID_CTIOSCLIENTLib, 1, 0>
{
   //Class definition
};

//Agent Event Sink
class ATL_NO_VTABLE CSinkAgentEvents :
public CComObjectRootEx<CComSingleThreadModel>,
public CComCoClass<CSinkAgentEvents, &__uuidof(_IAgentEvents)>,
public IDispatchImpl<_IAgentEvents,

&__uuidof(_IAgentEvents),
&LIBID_CTIOSCLIENTLib>,
public IDispEventImpl<1,
                                                   CSinkAgentEvents,

&__uuidof(_IAgentEvents),

&LIBID_CTIOSCLIENTLib, 1, 0>
{
   //Class definition
};
```

Registering Event Sink

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

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

Unregistering Event Sink

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

   // Stop event notification for IAgentEvents
   hr = m_pSinkAgentEvents->DispEventUnadvise( m_pSession );
   _ASSERTE(SUCCEEDED(hr));
```

```
        m_pSinkAgentEvents->Release();
        m_pSinkAgentEvents = NULL;
    }
```

Session Object

The session object establishes and maintains a connection to an active CTI OS server. The object provides two main functions: automatically recovery from connection failures and filtering and distributing events. The session object acts as a class factory for agent call and skill group objects. The session object receives all events from a CTI OS connection. It interprets the event, determines whether it applies to an agent, an existing call, or whether it must create a new call object.

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

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

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

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

Working With CTI OS Objects

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

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

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

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

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

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

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

```
Dim argsCalls As Arguments
    Dim strUOID As String
    ‘Gets all calls  from Session
    Set argsCalls = m_session.GetValue("Calls")

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

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

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

```
Private Sub m_session_OnCallDelivered(ByVal argEventParams As Object)

   Dim ctiCall As CTIOSCLIENTLib. Call
   Dim strUOID As String

    strUOID  =  argEventParams.GetValue(“UniqueObjectID”)

   ‘ Compose string "UniqueObjectID=XXXXXX" to retrieve a Call
referenece
    strUOID =  m_session.GetValue("Uniqueobjectid")

    Dim strObjToRetrieve As String
    strObjToRetrieve = "UniqueObjectID=" & strUOID
    ' Retrive Call Reference and use it
    Set ctiCall = m_session.GetValue(strObjToRetrieve)
   ‘Pick up the call
    CtiCall.Aswer

   Set ctiCall = Nothing

End Sub
```

The application developer should note that the type of reference returned by GetValue() and GetElement() variates from environment to environment as follows:

Table 4-1 Returned References

Environment	Reference Type	Description
C++	CILRefArg	Any method returning a reference to a CTI OS object will return a reference to the helper class named CILRefArg (See Chapter 8, "Helper Classes") that contains the actual reference to a CAgent, CCall or CSkillGroup object instance.
COM	Idispatch	Any method returning a reference to a CTI OS COM object will return a reference to IAgent, ICall and ISkillGroup interfaces.

Methods

Table 4-2 lists the available session object methods.

Table 4-2 Session Object Methods

Method	Description
Connect	Establishes a connection to a CTI OS server.
Disconnect	Closes the connection to the CTI OS server.
GetPropertyAttribute	Retrieves attribute information for a specified session property.
GetValue (also GetValueInt, GetValueString)	Retrieves the value of a specified session property.
isAgentMode	Checks the current mode and returns true if agent mode.
isSupervisorMode	Checks the current mode and returns true if supervisor mode.

Table 4-2 Session Object Methods

MakeRequest	Sends a message request to the CTI OS Server
OnEvent	Specifies an action to take on occurrence of a specified event.
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
SetMessageFilter	Sets a message filter that the client application must pass in order to connect in Monitor Mode,

Connect

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

Syntax

C++

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

COM

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

Input Parameters

rArguments

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

Table 4-3 Connect Parameters

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

pVariantArgs

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

Return Values

C++

Return Code	When Returned
CIL_OK	If Successful
E_CTIOS_IN_FAILOVER	If the Session object has started the connection recovery algorithm
E_CTIOS_INVALID_PROPERTY	When a parameter passed in the connection parameters is invalid

COM

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

Examples

C++

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

nRetVal = ctiSession.Connect(rArgSessionConn);

rArgSessionConn.Release();

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

COM C++

```
    HRESULT hr = S_OK;
    IArgumentsPtr      arConnectArgs;
    VARIANT vParam;
    VariantInit(&vParam);

    hr = arConnectArgs.CreateInstance(

OLESTR("CtiOsComArguments.ComArguments"));

    vParam.vt = VT_DISPATCH;
    hr = arConnectArgs->QueryInterface(IID_IDispatch ,

(void **) &vParam.pdispVal);
    m_pSession->Connect(&vParam);
```

```
   arConnectArgs = NULL;
```

VB

```
    ' Prepares the connect request
    Dim m_Args As New Arguments

    m_Args.AddItem "CtiOsA", CStr(m_SideA)
    m_Args.AddItem "PortA", m_PortA
    m_Args.AddItem "CtiOsB", CStr(m_SideB)
    m_Args.AddItem "PortB", m_PortB
    m_Args.AddItem "Heartbeat", m_Heartbeat

    Dim vRequestParam As Variant
    Set vRequestParam = m_Args

    m_Session.Connect vRequestParam

   Set m_Args = Nothing
```

Disconnect

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

Syntax

C++

```
void Disconnect ();
```

COM

```
HRESULT Disconnect ( );
```

Parameters

None.

Return Values

C++

None.

COM

Always returns S_OK .

Examples

C++

```
ctiSession.Disconnect();
```

COM C++

```
m_pSession-> Disconnect();
```

VB

```
m_Session. Disconnect
```

GetPropertyAttribute

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

Table 4-4 Session Properties

Property	Type	Description
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 able be blank if it has temporarily lost the current connection even if it is trying to reconnect. Otherwise, the name of the server should be the name of CTI OS server A or B.
CurrentPort	integer	Tcp/ip address of the current connected CTI OS server. May be port A or B.
ConnectedSince	integer	Time of day in milliseconds when connected.
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.
ConnectionMode	integer value	eAgentConnection, eMonitorConnection, or eNotConnected.
LastError	integer value	Last error code, if any. Otherwise this value is 0.
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.
TryingPort	integer	Tcp/ip address of the server where a connection is being attempted. May be port A or B.

Table 4-4 Session Properties (continued)

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

Table 4-4 Session Properties (continued)

Property	Type	Description
SkillGroup Reference Array	call skill group reference	Reference to SkillGroup Array.
EmailReference Array	call object reference	Reference to Call Array.

Syntax

```
GetPropertyAttribute( propertyname, attribute requested )
```

GetValue

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

Syntax

```
GetValue( key )
```

isAgentMode

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

Syntax

C++

```
   bool              isAgentMode ();
```

COM

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

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

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

COM C++

```
VARIANT_BOOL  bRet = VARIANT_FALSE;

m_pSession->isAgentMode(&bRet);

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

VB

```
bRet  = m_Session.isAgentMode

            if bRet = True Then
                 fnAutoLogin
            End If
```

isSupervisorMode

The isSupervisorMode method checks if the current connection mode is supervisor mode.

Syntax

C++

```
   bool             isSupervisorMode ();
```

COM

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

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

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

COM C++

```
VARIANT_BOOL  bRet = VARIANT_FALSE;

m_pSession->isSupervisorMode(&bRet);

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

VB

```
bRet  = m_Session.isSupervisorMode

            if bRet = True Then
                 fnAutoLogin
            End If
```

MakeRequest

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

Syntax

C++

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

COM

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

Input Parameters

nRequestId

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

rReqParam

Reference to an arguments array that contains the request parameters.

pReqParams

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

Return Values

C++

None.

COM

Always returns S_OK.

Examples

C++

```
       Args. AddItem(“DialNumber”, “1234”);
       Args. AddItem(“UniqueObjectID”,

ctiAgent.GetValueString(“UniqueObjectID”));

      m_pSession->MakeRequest(eMakeCallRequest, Args );

      Args.Release();
```

COM C++

```
    HRESULT hr = S_OK;
    IArgumentsPtr      Args;
    VARIANT vParam;
    VariantInit(&vParam);

    hr = Args.CreateInstance(

OLESTR("CtiOsComArguments.ComArguments"));

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

ctiAgent->GetValueString("UniqueObjectID"));

    vParam.vt = VT_DISPATCH;
    hr = Args ->QueryInterface(IID_IDispatch ,
                                                 (void **)
&vParam.pdispVal);

      m_pSession->MakeRequest(eMakeCallRequest, & vParam );

      Args = NULL;
```

VB

```
' Prepares the connect request
    Dim m_Args As New Arguments

     m_Args. AddItem "DialNumber",  "1234"
     m_Args. AddItem "UniqueObjectID",  _

ctiAgent.GetValueString("UniqueObjectID")

    Dim vRequestParam As Variant
    Set vRequestParam = m_Args

    m_pSession.MakeRequest eMakeCallRequest, Args
```

OnEvent (C++ Only)

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

Syntax

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

Input Parameters

iEventID

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

rEventParam

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

Return Values

None.

RequestDesktopSettings

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

Syntax

```
RequestDesktopSettings( desktop_type )
```

SetAgent

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

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

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

Syntax

C++

```
int SetAgent(CAgent & ctiAgent);
```

COM

```
HRESULT  SetAgent ( [in]  LPDISPATCH pDispParam  );
```

Input Parameters

ctiAgent

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

pDispParam

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

Return Values

C++

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

COM

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

Examples

C++

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

            nRetVal = ctiSession.SetAgent(ctiAgent);

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

COM C++

```
IAgentPtr m_pAgent;

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

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

        m_pSession->SetAgent(m_pAgent);
```

VB

```
Dim    ctiAgent As New CTIOSClientLib.Agent
     ctiAgent.SetValue “AgentID”,  "23840"
     ctiAgent.SetValue “Password” , "23840"
     ctiAgent.SetValue “PeeripheralID”, 5000
     ctiAgent.SetValue “Instrument”, "23801"

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

SetCurrentCall

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

Syntax

C++

```
 void SetCurrentCall(CCall * pCall);
```

COM

```
HRESULT    SetCurrentCall ([in] LPDISPATCH pCallParam);
```

Input Parameters

pCall

Pointer to the CCall to make current.

pCallParam

Pointer to an ICall to make current .

Return Values

C++

None.

COM

Always returns S_OK

Examples

C++

```
void CMyListBox::OnSelChange(…)
{
     int nCurSel =  this->GetCurSel();

     CCall * pCall = GetItemDataPtr(nCurSel );

     m_pSesion->SetCurrentCall(pCall);
}
```

COM C++

```
void CMyActiveListBox::OnSelChange(…)
            {
                 int nCurSel = this->GetCurSel();

                 ICall * pCall = GetItemDataPtr(nCurSel );

                 m_pSesion->SetCurrentCall((LPDISPATCH)pCall);
            }
```

VB

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

   ' Get UniqueObjectID to get the actual call and answer right away

    Dim m_uid As String
    m_uid = pDispParam.GetValue("UniqueObjectID")
    Dim tmp As String

    tmp = "UniqueObjectID=" + m_uid

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

    m_session.SetCurrentCall(m_call)

   m_call.Answer();

End Sub
```

SetMessageFilter

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

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

Operator

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

Table 4-5 CTI OS Logical Operators

Operator	Description
AND	Logical AND
;	Logical AND
OR	Logical OR
NOT	Logical NOT

Expression

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

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

Keyword

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

Table 4-6 Common CTI OS Keywords and Events

Keywords	Events
AgentID, Instrument, AgentExtension, AgentState	OnAgentStateChange, OnNewAgentStatistics, OnQueryAgentStateConf
SkillGroupNumber, SkillGoupPriority	OnNewSkillGroupStatistics
CallID, CallState, CallVarable1, …, CallVariable10, **user.***NamedVariable*, **user.***NamedArray[Offset]*, UserToUser, WrapUp	OnCallPreRoute, OnCallBegin, OnCallDelivered, OnCallEstablished, OnCallHeld, OnCallReceived, OnCallTransferInit, OnCallTransferComplete, OnCallCleared, OnCallEnd

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

Value

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

```
222
```

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

```
222,333,444
```

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

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

It may be a combination of the preceding:

```
222,333,44*
```

Syntax

C++

```
int  SetMessageFilter(string strFilter)
```

COM

```
HRESULT    SetMessageFilter ([in] BSTR bstrFilter);
```

Input Parameters

strFilter

String text containing the message filter.

bstrFilter

String text containing the message filter.

Return Values

C++

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

COM C++

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

Examples

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

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

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

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

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

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

C++

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

COM C++

```
        BSTR bstrFilter =

SysAllocString(OLESTR("SkillGroupNumber=23,24;OnCallBegin"));

        m_pSession->SetMessageFilter(bstrFilter);
        SysFreeString(bstrFilter);
```

VB

```
m_Session. SetMessageFilter  "SkillGroupNumber=23,24;OnCallBegin"
```

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.

Field	Description
TimeOfDay	Integer value with time of day expressed in milliseconds.
CurrentServer	Name or tcp/ip address of the current connected CTI OS server.

OnConnectionFailure

The OnConnectionFailure event is generated when an already established. It returns the name of the failed connected server and the failure time of day. Retry is automatic and can be followed by an OnConnection when connection has been successfully reestablished. The client application need not take any special action but may use it to display connection status.

Field	Description
TimeOfDay	Integer value with time of day expressed in milliseconds.
FailedServer	Name or tcp/ip address of the server that has failed to respond. See reason code.
ReasonCode	CTIOS_SERVER_CONNECTIONBROKEN, CTIOS_SERVER_MISSINGHEARTBEATS

OnHeartbeat

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

Field	Description
TimeOfDay	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) the connection to a new CTI OS server is automatically restarted.

Field	Description
TimeOfDay	Integer value with time of day expressed in milliseconds.
ConsecutiveMissedHeartbeats	Integer value with the number of heartbeats missed.
HeartbeatInterval	Integer value with the heartbeat interval, in milliseconds.

OnMonitorModeEstablished

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

Field	Description
TimeOfDay	Integer value with time of day expressed in milliseconds.
Mode	CTIOS_SERVER_MONITORMODE or CTIOS_SERVER_MONITORMODE_EXIT

OnConnectionClosed

The OnConnectionClosed message is generated when a connection is terminated.

Field	Description
MessageID	Integer, Message ID

OnConnectionRejected

The OnConnectionRejected message is generated when a connection is rejected.

Field	Description
EventTime	Integer, Time of day
ReasonCode	Integer, Reason code
MessageID	Integer, Message ID

OnSetAgentModeEvent

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

OnCurrentCallChanged

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

OnCurrentAgentReset

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

CHAPTER 5

Agent Object

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

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

Methods

Table 5-1 lists the supported Agent object methods.

Table 5-1 Agent Object Methods

Method	Description
AgentTeamList	Retrieves the current agent team list.
BadCallLine	Informs the CTI OS Server of a bad line.
DisableAgentStatistics	Disables agent statistic messages.
DisableSkillGroupStatistics	Disables skill group statistic messages.

Table 5-1 Agent Object Methods (continued)

Method	Description
Emergency	Lets an agent makes an emergency call to the supervisor.
EnableAgentStatistics	Enables agent statistic messages.
EnableSkillGroupStatistics	Enables skill group statistic messages.
GetAgentState	Returns the current agent state.
GetElement	Retrieves a property from the Agent object based on the property's index value.
GetMonitoredAgent	Returns the agent object that is currently being monitored.
GetMonitoredCall	Returns the call object that is currently being monitored.
GetPropertyAttribute	Retrieves attribute information for a specified agent property.
GetSkillGroups	Returns an array of SkillGroups objects
GetValue (also contains GetValueInt, GetValueString)	Retrieves a property from the Agent object based on the property's name key.
Login	Logs an agent into the ACD.
Logout	Logs an agent out of the ACD.
MakeCall	Initiates a call to a device or agent
MonitorAgentTeam	Enables monitoring of a specified agent team.
MonitorAgentTeamAll	Enables monitoring of all agent teams.
OnEvent	Specifies an action to take when an agent object receives agent state, agent statistic and associated skill group events.
SendChatMessage, SendUserMessage	Send asynchronous messages between CTI clients
SetAgentState	Requests a new agent state.
SetMonitoredAgent	Sets an agent object to be monitored.
SetMonitoredCall	Sets a call object to be monitored.

Table 5-1 Agent Object Methods (continued)

Method	Description
SetValue	Sets a value for a property name key.
SuperviseCall	Enables monitoring a call of an agent on your team.
SupervisorAssist	Lets an agent makes a request for assistance call to the supervisor.

AgentTeamList

The AgentTeamList method retrieves the current agent team list.

Syntax

C++

```
int RequestAgentTeamList ( );
```

COM (standard COM API)

```
HRESULT AgentTeamList ( );
```

Parameters

None.

Return Values

C++

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

COM

If the method was able to get AgentTeamList, it returns S_OK. Otherwise,it returns E_FAIL.

Examples

C++

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

// Get a valid agent pointer
   if (pAgent)
  {
      nRet  = pAgent->RequestAgentTeamList();

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

COM C++

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

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

return hr;
```

BadCallLine

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

Syntax

C++

```
int BadCallLineRequest ( );
```

COM

```
HRESULT BadCallLine ( );
```

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

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

// Get a valid agent pointer
   if (pAgent)
  {
      nRet  = pAgent-> BadCallLineRequest ();

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

\COM C++

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

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

return hr;
```

Emergency

The Emergency method lets an agent makes an emergency call to the supervisor.

Syntax

C++

```
int EmergencyCallRequest ( );
```

COM

```
HRESULT Emergency ( );
```

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

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

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

COM C++

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

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

return hr;
```

EnableAgentStatistics/DisableAgentStatistics

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

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

Syntax

C++

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

COM

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

Parameters

pVariantArgs

Pointer to variant that wraps an empty argument

Return Values

C++

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

COM

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

Examples

C++

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

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

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

COM C++

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

return hr;
```

GetAgentState

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

Syntax

C++

```
virtual enumCTIOS_AgentState GetAgentState();
```

COM (standard COM API)

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

Output Parameters

pAgentStateCode

Pointer that returns agent state

Return Values

C++

Returns the agent state.

COM

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

Examples

C++

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

// Get a valid agent pointer
   if (pAgent)
  {
AgentState = pAgent-> GetAgentState ();

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

COM C++

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

return hr;
```

VB

```
Dim myState As ctios.enumCTIOSAgentState

MyState = agent.GetAgentState
```

GetElement

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

GetMonitoredAgent

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

Syntax

C++

```
virtual CAgent * GetMonitoredAgent();
```

COM

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

Output Parameters

pIAgent

The returned agent object

Return Values

C++

An agent object.

COM

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

Examples

C++

```
 CAgent * pAgent = NULL;

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

                            // Check if the pMonitoredAgent is a good
pointer
}
```

COM C++

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

return hr;
```

GetMonitoredCall

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

Syntax

C++

```
virtual CCall * GetMonitoredCall();
```

COM (standard COM API)

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

Output Parameters

pIAgent

The returned call object

Return Values

C++

Returns a call object.

COM

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

Examples

C++

```
 CAgent * pAgent = NULL;

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

                            // Check if the pMonitoredCall is a good
pointer
}
```

COM C++

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

return hr;
```

GetPropertyAttribute

Syntax

GetPropertyAttribute(propertyname, attribute requested)

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

Table 5-2 Agent Properties

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

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

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

‘ Then get the desired statistics
Dim availTimeSession As Integer
Dim loggedOnTimeSession As Integer
availTimeSession = args.GetValue("AvailTimeSession")
bargeInCallsToday = args.GetValue("BargeInCallsToday")
```

Note Not all the statistics values listed below are present in every system configuration. Whether or not a particular statistic value is available depends both on the protocol version of CTIServer with which CTIOS connects and on the peripheral on which the agent resides.

Table 5-3 Agent Statistics

Statistic	Meaning
AvailTimeSession	Total time, in seconds, the agent was in the Available state for any skill group.
LoggedOnTimeSession	Total time, in seconds, the agent has been logged on.
NotReadyTimeSession	Total time, in seconds, the agent was in the Not Ready state for all skill groups.
AgentOutCallsSession	Total number of completed outbound ACD calls made by agent.
AgentOutCallsTalkTimeSession	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.
AgentOutCallsTimeSession	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.
AgentOutCallsHeldSession	The total number of completed outbound ACD calls the agent has placed on hold at least once.
AgentOutCallsHeldTimeSession	Total number of seconds outbound ACD calls were placed on hold.

Table 5-3 Agent Statistics (continued)

Statistic	Meaning
HandledCallsSession	The number of inbound ACD calls handled by the agent.
HandledCallsTalkTimeSession	Total talk time in seconds for Inbound ACD calls counted as handled by the agent. Includes hold time associated with the call.
HandledCallsAfterCall TimeSession	Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.
HandledCallsTimeSession	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.
IncomingCallsHeldSession	The total number of completed inbound ACD calls the agent placed on hold at least once.
IncomingCallsHeldTimeSession	Total number of seconds completed inbound ACD calls were placed on hold.
InternalCallsSession	Number of internal calls initiated by the agent.
InternalCallsTimeSession	Number of seconds spent on internal calls initiated by the agent.
InternalCallsRcvdSession	Number of internal calls received by the agent.
InternalCallsRcvdTimeSession	Number of seconds spent on internal calls received by the agent.
InternalCallsHeldSession	The total number of internal calls the agent placed on hold at least once.
InternalCallsHeldTimeSession	Total number of seconds completed internal calls were placed on hold.

Table 5-3 Agent Statistics (continued)

Statistic	Meaning
AvailTimeToday	Total time, in seconds, the agent was in the Available state for any skill group.
LoggedOnTimeToday	Total time, in seconds, the agent has been logged on.
NotReadyTimeToday	Total time, in seconds, the agent was in the Not Ready state for all skill groups.
AgentOutCallsToday	Total number of completed outbound ACD calls made by agent.
AgentOutCallsTalkTimeToday	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.
AgentOutCallsTimeToday	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.
AgentOutCallsHeldToday	The total number of completed outbound ACD calls the agent has placed on hold at least once.
AgentOutCallsHeldTimeToday	Total number of seconds outbound ACD calls were placed on hold.
HandledCallsToday	The number of inbound ACD calls handled by the agent.
HandledCallsTalkTimeToday	Total talk time in seconds for Inbound ACD calls counted as handled by the agent. Includes hold time associated with the call.

Table 5-3 Agent Statistics (continued)

Statistic	Meaning
HandledCallsAfterCallTimeToday	Total after call work time in seconds for Inbound ACD calls counted as handled by the agent.
HandledCallsTimeToday	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.
IncomingCallsHeldToday	The total number of completed inbound ACD calls the agent placed on hold at least once.
IncomingCallsHeldTimeToday	Total number of seconds completed inbound ACD calls were placed on hold.
InternalCallsToday	Number of internal calls initiated by the agent.
InternalCallsTimeToday	Number of seconds spent on internal calls initiated by the agent.
InternalCallsRcvdToday	Number of internal calls received by the agent.
InternalCallsRcvdTimeToday	Number of seconds spent on internal calls received by the agent.
InternalCallsHeldToday	The total number of internal calls the agent placed on hold at least once.
InternalCallsHeldTimeToday	Total number of seconds completed internal calls were placed on hold.

GetSkillGroups

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

Syntax

```
GetSkillGroups( optionalSkillGroup )
```

Example

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

GetValue

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

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

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

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

' iterate over all statistics
Dim agent As CTIOSCLIENTLIB.Agent
Dim allStatistics As Arguments

allStatistics = agent.GetValue( “Statistics” )
For Each stat In allStatistics
...

Next
```

Syntax

```
GetValue (key)
```

Login

The Login method performs a login to the ACD (if supported). Generally, the minimum parameters required to log into an ACD are AgentID and Instrument. Often, based on customer configuration, the minimum requirement includes an ACD password. Optional arguments include Extension or PositionID and Password.

Syntax

C++

```
virtual int Login(Arguments & rArguments);
```

COM (standard COM API)

```
HRESULT Login  ( [in]  VARIANT * pVariantArgs  );
```

Input Parameters

rArguments

Arguments array that contains login parameters.

pVariantArgs

Arguments array that contains login parameters.

Return Values

C++

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

COM

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

Examples

C++

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

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

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

COM C++

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

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

VB

```
Dim argsLogin Param As New Arguments

Dim WithEvents session As ctios. Session

Dim WithEvents agent As ctios. Agent



session.SetAgent( agent )

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

Logout

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

Syntax

C++

```
virtual int Logout(Arguments & rArguments);
```

COM (standard COM API)

```
HRESULT Logout ( [in]  VARIANT * pVariantArgs  );
```

Input Parameters

rArguments

Arguments array that contains login parameters.

pVariantArgs

Arguments array that contains login parameters.

Return Values

C++

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

COM

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

Examples

C++

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

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

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

COM C++

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

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

VB

```
Dim argsLoginParam As New Arguments

Dim WithEvents session As ctios. Session

Dim WithEvents agent As ctios. Agent

argsLoginParam .AddItem(
"agentid=23814;instrument=23815;password=1234")

agent.Login (argsLoginParam)

agent.Logout(argsLoginParam)
```

MakeCall

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

Table 5-4 Make Call Optional Parameters

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

Table 5-4 Make Call Optional Parameters (continued)

Parameter	Description
AuthorizationCode	An authorization code needed to access the resources required to initiate the call.
AccountCode	A cost-accounting or client number used by the peripheral for charge-back purposes.

Syntax

C++

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

COM (standard COM API)

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

Input Parameters

rArguments

Arguments array that holds parameters from Table 5-4.

pVariantArgs

Arguments array that holds parameters from Table 5-4.

Return Values

C++

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

COM

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

Examples

C++

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

COM C++

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

MonitorAgentTeam

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

Syntax

C++

```
virtual int MonitorAgentTeam( bool bStart, int TeamId );
```

COM (standard COM API)

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

Input Parameters

.bStart

Specifies whether to start or stop monitoring

TeamId

Team ID for the agent teams to be monitored

Return Values

C++

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

COM

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

Examples

C++

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

COM C++

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

MonitorAgentTeamAll

The MonitorAgentTeamAll method enables monitoring of all agent teams.

Syntax

C++

```
virtual int MonitorAgentTeamAll ( bool bStart);
```

COM (standard COM API)

```
HRESULT MonitorAgentTeamAll ( [in] VARIANT_BOOL bStart);
```

Input Parameters

bStart

Specifies whether to start or stop monitoring all agent teams.

Return Values

C++

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

COM

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

Examples

C++

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

COM C++

```
HRESULT hr =S_OK;
IAgentPtr pAgent = NULL;
// First, get agent object and make sure it is valid pointer
if (pAgent)
{
```

```
hr = pAgent-> MonitorAgentTeamAll ((bStart);
if(FAILED(hr))
{
     // You might want to log an error description HERE
}
pAgent = NULL;
}
return hr;
```

OnEvent (C++ Only)

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

Syntax

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

Input Parameters

iEventID

A unique number that points to a unique event.

rEventParam

Parameters for an event

Return Values

None.

Examples

```
CAgent * pAgent = NULL;
Arguments & rArgs;
int nRet;
// Get a valid agent pointer
```

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

SendUserMessage/SendChatMessage

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

Table 5-5 SendUserMessage/SendChatMessage Parameters

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

Syntax

C++

```
virtual int SendUserMessage(Arguments & rArguments);
virtual int SendChatMessage(Arguments & rArguments);
```

COM (standard COM API)

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

Input Parameters

rArguments

Array that contains parameters from Table 5-5.

nDistribution

Indicates to whom this message is to be sent.

pVariantArgs

Array that contains parameters fromTable 5-5.

Return Values

C++

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

COM

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

Examples

C++

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

COM C++

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

SetAgentState

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

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

Syntax

C++

```
virtual int SetAgentState(Arguments & rArguments);
```

COM (standard COM API)

```
HRESULT SetAgentState ( [in]  VARIANT * pVariantArgs );
```

Input Parameters

rArguments

Contains the new agent state.

pVariantArgs

Contains the new agent state.

Return Values

C++

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

COM

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

Examples

C++

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

// Check if the method fails
}
```

COM C++

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

SetMonitoredAgent

The SetMonitoredAgent method sets a agent object to be monitored.

Syntax

C++

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

COM (standard COM API)

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

Input Parameters

pAgent / pIAgent

Pointer to the agent object to be monitored.

bMonitor / bMonitor

Whether to enable or disable agent monitoring.

Return Values

C++

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

COM

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

Examples

C++

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

// Check if the method fails
}
```

COM C++

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

SetMonitoredCall

The SetMonitoredCall method sets a call object to be monitored.

Syntax

C++

```
virtual int SetMonitoredCall( CCall * pCall );
```

COM (standard COM API)

```
HRESULT SetMonitoredCall   ( [in]  ICall *  pICall   );
```

Input Parameters

pCall

Pointer to the call object that needs to be monitored.

pICall

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

Return Values

C++

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

COM

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

Examples

C++

```
CAgent * pAgent = NULL;
CCall * pCall =NULL;
int nRet;
// Get a valid agent pointer
   if (pAgent)
  {
nRet = pAgent-> SetMonitoredCall (pCall);

// Check if the method fails
}
```

COM C++

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

SetValue

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

Syntax

```
SetValue (key)
```

Example

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

SuperviseCall

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

Syntax

C++

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

COM (standard COM API)

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

Input Parameters

nSuperviseMode

Supervisor or agent mode.

pAgent / pIAgent

Pointer to COM agent object.

pCall / pICall

Pointer to call object.

Return Values

C++

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

COM

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

Examples

C++

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

COM C++

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

SupervisorAssist

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

Syntax

C++

```
virtual int SupervisorAssistRequest();
```

COM (standard COM API)

```
HRESULT SupervisorAssist ();
```

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

```
 CAgent * pAgent = NULL;
int nRet;
// Get a valid agent pointer
   if (pAgent)
  {
nRet = pAgent-> SupervisorAssistRequest ();

// Check if the method fails
}
```

COM C++

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

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

IAgentEvents Interface

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

OnAgentStateChange

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

Field	Description
PeripheralID	The ICM PeripheralID of the ACD where the occurred.
PeripheralType	The type of the peripheral.
SkillGroupState	Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.
StateDuration	The number of seconds since the agent entered this state (typically 0).
SkillGroupNumber	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	The ICM SkillGroupID of the agent SkillGroup affected by the state change. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
AgentState	One of the values representing the current overall state of the associated agent.
EventReasonCode	A peripheral-specific code indicating the reason for the state change.
CTIClientSignature (optional)	The Client Signature of the CTI Client that is associated with this agent.

Field	Description
AgentID (optional)	The agent’s ACD login ID.
AgentExtension (optional)	The agent’s ACD teleset extension.
AgentInstrument (optional)	The agent’s ACD instrument number.

OnAgentStatistics

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

Field	Description
PeripheralID	The ICM PeripheralID of the ACD where the occurred.
PeripheralType	The type of the peripheral.
SkillGroupState	Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.

OnUserMessage

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

Field	Description
ICMCentralControllerTime	The current ICM Central Controller date and time.
Distribution	One of value of eDistributeToClient, eDistributeToSupervisor, eDistributeToTeam, or eDistributeToAll.
ClientID	The ClientID of the message sender.
Text	The text message provided by the sender.

CHAPTER 6

Call Object

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

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

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

Active Call

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

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

- The active call is set when the call is answered;
- The active call is set when another call is retrieved from hold;

- When the active call clears, the previous call in the stack is selected as the active call.

Methods

Table 6-1 lists the available call object methods.

Table 6-1 Call Object Methods

Method	Description
Alternate	Places the current call on hold and then either retrieves a previously held call or answers another call.
Answer	Answers a call that is in the alerting or ringing state.
Clear	Clears a call.
ClearConnection	Clears a call connection.
Conference	Conferences a third party to a call.
Deflect	Dialed number
GetElement	Retrieves a property from the Agent object based on the property's index value.
GetPropertyAttribute	Retrieves attribute information for a specified call property.
GetValue (also GetValueInt, GetValueString)	Retrieves a property from the Agent object based on the property's name key.
Hold	Places an active call on hold.
MakeConsultCall	Places an active call on hold and makes a new call.
OnEvent	Specifies an action to take when a call variable or call state changes.
Reconnect	Clears *the current* call and then retrieves a specified call.
Record	Lets the user record a call.
Retrieve	Retrieves a held call.

Table 6-1 Call Object Methods (continued)

Method	Description
SetCallData	Sets call and expanded call context (ECC) variables.
SendDTMFSignal	Requests the ACD transmit to send a sequence of DTMF tones.
SingleStepConference	Initiates a single step (blind) conference.
SingleStepTransfer	Initiates a single step (blind) transfer.
Snapshot	Issues a server request to update the current call information.
Transfer	Transfers a call to a third party

Alternate

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

Syntax

C++

```
int Alternate ( );
```

COM (standard COM API)

```
HRESULT Alternate ( );
```

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

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

   if (pCall)
  {
      nRet  =  pCall->Alternate();

                   // Check if the alternate call method failed
}
```

COM C++

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

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

return hr;
```

Answer

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

Syntax

C++

```
int Answer ( );
```

COM (standard COM API)

```
HRESULT Answer ( );
```

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

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

   if (pCall)
  {
      nRet  =  pCall-> Answer ();

                   // Check if the Answer call method failed
}

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

COM C++

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

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

return hr;
```

Clear

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

Syntax

C++

```
int Clear ( );
```

COM (standard COM API)

```
HRESULT Clear ( );
```

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

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

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

COM C++

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

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

return hr;
```

ClearConnection

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

Syntax

C++

```
int ClearConnection ( );
```

COM (standard COM API)

```
HRESULT ClearConnection ( );
```

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

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

 if (pCall)
{
      nRet  =  pCall-> ClearConnection ();

                   // Check if the ClearConnection call method failed
}

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

COM C++

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

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

return hr;
```

Conference

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

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

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

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

Syntax

C++

```
int Conference(CCall * pCall = NULL);
```

COM (standard COM API)

```
HRESULT Conference  ( [in]  VARIANT * pVariantArgs  );
```

Input Parameters

PCall

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

pVariantArgs

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

Return Values

C++

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

COM

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

Examples

C++

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

 if (pCall)
{
    //
    // You need a second call to make the conference, it also could be 
null
      nRet  =  pCall-> Conference (pAnotherCall);

                   // Check if the Conference call method failed
}

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

COM C++

```
HRESULT hr =S_OK;
ICall * pCall = NULL;
VARIANT * vAnotherCall = NULL;
// First, get call object and make sure it is valid pointer
if ( pCall )
{
// You need a second call to make the conference, it also could be
null
// and use variant to wrap it
hr = pCall-> Conference (vAnotherCall);

if(FAILED(hr))
{
     // You might want to log an error description HERE
}
// Release the call after you are done with it
pCall->Release();
}

return hr;
```

Deflect

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

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

Syntax

C++

```
int Deflect(string & strDialedNumber);
```

COM (standard COM API)

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

Input Parameters

strDialedNumber

String that contains a dialed number.

pVariantArgs

Pointer to a variant that wraps a dialed number.

Return Values

C++

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

COM

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

Examples

C++

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

 if (pCall)
{
    // Get a valid dialed number from somewhere
     nRet  =  pCall-> Deflect( strDialedNumber );

                   // Check if the Deflect call method failed
}

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

COM C++

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

return hr;
```

GetElement

The GetElement method is identical to the GetValue method, except that is uses an index value instead of a key. The index value is not related the order in which items are added or removed. The order of items in the Arguments array is never guaranteed. This method is useful for sequentially iterating over all items in Arguments. The index is one (1) based. The index parameter should never be less than one or equal to the number of elements in Arguments.

Syntax

```
GetElement (index )
```

GetPropertyAttribute

The GetPropertyAttribute method retrieves attribute information for any of the properties listed in Table 6-2. You can access properties with the GetValue Method. For additional information on GetPropertyAttribute, see Chapter 1, "Introduction."

Table 6-2 Call Properties

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

Table 6-2 Call Properties (continued)

Property	Type	Meaning
RouterCallKeyDay	integer	Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the ICM database. Only provided for Post-routed and Translation-routed calls.
RouterCallKeyCallID	integer	The call key created by the ICM. The ICM resets this counter at midnight.
CallWrapupData	string, maximum length 40	Call-related wrapup data.
CallStatus	integer	The status of a call at a client.

Syntax

```
GetPropertyAttribute( propertyname, attribute requested )
```

GetValue

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

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

Syntax

```
GetValue( key )
```

Example

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

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

' iterate over all statistics
Dim agent As ctios.Agent
Dim allStatistics As ctios.Arguments

allStatistics = agent.GetValue( eStatistics )
For Each stat In allStatistics
...
Next
```

Hold

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

Syntax

C++

```
int Hold();
```

COM (standard COM API)

```
HRESULT Hold();
```

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

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

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

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

COM C++

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

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

return hr;
```

MakeConsultCall

The MakeConsultCall method initiates the combined action of placing an active call on hold and then making a new call. By default, the call context data of theactive call is used to initialize the context data of the consultation call. The application may override some or all of the original call context in the consultation call by providing the desired values in this request.

The simplest form of the request only requires a dialed number. The request may also include the parameters in Table 6-3.

Table 6-3 MakeConsultCall Optional Parameters

Parameter	Description
CallPlacementType	A value specifying how the call is to be placed identified in the Call Placement Type Table.
CallMannerType	A value specifying additional call processing options identified in the Call Manner Type Table.
AlertRings	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	A value specifying additional peripheral-specific call options.
FacilityType	A value indicating the type of facility to be used.
AnsweringMachine	A value specifying the action to be taken if the call is answered by an answering machine.
Priority	This field should be set to TRUE if the call should receive priority handling.

Table 6-3 MakeConsultCall Optional Parameters (continued)

Parameter	Description
PostRoute	When this field is set to TRUE, the Post-Routing capabilities of the ICM are to be used to determine the new call destination.
UserToUserInfo	The ISDN user-to-user information.
Call and ECC Variables	Up to 10 call variables and any of the defined ECC variables or array variables.
CallWrapupData	Call-related wrapup data.
FacilityCode	A trunk access code, split extension, or other data needed to access the chosen facility.
AuthorizationCode	An authorization code needed to access the resources required to initiate the call.
AccountCode	A cost-accounting or client number used by the peripheral for charge-back purposes.

Syntax

C++

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

COM (standard COM API)

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

Input Parameters

rArguments

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

pVariantArgs

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

Return Values

C++

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

COM

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

Examples

C++

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

Int nRet = 0;

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

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

COM C++

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

return hr;
```

OnEvent (C++ Only)

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

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

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

Syntax

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

Input Parameters

iEventID

A unique number that points to a unique event.

rEventParam

Arguments array that contains parameters for an event.

Return Values

None.

Examples

```
C++
 CCall * pCall = NULL;
Arguments & rEventParam;

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

}

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

Reconnect

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

Syntax

C++

```
int Reconnect ( );
```

COM (standard COM API)

```
HRESULT Reconnect ( );
```

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

```
 CCall * pCall = NULL;

Int nRet = 0;

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

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

COM C++

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

return hr;
```

Record

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

Syntax

C++

```
int Record( bool bEnable, Arguments& rArguments );
```

COM (standard COM API)

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

Input Parameters

bEnable

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

rArguments

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

pVariantArgs

Pointer to a variant that wraps an argument array.

Return Values

C++

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

COM

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

Examples

C++

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

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

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

COM C++

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

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

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

return hr;
```

Retrieve

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

Syntax

C++

```
int Retrieve();
```

COM (standard COM API)

```
HRESULT Retrieve ( );
```

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

```
 CCall * pCall = NULL;

Int nRet = 0;

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

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

COM C++

```
HRESULT hr =S_OK;
ICall * pCall = NULL;
// First, get call object and make sure it is valid pointer
if ( pCall )
{
hr = pCall-> Retrieve ();

if(FAILED(hr))
{
     // You might want to log an error description HERE
}
// Release the call after you are done with it
pCall->Release();
}

return hr;
```

SendDTMFSignal

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

Syntax

C++

```
int SendDTMFSignal(Arguments & rArguments);
```

COM (standard COM API)

```
HRESULT SendDTMFSignal  ( [in]  VARIANT * pVariantArgs            );
```

Input Parameters

rArguments

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

pVariantArgs

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

Table 6-4 SendDTMFSignal Parameters

Parameter	Description
DTMFString	The sequence of tones to be generated.
ToneDuration (optional)	Dial number. The number to be dialed to establish the new call.
PauseDuration (optional)	One of the values specifying how the call is to be placed identified in the Call Placement Type Table.

Return Values

C++

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

COM

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

Examples

C++

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

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

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

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

COM C++

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

if(FAILED(hr))
{
     // You might want to log an error description HERE
}
// Release the call after you are done with it
pCall->Release();
}

return hr;
```

SetCallData

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

Syntax

C++

```
int SetCallData(Arguments & rArguments);
```

COM (standard COM API)

```
HRESULT SetCallData ( [in]  VARIANT * pVariantArgs );
```

Input Parameters

rArguments

Arguments array that contains call data parameters.

pVariantArgs

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

Return Values

C++

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

COM

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

Examples

C++

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

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

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

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

COM C++

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

if(FAILED(hr))
{
     // You might want to log an error description HERE
}
// Release the call after you are done with it
pCall->Release();
}
return hr;
```

VB

```
' update call variables using one call to SetCallData

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

Dim WithEvents args As New ctios.Arguments

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

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

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

call.SetCallData( args )
```

SingleStepConference

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

Syntax

C++

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

COM (standard COM API)

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

Input Parameters

rArguments

Arguments array that contains a consult call.

pVariantArgs

Pointer to a variant that contains a consult call

Return Values

C++

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

COM

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

Examples

C++

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

 if (pCall)
{
     // fill the argument with the consult call

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

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

COM C++

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

if(FAILED(hr))
{
     // You might want to log an error description HERE
}
// Release the call after you are done with it
pCall->Release();
}
return hr;
```

SingleStepTransfer

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

Syntax

C++

```
int SingleStepTransfer (Arguments & rArguments);
```

COM (standard COM API)

```
HRESULT SingleStepTransfer( [in]  VARIANT * pVariantArgs
```

Input Parameters

rArguments

Arguments array that contains a consult call.

pVariantArgs

Pointer to a variant that contains a consult call.

Return Values

C++

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

COM

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

Examples

C++

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

 if (pCall)
{
     // fill the argument with the consult call

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

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

COM C++

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

if(FAILED(hr))
{
     // You might want to log an error description HERE
}
// Release the call after you are done with it
pCall->Release();
}
return hr;
```

Snapshot

The Snapshot method issues a server request to update the current call information. If values are passed in the optional ArgumentList, the snapshot record will return the server's current call information for the requested arguments; otherwise all call information is returned. The server returns the requested values as an OnCallDataUpdate event with the following key/ value pair "snapshot=true".

Syntax

C++

```
int Snapshot ( );
```

COM (standard COM API)

```
HRESULT Snapshot ( );
```

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

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

 if (pCall)
{
     // fill the argument with the consult call

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

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

COM C++

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

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

if(FAILED(hr))
{
     // You might want to log an error description HERE
}
// Release the call after you are done with it
pCall->Release();
}
return hr;
```

Transfer

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

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

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

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

Syntax

C++

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

COM (standard COM API)

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

Input Parameters

PCall

Pointer to a Call object, the default value is NULL

pVariantArgs

Pointer to a variant that holds a pointer to a call object

Return Values

C++

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

COM

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

Examples

C++

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

 if (pCall)
{
    //
    // You need a second call to make the conference, it also could be 
null
      nRet  =  pCall-> Transfer (pAnotherCall);

                   // Check if the Transfer call method failed
}

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

COM C++

```
HRESULT hr =S_OK;
ICall * pCall = NULL;
VARIANT * vAnotherCall = NULL;
// First, get call object and make sure it is valid pointer
if ( pCall )
{
// You need a second call to make the Transfer, it also could be null
// and use variant to wrap it
hr = pCall-> Transfer (vAnotherCall);

if(FAILED(hr))
{
     // You might want to log an error description HERE
}
// Release the call after you are done with it
pCall->Release();
}

return hr;
```

ICallEvents Interface

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

OnCallBegin

The OnCallBegin event is generated at the first association between a call and the CTI Client. The event passes the call identifier and the initial call context data. The ConnectionCallID identifies the call, and the ConnectionDeviceIDType and ConnectionDeviceID uniquely identify the client's local call connection, if any, or another valid call connection. This message always precedes any other event messages for that call.

Subsequent changes to the call context data (if any) are forwarded by an OnCallDataUpdate event. The event contains the changed call data.

Note There can be multiple calls with the same ConnectionCallID value.

Field	Description
CallIdentifier	Unique reference generated for a call at client.
PeripheralID	The ICM PeripheralID of the ACD where the call activity occurred.

Field	Description
PeripheralType	The type of the peripheral.
ConnectionDeviceIDType	Indicates the type of ConnectionDeviceID value.
ConnectionDeviceID	The device identifier of the connection between the call and the device.
ConnectionCallID	The Call ID value assigned to this call by the peripheral or the ICM.
NumCTIClients	The number of CTI Clients previously associated with this call. This value also indicates the number of CTI Client signatures and timestamps that are present in the floating part of the message.
Call and ECC Variables	Initial array of call and ecc variables.
CallType	The general classification of the call type.
ANI (optional)	The calling line ID of the caller.
UserToUserInfo (optional)	The ISDN user-to-user information element. unspecified, up to 131 bytes.
DNIS (optional)	The DNIS provided with the call.
DialedNumber (optional)	The number dialed.
CallerEnteredDigits (optional)	The digits entered by the caller in response to IVR prompting.
RouterCallKeyDay (optional)	Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the ICM database. Only provided for Post-routed and Translation-routed calls.
RouterCallKeyCallID (optional)	The call key created by the ICM. The ICM resets this counter at midnight.
CallWrapupData (optional)	Call-related wrapup data.

Field	Description
CTIClientSignature (optional)	The Client Signature of a CTI Client that was previously associated with this call. There may be more than one CTIClientSignature field in the message (see NumCTIClients).
CTIClientTimestamp (optional)	The date and time that the preceding CTI Client signature was first associated with the call. There may be more than one CTIClientTimestamp field in the message (see NumCTIClients). This field always immediately follows the CTIClientSignature field to which it refers.

OnCallEnd

The OnCallEnd event is generated when the association between a call and the CTI Client is dissolved. The OnCallEnd event is the last event received for a Call. The event does not necessarily indicate that the subject call has been terminated.

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

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.

Field	Description
CallIdentifier	Unique reference generated for a call at client.
PeripheralID	The ICM PeripheralID of the ACD where the call is located.
PeripheralType	The type of the peripheral. type: integer.
NumCTIClients	The number of CTI Clients associated with this call. This value also indicates the number of CTI Client signatures and timestamps that are present in the floating part of the message.
CallType	The general classification of the call type.
ConnectionDeviceIDType	Indicates the type of ConnectionDeviceID value.
ConnectionDeviceID	The previous device identifier of the call connection.
ConnectionCallID	The Call ID value previously assigned to this call by the peripheral or the ICM.
NewConnectionDeviceIDType	Indicates the type of the connection identifier supplied in the NewConnectionDeviceID field.
NewConnectionDeviceID	The new identifier of call connection.
NewConnectionCallID	The new Call ID value assigned to this call by the peripheral or the ICM.
ANI (optional)	The calling line ID of the caller.
UserToUserInfo (optional)	The ISDN user-to-user information element.
DNIS (optional)	The DNIS provided with the call.

Field	Description
DialedNumber (optional)	The number dialed.
CallerEnteredDigits (optional)	The digits entered by the caller in response to IVR prompting.
RouterCallKeyDay (optional)	Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the ICM database. Only provided for Post-routed and Translation-routed calls.
RouterCallKeyCallID (optional)	The call key created by the ICM. The ICM resets this counter at midnight.
Call, ECC, and ECC Array Variables (optional)	Call-related variable data.
CallWrapupData (optional)	Call-related wrapup data.
Snapshot (optional)	TRUE if in response to a snapshot request.
CTIClientSignature (optional)	The Client Signature of a CTI Client that was previously associated with this call. There may be more than one CTIClientSignature field in the message (see NumCTIClients).
CTIClientTimestamp (optional)	The date and time that the preceding CTI Client signature was first associated with the call. There may be more than one CTIClientTimestamp field in the message (see NumCTIClients). This field always immediately follows the CTIClientSignature field to which it refers.

OnCallDelivered

The OnCallDelivered event may be generated when the call arrives at the agent's teleset. Note that when a call is alerting at a teleset, there is no formal connection between the call and the alerting device, so the ConnectionDeviceID of the *calling* connection is reported in this message.

Field	Description
CallIdentifier	Unique reference generated for a call at client.
PeripheralID	The ICM PeripheralID of the ACD where the call is located.
PeripheralType	The type of the peripheral.
ConnectionDeviceIDType	Indicates the type of ConnectionDeviceID value.
ConnectionDeviceID	The device identifier of the connection between the call and the device.
ConnectionCallID	The Call ID value assigned to this call by the peripheral or the ICM.
LineHandle	When LocalConnectionState is LCS_ALERTING, this field identifies the alerting teleset line, if known. Otherwise this field is set to 0xffff.
LineType	Indicates the type of the teleset line given in the LineHandle field, if any.
ServiceNumber	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	The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.
AlertingDeviceType	Indicates the type of the device identifier supplied in the AlertingDeviceID field.
CallingDeviceType	Indicates the type of the device identifier supplied in the CallingDeviceID field.
CalledDeviceType	Indicates the type of the device identifier supplied in the CalledDeviceID field.

Field	Description
LastRedirectDeviceType	Indicates the type of the device identifier supplied in the LastRedirectDeviceID field.
LocalConnectionState	The state of the local end of the connection. When a call first appears at an ACD, the LocalConnectionState will be LCS_INITIATE. When the call is delivered to an agent teleset, the LocalConnectionState will be LCS_ALERTING for AllEvent clients and the client associated with the alerting device. For all other clients this field will indicate the state of the client's local connection to the call.
EventCause	Indicates a reason or explanation for the occurrence of the event.
AlertingDeviceID (optional)	The device identifier of the device that is alerting.
CallingDeviceID (optional)	The device identifier of the calling device.
CalledDeviceID (optional)	The device identifier of the originally called device.
LastRedirectDeviceID (optional)	The device identifier of the previously alerted device.

OnCallEstablished

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

Field	Description
CallIdentifier	Unique reference generated for a call at client.
PeripheralID	The ICM PeripheralID of the ACD where the call is located.
PeripheralType	The type of the peripheral.
ConnectionDeviceIDType	Indicates the type of ConnectionDeviceID value.
ConnectionDeviceID	The device identifier of the connection between the call and the device.
ConnectionCallID	The Call ID value assigned to this call by the peripheral or the ICM.
LineHandle	When LocalConnectionState is LCS_ALERTING, this field identifies the alerting teleset line, if known. Otherwise this field is set to 0xffff.
LineType	Indicates the type of the teleset line given in the LineHandle field, if any.
ServiceNumber	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	The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.
SkillGroupNumber	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.

Field	Description
SkillGroupID	The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
AnsweringDeviceType	Indicates the type of the device identifier supplied in the AnsweringDeviceID field.
CallingDeviceType	Indicates the type of the device identifier supplied in the CallingDeviceID field.
CalledDeviceType	Indicates the type of the device identifier supplied in the CalledDeviceID field.
LastRedirectDeviceType	Indicates the type of the device identifier supplied in the LastRedirectDeviceID field.
LocalConnectionState	The state of the local end of the connection.
EventCause	Indicates a reason or explanation for the occurrence of the event.
AnsweringDeviceID (optional)	The device identifier of the device that is alerting.
CallingDeviceID (optional)	The device identifier of the calling device.
CalledDeviceID (optional)	The device identifier of the originally called device.
LastRedirectDeviceID (optional)	The device identifier of the previously alerted device.

OnCallHeld

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

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

OnCallRetrieved

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

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

OnCallCleared

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

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

OnCallConnectionCleared

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

Field	Description
CallIdentifier	Unique reference generated for a call at client.
PeripheralID	The ICM PeripheralID of the ACD where the call activity occurred.
PeripheralType	The type of the peripheral.
ConnectionDeviceIDType	Indicates the type of ConnectionDeviceID value.

ConnectionDeviceID	The device identifier of the connection between the call and the device.
ConnectionCallID	The Call ID value assigned to this call by the peripheral or the ICM.
ReleasingDeviceType	Indicates the type of the device identifier supplied in the ReleasingDeviceID field.
LocalConnectionState	The state of the local end of the connection.
EventCause	Indicates a reason or explanation for the occurrence of the event.
ReleasingDeviceID	The device identifier of the device that cleared the connection.

OnCallOriginated

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

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

Field	Description
ServiceNumber	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	The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.
SkillGroupNumber	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	The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
CallingDeviceType	Indicates the type of the device identifier supplied in the CallingDeviceID field.
CalledDeviceType	Indicates the type of the device identifier supplied in the CalledDeviceID field.
LocalConnectionState	The state of the local end of the connection.
EventCause	Indicates a reason or explanation for the occurrence of the event.
CallingDeviceID (optional)	The device identifier of the calling device.
CalledDeviceID	The device identifier of the originally called device.

OnCallFailed

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

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

OnCallTransferConferenceInitiated

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

Field	Description
CallIdentifier	Unique reference generated for a call at client.

OnCallTransferred

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

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

Field	Description
SkillGroupID	The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
SecondaryDeviceIDType	Indicates the type of the connection identifier supplied in the SecondaryDeviceID field.
SecondaryCallID	The Call ID value assigned to the secondary call by the peripheral or the ICM.
TransferringDeviceType	Indicates the type of the device identifier supplied in the TransferringDeviceID field.
TransferredDeviceType	Indicates the type of the device identifier supplied in the TransferredDeviceID field.
LocalConnectionState	The state of the local end of the connection.
EventCause	Indicates a reason or explanation for the occurrence of the event.
PrimaryDeviceID	The connection identifier of the primary call connection.
SecondaryDeviceID	The connection identifier of the secondary call connection.
TransferringDeviceID (optional)	The device identifier of the device that transferred the call.

Field	Description
TransferredDeviceID (optional)	The device identifier of the device to which the call was transferred.
ConnectedParties (optional)	The Call ID, DeviceIDType, and DeviceID of parties connected to this conference call, up to a maximum of 16. This is an array of arguments, each containing the call identification information for each connected party.

OnCallConferenced

The joining of calls into a conference call may generate an OnCallConferenced event.

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

Field	Description
SkillGroupID	The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
SecondaryDeviceIDType	Indicates the type of the connection identifier supplied in the SecondaryDeviceID field.
SecondaryCallID	The Call ID value assigned to the secondary call by the peripheral or the ICM.
ControllerDeviceType	Indicates the type of the device identifier supplied in the ControllerDeviceID field.
AddedPartyDeviceType	Indicates the type of the device identifier supplied in the AddedPartyDeviceID field.
LocalConnectionState	The state of the local end of the connection.
EventCause	Indicates a reason or explanation for the occurrence of the event.
PrimaryDeviceID	The connection identifier of the primary call connection.
SecondaryDeviceID	The connection identifier of the secondary call connection.
ControllerDeviceID (optional)	The device identifier of the conference controller device.

Field	Description
AddedPartyDeviceID (optional)	The device identifier of the device added to the call.
ConnectedParties (optional)	The Call ID, DeviceIDType, and DeviceID of parties connected to this conference call, up to a maximum of 16. This is an array of arguments, each containing the call identification information for each connected party.

OnCallDiverted

The removal of a call from a previous delivery target may generate an OnCallDiverted event.

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

Field	Description
ServiceID	The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.
DivertingDeviceType	Indicates the type of the device identifier supplied in the DivertingDeviceID field.
CalledDeviceType	Indicates the type of the device identifier supplied in the CalledDeviceID field.
LocalConnectionState	The state of the local end of the connection.
EventCause	Indicates a reason or explanation for the occurrence of the event.
DivertingDeviceID (optional)	The device identifier of the device from which the call was diverted.
CalledDeviceID (optional)	The device identifier of the device to which the call was diverted.

OnTranslationRoute

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

Field	Description
ANI (optional)	The calling line ID of the caller.
UserToUserInfo (optional)	The ISDN user-to-user information element.
DNIS	The DNIS of the expected call.
DialedNumber (optional)	The number dialed.
CallerEnteredDigits (optional)	The digits entered by the caller in response to IVR prompting.

Field	Description
RouterCallKeyDay	Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the ICM database.
RouterCallKeyCallID	The call key created by the ICM. The ICM resets this counter at midnight.
Call and ECC Variables (optional)	Call-related variable data.

OnCallEnterpriseAgent

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

Field	Description
ServiceNumber	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	The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.
SkillGroupNumber	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.

Field	Description
SkillGroupID	The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
AgentInstrument	The agent instrument that the call will be routed to.
RouterCallKeyDay	Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the ICM database
RouterCallKeyCallID	The call key created by the ICM. The ICM resets this counter at midnight.
ANI (optional)	The calling line ID of the caller.
UserToUserInfo (optional)	The ISDN user-to-user information element.
DialedNumber (optional)	The number dialed.
CallerEnteredDigits (optional)	The digits entered by the caller in response to IVR prompting.
Call and ECC Variables (optional)	Call-related variable data.

OnCallPreEventAbort

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

Field	Description
AgentInstrument	The agent instrument that the call was to have been routed to.
RouterCallKeyDay	Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the ICM database
RouterCallKeyCallID	The call key created by the ICM. The ICM resets this counter at midnight.

OnCallRequestFailed

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

Field	Description
Request	ID of request

OnAgentPrecallEvent

The OnAgentPrecallEvent event is roughly equivalent to a TranslationRouteEvent as far as CTI OS is concerned. A non-ACD agent (e.g enterprise agent) will get this event when he has been selected to receive the call, but before it is actually delivered. As a result we will identify the call by it's RouterCallKeyDay and RouterCallKeyCallID. We can receive events for the call while it is in queue, and we can set the call data. However, since this event happens before the BEGIN_CALL_EVENT, we will not know the PeripheralID, and thus cannot create a peripheral-specific behavior implementation object. All we can do at this point is get and set data.

Field	Description
RouterCallKeyDay	Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the ICM database. Only provided for Post-routed and Translation-routed calls.
RouterCallKeyCallID	The call key created by the ICM. The ICM resets this counter at midnight.
AgentInstrument	The agent instrument that the call will be routed to.
NumNamedVariables	Number of Named variables
NumNamedArrays	Number of Named Array
ServiceNumber	The service that the call is attributed to, as known to the peripheral.
ServiceID	The ICM ServiceID of the service that the call is attributed to.
SkillGroupNumber	The number of the agent SkillGroup the call is attributed to, as known to the peripheral.
SkillGroupID	The ICM SkillGroupID of the agent SkillGroup the call is attributed to.
SkillGroupPriority	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
ANI	The calling line ID of the caller.
UserToUserInfo	The ISDN user-to-user information element.
DialedNumber	The number dialed.
CallerEnteredDigits	The digits entered by the caller in response to IVR prompting.
CallVariable 1 through 10	Call-related variable data.

OnAgentPrecallAbortEvent

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

Field	Description
RouterCallKeyDay	Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the ICM database. Only provided for Post-routed and Translation-routed calls.
RouterCallKeyCallID	The call key created by the ICM. The ICM resets this counter at midnight.
AgentInstrument	The agent instrument that the call will be routed to.

OnCallServiceInitiatedEvent

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

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

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

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.

Field	Description
ConnectionDeviceID	The identifier of the connection between the call and the device.
ConnectionDeviceIDType	Indicates the type of the connection identifier supplied in the ConnectionDeviceID
QueuedDeviceID	The device identifier of the queuing device.

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

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

OnCallDequeuedEvent

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

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

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

OnCallReachedNetworkEvent

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

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

OnControlFailureConf

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

Field	Description
FailureCode	Code ID
ErrorCode	Description of an error
AgentID	Agent ID that represents a specific client
PeripheralID	Peripheral ID

OnSnapshotCallConf

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

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

Field	Description
RouterCallKeyDay	Together with the RouterCallKeyCallID field forms the unique 64-bit key for locating this call's records in the ICM database. Only provided for Post-routed and Translation-routed calls.
RouterCallKeyCallID	The call key created by the ICM. The ICM resets this counter at midnight.
CallWrapupData	Call-related wrapup data.
Ecc variables	Call data variables
CallVariable 1 through 10	Call-related variable data.

OnServiceInitiated

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

Field	Description
PeripheralID	The ICM PeripheralID of the ACD where the call activity occurred.
PeripheralType	The type of the peripheral.
ConnectionDeviceIDType	Indicates the type of ConnectionDeviceID value.
ConnectionDeviceID	The device identifier of the connection between the call and the device.
ConnectionCallID	The Call ID value assigned to this call by the peripheral or the ICM.
LineHandle	Identifies the teleset line being used.
LineType	Indicates the type of the teleset line.
ServiceNumber	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.

Field	Description
ServiceID	The ICM ServiceID of the service that the call is attributed to. May contain the special value NULL_SERVICE when not applicable or not available.
SkillGroupNumber	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	The ICM SkillGroupID of the agent SkillGroup the call is attributed to. May contain the special value NULL_SKILL_GROUP when not applicable or not available.
SkillGroupPriority	The priority of the skill group, or 0 when skill group priority is not applicable or not available.
CallingDeviceType	Indicates the type of the device identifier supplied in the CallingDeviceID field.
LocalConnectionState	The state of the local end of the connection.
EventCause	Indicates a reason or explanation for the occurrence of the event.
CalledDeviceID (optional)	The device identifier of the device to which the call was diverted.

SkillGroup Object

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

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

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

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

Methods

Table 7-1 lists the SkillGroup object methods.

Table 7-1 SkillGroup Object Methods

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

EnableSkillGroupStatistics/DisableSkillGroupStatistics

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

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

Syntax

C++

```
int EnableSkillGroupStatistics(Arguments & rArguments);
int DisableSkillGroupStatistics (Arguments & rArguments);
```

COM (standard COM API)

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

Parameters

None.

Return Values

C++

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

COM

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

Examples

C++

```
 CSkillGroup * pSkillGroup = NULL;
 int nRet = 0;
Arguments & rArgs;
// Get a valid SkillGroup pointer
   if (pSkillGroup)
  {
       //
       nRet  = pSkillGroup -> EnableSkillGroupStatistics (rArgs);

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

COM C++

```
HRESULT hr =S_OK;
ISkillGroupPtr  pSkillGroup = NULL;

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

return hr;
```

VB

```
Dim skillNumbers As New ctios.Arguments

skillNumbers.AddItem( "Skill1", 7 )

skillNumbers.AddItem( "Skill2", 99 )

Dim enableArgs As New ctios.Arguments

enableArgs.AddItem( "SkillGroupNumbers", skillNumbers)
agent.EnableSkillGroupStatistics (enableArgs)

Dim enableArgs As New ctios.Arguments

agent.EnableSkillGroupStatistics (enableArgs)
```

GetPropertyAttribute

The GetPropertyAttribute method gets, sets, or retrieves attribute information for any of the properties listed in Table 7-2. For additional information on GetPropertyAttribute, see Chapter 1, “Introduction.” Properties can be accessed using GetValue.

Table 7-2 SkillGroup Properties

Property	Description
SkillGroupNumber	The number of the skill group from the Peripheral.
SkillGroupID	The ICM SkillGroupID of the SkillGroup, if available.
SkillGroupName	The ICM SkillGroupName of the SkillGroup, if available.
SkillGroupState	Values representing the current state of the associated agent with respect to the indicated Agent Skill Group.
ClassIdentifier	Value represents skillgroup class

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

Note Not all the statistics values listed in Table 7-3 are present in every system configuration. Whether a particular statistic value is available depends both on the protocol version of CTIServer with which CTI OS connects and on the peripheral on which the agent resides.

Table 7-3 SkillGroup Statistics

Statistic	Description
AgentsLoggedOn	Number of agents that are currently logged on to the skill group
AgentsAvail	Number of agents in the skill group in Available state.
AgentsNotReady	Number of agents in the skill group in Not Ready state.
AgentsReady	Number of agents in the skill group in Ready state.
AgentsTalkingIn	Number of agents in the skill group currently talking on inbound calls.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
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.
AgentsWorkNotReady	Number of agents in the skill group in Work Not Ready state.
AgentsWorkReady	Number of agents in the skill group in Work Ready state.
AgentsBusyOther	The number of agents in the skill group currently busy with calls assigned to other skill groups.
AgentsReserved	Number of agents in the skill group in Reserved state.
AgentsHold	Number of agents in the skill group with calls currently on hold.
AgentsTalkingAutoOut	Number of agents in the skill group currently talking on AutoOut (predictive) calls.
AgentsTalkingPreview	Number of agents in the skill group currently talking on outbound Preview calls.
AgentsTalkingReservation	Number of agents in the skill group currently talking on agent reservation calls.
RouterCallsQNow	The number of calls currently queued by the ICM call router to the skill group. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
LongestRouterCallQNow	The queue time in seconds of the ICM call router queued call that has been queued to this skill group the longest. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
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.
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 during the last five minutes.
LoggedOnTimeTo5	Total time, in seconds, agents in the skill group were logged on during the last five minutes.
NotReadyTimeTo5	Total seconds agents in the skill group were in the Not Ready state during the last five minutes.
AgentOutCallsTo5	Total number of completed outbound ACD calls made by agents in the skill group during the last five minutes.
AgentOutCallsTalkTimeTo5	Total talk time, in seconds, for completed outbound ACD calls handled by agents in the skill group during the last five minutes. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
AgentOutCallsTimeTo5	Total handle time, in seconds, for completed outbound ACD calls handled by agents in the skill group during the last five minutes. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
AgentOutCallsHeldTo5	The total number of completed outbound ACD calls agents in the skill group have placed on hold at least once during the last five minutes.
AgentOutCallsHeldTimeTo5	Total number of seconds outbound ACD calls were placed on hold by agents in the skill group during the last five minutes.
HandledCallsTo5	The number of inbound ACD calls handled by agents in the skill group during the last five minutes.
HandledCallsTalkTimeTo5	Total talk time in seconds for Inbound ACD calls counted as handled by agents in the skill group during the last five minutes. Includes hold time associated with the call.
HandledCallsAfterCallTimeTo5	Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the skill group during the last five minutes.
HandledCallsTimeTo5	Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the skill group during the last five minutes. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
IncomingCallsHeldTo5	The total number of completed inbound ACD calls agents in the skill group placed on hold at least once during the last five minutes.
IncomingCallsHeldTimeTo5	Total number of seconds completed inbound ACD calls were placed on hold by agents in the skill group during the last five minutes.
InternalCallsRcvdTo5	Number of internal calls received by agents in the skill group during the last five minutes.
InternalCallsRcvdTimeTo5	Number of seconds spent on internal calls received by agents in the skill group during the last five minutes.
InternalCallsHeldTo5	The total number of internal calls agents in the skill group placed on hold at least once during the last five minutes.
InternalCallsHeldTimeTo5	Total number of seconds completed internal calls were placed on hold by agents in the skill group during the last five minutes.
AutoOutCallsTo5	Total number of AutoOut (predictive) calls completed by agents in the skill group during the last five minutes.
AutoOutCallsTalkTimeTo5	Total talk time in seconds for completed AutoOut (predictive) calls handled by agents in the skill group during the last five minutes. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
AutoOutCallsTimeTo5	Total handle time in seconds for completed AutoOut (predictive) calls handled by agents in the skill group during the last five minutes. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
AutoOutCallsHeldTo5	Total number of completed AutoOut (predictive) calls that agents in the skill group have placed on hold at least once during the last five minutes.
AutoOutCallsHeldTimeTo5	Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group during the last five minutes.
PreviewCallsTo5	Total number of outbound Preview calls completed by agents in the skill group during the last five minutes.
PreviewCallsTalkTimeTo5	Total talk time in seconds for completed outbound Preview calls handled by agents in the skill group during the last five minutes. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.
PreviewCallsTimeTo5	Total handle time in seconds for completed outbound Preview calls handled by agents in the skill group during the last five minutes. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
PreviewCallsHeldTo5	Total number of completed outbound Preview calls that agents in the skill group have placed on hold at least once during the last five minutes.
PreviewCallsHeldTimeTo5	Total number of seconds outbound Preview calls were placed on hold by agents in the skill group during the last five minutes.
ReservationCallsTo5	Total number of agent reservation calls completed by agents in the skill group during the last five minutes.
ReservationCallsTalkTimeTo5	Total talk time in seconds for completed agent reservation calls handled by agents in the skill group during the last five minutes. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.
ReservationCallsTimeTo5	Total handle time in seconds for completed agent reservation calls handled by agents in the skill group during the last five minutes. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
ReservationCallsHeldTo5	Total number of completed agent reservation calls that agents in the skill group have placed on hold at least once during the last five minutes.
ReservationCallsHeldTimeTo5	Total number of seconds agent reservation calls were placed on hold by agents in the skill group during the last five minutes.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
BargeInCallsTo5	Total number of supervisor call barge-ins completed in the skill group during the last five minutes.
InterceptCallsTo5	Total number of supervisor call intercepts completed in the skill group during the last five minutes.
MonitorCallsTo5	Total number of supervisor call monitors completed in the skill group during the last five minutes.
WhisperCallsTo5	Total number of supervisor call whispers completed in the skill group during the last five minutes.
EmergencyCallsTo5	Total number of emergency calls completed in the skill group during the last five minutes.
CallsQ5	The number of calls queued to the skill group during the last five minutes. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
CallsQTime5	The total queue time, in seconds, of calls queued to the skill group during the last five minutes. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
LongestCallQ5	The longest queue time, in seconds, of all calls queued to the skill group during the last five minutes. This field is set to 0xFFFFFFFF when this value is unknown or unavailable.
AvailTimeToHalf	Total seconds agents in the skill group were in the Available state during the last half-hour.
LoggedOnTimeToHalf	Total time, in seconds, agents in the skill group were logged on during the last half-hour.
NotReadyTimeToHalf	Total seconds agents in the skill group were in the Not Ready state during the last half-hour.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
AgentOutCallsToHalf	Total number of completed outbound ACD calls made by agents in the skill group during the last half-hour.
AgentOutCallsTalkTimeTo Half	Total talk time, in seconds, for completed outbound ACD calls handled by agents in the skill group during the last half-hour. The value includes the time spent from the call being initiated by the agent to the time the agent begins after call work for the call. The time includes hold time associated with the call.
AgentOutCallsTimeToHalf	Total handle time, in seconds, for completed outbound ACD calls handled by agents in the skill group during the last half-hour. The value includes the time spent from the call being initiated by the agent to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
AgentOutCallsHeldToHalf	The total number of completed outbound ACD calls agents in the skill group have placed on hold at least once during the last half-hour.
AgentOutCallsHeldTime ToHalf	Total number of seconds outbound ACD calls were placed on hold by agents in the skill group during the last half-hour.
HandledCallsToHalf	The number of inbound ACD calls handled by agents in the skill group during the last half-hour.
HandledCallsTalkTimeToHalf	Total talk time in seconds for Inbound ACD calls counted as handled by agents in the skill group during the last half-hour. Includes hold time associated with the call.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
HandledCallsAfterCallTime ToHalf	Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the skill group during the last half-hour.
HandledCallsTimeToHalf	Total handle time, in seconds, for inbound ACD calls counted as handled by agents in the skill group during the last half-hour. The time spent from the call being answered by the agent to the time the agent completed after call work time for the call. Includes hold time associated with the call.
IncomingCallsHeldToHalf	The total number of completed inbound ACD calls agents in the skill group placed on hold at least once during the last half-hour.
IncomingCalls HeldTimeToHalf	Total number of seconds completed inbound ACD calls were placed on hold by agents in the skill group during the last half-hour.
InternalCallsRcvdToHalf	Number of internal calls received by agents in the skill group during the last half-hour.
InternalCallsRcvdTimeToHalf	Number of seconds spent on internal calls received by agents in the skill group during the last half-hour.
InternalCallsHeldToHalf	The total number of internal calls agents in the skill group placed on hold at least once during the last half-hour.
InternalCalls HeldTimeToHalf	Total number of seconds completed internal calls were placed on hold by agents in the skill group during the last half-hour.
AutoOutCallsToHalf	Total number of AutoOut (predictive) calls completed by agents in the skill group during the last half-hour.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
AutoOutCallsTalkTimeToHalf	Total talk time in seconds for completed AutoOut (predictive) calls handled by agents in the skill group during the last half-hour. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.
AutoOutCallsTimeToHalf	Total handle time in seconds for completed AutoOut (predictive) calls handled by agents in the skill group during the last half-hour. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
AutoOutCallsHeldToHalf	Total number of completed AutoOut (predictive) calls that agents in the skill group have placed on hold at least once during the last half-hour.
AutoOutCallsHeldTimeToHalf	Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group during the last half-hour.
PreviewCallsToHalf	Total number of outbound Preview calls completed by agents in the skill group during the last half-hour.
PreviewCallsTalkTimeToHalf	Total talk time in seconds for completed outbound Preview calls handled by agents in the skill group during the last half-hour. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
PreviewCallsTimeToHalf	Total handle time in seconds for completed outbound Preview calls handled by agents in the skill group during the last half-hour. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
PreviewCallsHeldToHalf	Total number of completed outbound Preview calls that agents in the skill group have placed on hold at least once during the last half-hour.
PreviewCallsHeldTimeToHalf	Total number of seconds outbound Preview calls were placed on hold by agents in the skill group during the last half-hour.
ReservationCallsToHalf	Total number of agent reservation calls completed by agents in the skill group during the last half-hour.
ReservationCallsTalkTime ToHalf	Total talk time in seconds for completed agent reservation calls handled by agents in the skill group during the last half-hour. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.
ReservationCallsTimeToHalf	Total handle time in seconds for completed agent reservation calls handled by agents in the skill group during the last half-hour. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
ReservationCallsHeldToHalf	Total number of completed agent reservation calls that agents in the skill group have placed on hold at least once during the last half-hour.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
ReservationCallsHeldTime ToHalf	Total number of seconds agent reservation calls were placed on hold by agents in the skill group during the last half-hour.
BargeInCallsToHalf	Total number of supervisor call barge-ins completed in the skill group during the last half-hour.
InterceptCallsToHalf	Total number of supervisor call intercepts completed in the skill group during the last half-hour.
MonitorCallsToHalf	Total number of supervisor call monitors completed in the skill group during the last half-hour.
WhisperCallsToHalf	Total number of supervisor call whispers completed in the skill group during the last half-hour.
EmergencyCallsToHalf	Total number of emergency calls completed in the skill group during the last half-hour.
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.
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.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
LoggedOnTimeToday	Total time, in seconds, agents in the skill group were logged on.
NotReadyTimeToday	Total seconds agents in the skill group were in the Not Ready state.
AgentOutCallsToday	Total number of completed outbound ACD calls made by agents in the skill group.
AgentOutCallsTalkTimeToday	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.
HandledCallsTalkTimeToday	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 7-3 SkillGroup Statistics (continued)

Statistic	Description
HandledCallsAfter CallTimeToday	Total after call work time in seconds for Inbound ACD calls counted as handled by agents in the skill group.
HandledCallsTimeToday	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.
InternalCallsRcvdToday	Number of internal calls received by agents in the skill group.
InternalCallsRcvdTimeToday	Number of seconds spent on internal calls received by agents in the skill group.
InternalCallsHeldToday	The total number of internal calls agents in the skill group placed on hold at least once.
InternalCallsHeldTimeToday	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 during the current day.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
AutoOutCallsTalkTimeToday	Total talk time in seconds for completed AutoOut (predictive) calls handled by agents in the skill group during the current day. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.
AutoOutCallsTimeToday	Total handle time in seconds for completed AutoOut (predictive) calls handled by agents in the skill group during the current day. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
AutoOutCallsHeldToday	Total number of completed AutoOut (predictive) calls that agents in the skill group have placed on hold at least once during the current day.
AutoOutCallsHeldTimeToday	Total number of seconds AutoOut (predictive) calls were placed on hold by agents in the skill group during the current day.
PreviewCallsToday	Total number of outbound Preview calls completed by agents in the skill group during the current day.
PreviewCallsTalkTimeToday	Total talk time in seconds for completed outbound Preview calls handled by agents in the skill group during the current day. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
PreviewCallsTimeToday	Total handle time in seconds for completed outbound Preview calls handled by agents in the skill group during the current day. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
PreviewCallsHeldToday	Total number of completed outbound Preview calls that agents in the skill group have placed on hold at least once during the current day.
PreviewCallsHeldTimeToday	Total number of seconds outbound Preview calls were placed on hold by agents in the skill group during the current day.
ReservationCallsToday	Total number of agent reservation calls completed by agents in the skill group during the current day.
ReservationCallsTalk TimeToday	Total talk time in seconds for completed agent reservation calls handled by agents in the skill group during the current day. The value includes the time spent from the call being initiated to the time the agent begins after call work time for the call. The time includes hold time associated with the call.
ReservationCallsTimeToday	Total handle time in seconds for completed agent reservation calls handled by agents in the skill group during the current day. The value includes the time spent from the call being initiated to the time the agent completes after call work time for the call. The time includes hold time associated with the call.
ReservationCallsHeldToday	Total number of completed agent reservation calls that agents in the skill group have placed on hold at least once during the current day.

Table 7-3 SkillGroup Statistics (continued)

Statistic	Description
ReservationCallsHeldTime Today	Total number of seconds agent reservation calls were placed on hold by agents in the skill group during the current day.
BargeInCallsToday	Total number of supervisor call barge-ins completed in the skill group during the current day.
InterceptCallsToday	Total number of supervisor call intercepts completed in the skill group during the current day.
MonitorCallsToday	Total number of supervisor call monitors completed in the skill group during the current day.
WhisperCallsToday	Total number of supervisor call whispers completed in the skill group during the current day.
EmergencyCallsToday	Total number of emergency calls completed in the skill group during the current day.
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.

Syntax

```
GetPropertyAttribute(propertyname, attribute requested)
```

Example

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

` Then get the desired statistics
Dim agentsLoggedOn As Integer
Dim agentsAvail As Integer
agentsLoggedOn = args.GetValue("AgentsLoggedOn")
agentsAvail = args.GetValue("AgentsAvail")
```

GetValue

The GetValue method retrieves a property (or an array of properties) from the SkillGroup object based on the property's name key. GetValue takes either a single key name or an array of key names as its required argument, and returns the value associated with that key.

Syntax

```
GetValue(key)
```

Example

```
Dim WithEvents sg as ctios. SkillGroupObject
Dim SkillGroupNumber As String
SkillGroupNumber = sg.GetValue(eSkillGroupNumber)
' Also valid
SkillGroupNumber = sg.GetValue("skillgroupnumber")
```

OnEvent

The object receives skill group statistic update events. The Agent object enables these when the client initiates a EnableSkillGroupStatistics or skill group statistics may have automatically been enabled on the server on behalf of all clients.

Syntax

```
OnEvent( event_id, arrayParameters )
```

SkillGroupEvents Interface

The SkillGroup object fires events on the ISkillGroupEvents interface. The following events are published to subscribers of the ISkillGroupEvents interface.

OnSkillGroupStatisticsUpdated

The OnSkillGroupStatisticsUpdated event is generated when skill group statistics are reported. The OnSkillGroupStatisticsUpdated message may contain the following fields as well as the fields listed in Table 7-3.

Field	Description
PeripheralID	The ICM PeripheralID of the ACD on which the agent resides.
SkillGroupNumber	The number of the agent skill group as known to the peripheral. May contain the special value NULL_SKILL_GROUP when not available.
SkillGroupID	The ICM SkillGroupID of the skill group. May contain the special value NULL_SKILL_GROUP when not available.

CHAPTER 8

Helper Classes

The CTI OS CILincludes the following helper classes.

- The Arg class is the basic data type used in CIL for any parameter included in methods or events. Objects of this type allow CIL to be fully extensible and reusable.
- Arguments class is a subclass of Arg and its main responsibility is to maintain an array of objects of type Arg. Each object contained in this class is always associated with a key. Elements in an Arguments array can be dynamically added, removed, accessed using a key, or sequentially by a one based index value.
- CilRefArg class is a subclass of Arg and its main responsibility is to store a reference of a CCtiOsObject object. For instance, it can hold reference to a CAgent, CCall, CskillGroup, CWaitObject or CCtiOsSession.
- The CCtiosException class it is normally used within the Arguments class. It provides access to additional information when errors are generated, such as what parameter is in error, memory allocation failed among other.
- CCtiOsObject is the principal base class for the CTI OS Client Library. It serves as the root for all classes such as CAgent, CCall , CSkillGroup, CSession and CWaitObject.

Arg Class

The Arg is a generic class used in parameters or return values in CIL methods. Information sent by CTI OS server to the CIL in an event is packed in an Arguments object where each element of the array is an object of type Arg. An Arg object's absolute data type is determined by the type of data it stores. The basic types an object can store are identified by the enumerated constants in Table 8-1.

Table 8-1 enumArgTypes

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	STL character string
ARG_ARGARRAY	Variable length array of Arg

Table 8-2 lists the available Arg class methods.

Table 8-2 Arg Class Methods

Method	Description
CreateInstance	Creates an Arg object
SetValue	Sets the data in the Arg object
GetValueInt GetValueUInt GetValueUInt GetValueUShort GetValueShort GetValueBool GetValueString	Returns the value stored in the argument
Clone	Creates an exact copy of the Arg object
GetType	Returns the type of the data stored in the argument (one of the values in Table 8-1).
GetClassID	Returns eArg

CreateInstance

The CreateInstance method creates an object of type Arg class. It is important to release the object when is not longer in use in the program.

Syntax

C++

```
static Arg& CreateInstance()
static bool CreateInstance(Arg ** ppArg)
```

COM (standard COM API)

```
HRESULT CoCreateInstance (…, REFIID riid, ...)
```

Parameters

ppArg

(output) Pointer to an Arg class object to receive the address of the new object instance.

riid

(input) A reference to the identifier of the interface IArg.

Return Values

C++

The first format, if successful, will return a reference to a new *Arg* object. Otherwise, it will raise a CCtiosException with iCode set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED.

The second format returns True if successful, False otherwise.

COM

Standard COM API error code

Examples

C++

```
try
         {
                //First method to create objects
    Arg & arParam = Arg::CreateInstance();

                //Second method to create objects
                bool    bAllocOk = false;
               Arg * pParam = NULL;
bAllocOk = Arg::CreateInstance(&pParam);
                //Do not forget to release the two objects when done
with them
}
catch (CCtiOsException & e)
{
}
```

COM C++

```
           IArg * pParam = NULL;

hr = CoCreateInstance(CLSID_Arg, NULL, CLSCTX_ALL, IID_IArg, (LPVOID
*) & pParam);
```

VB

```
Dim arParam as Arg
Set arParam = new Arg
```

SetValue

The SetValue method sets the value in the object.

Syntax

C++

```
bool SetValue( int iValue );
bool SetValue( unsigned int uiValue );
bool SetValue( unsigned short usValue );
bool SetValue( short sValue );
bool SetValue( bool bValue );
bool SetValue( char * pcValue );
bool SetValue( string& strValue);
bool SetValue( const string& cstrValue);
bool SetValue( Arg & arValue);
```

COM

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

Input Parameters

Value

Integer value to be set

uiValue

Unsigned integer value to be set

usValue

Unsigned short integer value to be set.

sValue

Signed short integer value to be set.

bValue

Boolean value to be set.

pcValue

Points to the buffer containing the null terminated character string to be set.

strValue

Reference to an STL string object containing the character string to be set.

cstrValue

Constant reference to an STL string object containing the character string to be set.

arValue

Reference to an Arg object to be set.

pVariant

Points to a variant parameter that contains the data to be set on the IArg object. The following are the permitted types for this variant.

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

Return Values

C++

If the method was able to set the value it returns true, otherwise it returns false.

COM

If the method was able to set the value it returns VARIANT_TRUE. Otherwise, it returns VARIANT_FALSE.

Examples

C++

```
            bool    bAllocOk = false;
           Arg * pParam = NULL;

            bAllocOk = Arg::CreateInstance(&pParam);

if(bAllocOk)
{
    pParam->SetValue(10);
   //Do useful work with the object
    pParam->Release();
}
```

COM C++

```
            VARIANT    vParam;
            VARIANT_BOOL  bRet = VARIANT_FALSE;

            VariantInit(&vParam);

hr = CoCreateInstance(CLSID_Arg,
                                                NULL,
                                               CLSCTX_ALL,
                                                IID_IArg, (LPVOID *)
& pParam);

vParam.vt  =  VT_INT;
     vParam.intVal = 10;

           pParam->SetValue(&vParam, &bRet);

            //Do useful work with the object

            pParam->Release();
```

VB

```
Dim arParam As Arg
 Dim bRet  As Boolean
Set arParam = new Arg

bRet  = arParam.SetValue(10)

             'Do useful work with the object

            Set  arParam = Nothing
```

GetValueType

The GetValueType method returns the value stored in the object. To extract a specific type of data you invoke the method designated for it.

Syntax

C++

```
int GetValueInt();
unsigned int GetValueUInt();
unsigned shortGetValueUShort();
short GetValueShort();
string& GetValueString();
bool GetValueBool();

bool GetValueInt(int * piValue);
bool GetValueUInt(unsigned int * puiValue);
bool GetValueUShort(unsigned short * pusValue);
bool GetValueShort( short * psVallue);
bool GetValueBool( bool * pbValue);
bool GetValueString(string* pstrValue);
```

COM

```
HRESULT GetValue([out, retval] VARIANT * pVariantOut);
```

Output Parameters

piValue

Points to an integer variable that will receive the value.

puiValue

Points to an unsigned integer variable that will receive the value.

pusValue

Points to an unsigned short integer variable that will receive the value.

psValue

Points to a signed short integer variable that will receive the value.

pbValue

Points to a Boolean variable that will receive the value.

pstrValue

Points to a STL string object that will receive the value.

pVarianOut

Points to a variant variable that will receive the value. The following are the possible types to be returned.

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++

First implementation, if successful, will return the value in the object; otherwise, it will raise a CCtiosException with iCode set to E_CTIOS_INVALID_ARGUMENT.

Second implementation. If the method was able to get the value it returns true. Otherwise, false.

COM

If the method was able to set the value pVarianOut->vt to any of the types listed in the table of pVarianOut, it returns the value in the object. Otherwise it returns VT_EMPTY.

Examples

C++

```
  try
         {
                //First method to create objects
    int       iPeripheral
                string  strInstrument;

    //Getting peripheral using first implementation
    Arg & arPeripheral = m_Agent.GetValue(_T("PeripheralID"));
                iPeripheral = arPeripheral.GetValueInt();

               //Getting instrument using second implementation
               Arg & arInstrument =
m_Agent.GetValue(_T("Instrument"));

               if(arInstrument.GetValueString(&strInstrument))
               {
                    //Do something interesting with peripheral and
instrument
               }

   arPeripheral.Release();
               arInstrument.Release();

}
catch (CCtiOsException & e)
{

}
```

COM C++

```
IArg *          pInstrument = NULL
           VARIANT    vPeripheralID;

          //Get peripheral from COM Agent object and assign it to
pInstrument

          //Retrieveing data from argument
         VariantInit(&vPeripheralID);

         pInstrument->GetValue((&vPeripheralID);
         if(vPeripheral.vt != VT_EMPTY)
         {
              m_iPeripheralID =  vPeripheral.intVal;
         }

        pInstrument->Release()
```

VB

```
 Dim arInstrument As Arg
 Dim iPeripheral As Integer

 'Initializes IArg

 Set arInstrument = m_Agent.GetValue("Instrument")

 iPeripheralID = arInstrument.GetValueInt

  Set arInstrument = Nothing
```

Clone

The Clone method causes the object allocates a new object in memory and to copy its value and type to the new instance.

Syntax

C++

```
Arg & Clone()
```

COM

```
HRESULT Clone([out,retval] LPDISPATCH * pClonedArg);
```

Output Parameters

pClonedArg

Pointer to an IArg interface instance that receive the address of the new object instance.

Return Values

C++

If successful, will return a reference to a new Arg object; otherwise, it will raise a CCtiosException with iCode set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED.

COM

If successful, will return a pointer to an IArg interface; otherwise, it will raise a COM Exception with error code set to E_CTIOS_ARGUMENT_ALLOCATION_FAILED.

Examples

C++

```
  try
         {
                //First method to create objects
    Arg & arDNCopied = arDiledNumber.Clone();

               cout <<  “Dialing “ << arDNCopied.GetValueString();

               arDNCopied.Release();
               arDiledNumber.Release()
}
catch (CCtiOsException & e)
{
    arDiledNumber.Release();
}
```

COM C++

```
           IArg * pParam = NULL;

           hr = arDiledNumber.Clone( (LPDISPATCH*) & pParam);
           if(CIL_FAILED(hr))
           {
               MessageBox(NULL,"Failed cloning Argument
","App",MB_STOP);
           }
          arDiledNumber->Release();
```

VB

```
      On Error Goto HandleError
Dim arParam as Arg

Set arParam = arDialedNumber.Clone

           Set arParam = Nothing
           Set arDialedNumber = Nothing
     Exit Sub
     HandleError:
           MsgBox  "Failed cloning Argument"
ArDialedNumber = Nothing
```

GetType

The GetType method returns the type of the data stored by the argument. See the table listed under the pVarianOut parameter for the GetValueType method for a list of possible types.

Syntax

C++

```
enumArgTypes GetType()
```

COM

```
HRESULT GetType([out,retval] int  * nArgType );
```

Output Parameters

nArgType

Pointer to an integer that receive the enumerated constant that identifies data type stored in IArg.

Return Values

C++

Returns the enumerated value that identifies the data type stored in the Arg

COM

If the method invocation succeeded it returns S_OK. Otherwise, it returns CIL_FAILED.

Examples

C++

```
           enumArgTypes enType;
for (int nI = 1; nI < arEventParam.NumElements(); nI++)
{
     Arg & arParam =  arEventParam.GetelEment(nI);
    enType = arParam.GetType();

                if(enType == ARG_STRING)
                {
                     //Doing some work
                }
    arParam.Release();
}
```

COM C++

```
            HRESULT  hr = S_OK;
          int nType;
            int nNumElem = 0;
            arEventParam.NumElements(&nNumElem);

for (int nI = 1; nI < nNumElem; nI++)
{
     CComVariant vParam =  arEventParam.GetelEment(nI);
      IArg * pParam = (IArg*)  vParam.pdispVal

      hr = pParam->GetType(&nType);

                  if(nType == ARG_STRING)
                 {
                     //Doing some work
                 }
      pParam->Release();
}
```

VB

```
            Dim nType As Integer;
            Dim nI As Integer
            Dim arParam As Arg

For nI = 1 To nI <  arEventParam.NumElements()

    Set arParam =  arEventParam.GetelEment(nI);
    nType = arParam.GetType;

                If nType = ARG_STRING Then

                     //Doing some work
                End If

    Set arParam = Nothing;
```

GetClassID

The GetClassID method returns the object's class ID.

Syntax

C++

```
int GetClassID()
```

COM

```
HRESULT GetClassID ([out,retval] int * pClassID);
```

Output Parameters

pClassID

Pointer to an integer that receives the enumerated constant that identifies the object class type.

Return Values

C++ and COM

Retuns eArg if the pointer is referencing an Arg object.

Examples

C++

```
            int nClassID;

for (int nI = 1; nI < arEventParam.NumElements(); nI++)
{
     Arg & arParam =  arEventParam.GetelEment(nI);
    enType = arParam.GetClassID();

                  switch(nClassID)
```

```
                    {
                          case  eArg:
                                   //Doing some work
                          break;
                          case eCILRefArg:
                          break;
                    }
        arParam.Release();
    }
```

COM C++

```
                HRESULT  hr = S_OK;
              int nClassID;
                int nNumElem = 0;

                arEventParam.NumElements(&nNumElem);

    for (int nI = 1; nI < nNumElem; nI++)
    {
         CComVariant vParam =  arEventParam.GetelEment(nI);
          IArg * pParam = (IArg*)  vParam.pdispVal

          hr = pParam->GetClassID(&nClassID);

                     switch(nClassID)
                    {
                          case  eArg:
                                   //Doing some work
                          break;
                          case eCILRefArg:
                          break;
                    }
          pParam->Release();
    }
```

VB

```
    Dim nClassID As Integer;
                Dim nI As Integer
                Dim arParam As Arg

    For nI = 1 To nI <  arEventParam.NumElements()

        Set arParam =  arEventParam.GetelEment(nI);
        nClassID = arParam.GetClassID;
```

```
                If nClassID = eArg Then

                     //Doing some work
                Else
                    If nClassID = eCILRefArg Then

                End If

    Set arParam = Nothing;

Next nI
```

Arguments Class

The Arguments class provides key/ value support. The Arguments class contains a list of values that can be associated with a key. To add an item, use the AddItem method and pass a key value, which must be a string or an integer, and a value, which can be anything. To retrieve the item, use GetValue with a key value. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Items stored in an Arguments array use binary trees and other techniques to provide fast access to any item. They can support very large array of values. Arguments also supports access by index value or by search key. The index value is useful for retrieving items sequentially and may not be as fast as retrieval by key.

The Arguments class methods includes all the Arg class methods *plus* the methods that are listed in Table 8-3.

Table 8-3 Arguments Class Methods

Method	Description
AddItem	Adds an item to an Arguments array.
Clear	Deletes all elements from an Arguments array.
Clone	Creates an Arguments array.
CreateInstance	Creates an Arguments array.

Table 8-3 Arguments Class Methods

DumpArgs	Returns Arguments object as a string
GetClassID	Returns Class ID
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.
Initialize	Removes all items from an Arguments array.
IsValid	Tests if a key is present in the current Arguments array.
NumElements	Returns the number of arguments in the current Arguments array,.
RemoveItem	Removes an item from an Arguments array.
SetElement	Sets the value of an index.
SetValue	Sets the value of a key.

AddItem

The AddItem method expects a key/value pair. The key value may be a string or an integer. The value may be a string, an integer, or an object reference. If there is an entry with the same key, it will be replaced with this entry. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Syntax

```
AddItem( key, value )
```

Parameters

key

Key value for the item to be added.

value

Value of the item to be added.

Returns

Returns True if the entry replaced an entry in Arguments with the same key value.

Clear

The Clear method deletes all the elements from Arguments object.

Syntax

```
void Clear( )
```

Parameters

None.

Clone

The Clone method creates a copy of the Arguments structure.

Syntax

```
Arg & Clone()
```

Parameters

None.

Returns

A reference to the Arguments structure copy.

CreateInstance

The CreateInstance method creates an object of type Arguments class. This object is released by calling the Release method.

Syntax

```
static Arguments & CreateInstance()
static bool CreateInstance(Arguments ** ppArguments)
```

DumpArgs

The DumpArgs method dumps all of the args to a string. It is used for debugging purposes.

Syntax

```
string DumpArgs()
```

Parameters

None.

GetClassID

The GetClassID method returns a ClassID as integer.

Syntax

```
int GetClassID()
```

Parameters

None.

Returns

Integer value for the ClassID.

GetElement

The GetElement method is identical to GetValue, except that it uses an index value instead of a key. The index value is not related to the order in which items are added or removed. The order of items in Arguments is never guaranteed. This method is useful for sequentially iterating over all items in Arguments. Index is zero (0) based. Index should never be less than zero or greater than NumElements minus 1.

Syntax

```
GetElement( index )
```

GetValue

The GetValue method returns the value stored under a key. This method will return a blank string if the key is invalid. The key can be tested using IsValid. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Syntax

```
GetValue( key )
```

Initialize

The Initialize method removes all items from an Arguments class. This method *does not* need to be called when an instance is created. It will automatically be in initialize mode.

Syntax

```
Initialize()
```

Parameters

None.

IsValid

The IsValid method returns True if key exists in the current Arguments array, otherwise it returns False.

Syntax

```
IsValid( key )
```

NumElements

The NumElements method returns number of elements stored in the current arguments array.

Syntax

```
NumElements()
```

Parameters

None.

RemoveItem

The RemoveItem method expects a key. It locate and remove the item by key in Arguments. A True return value means the entry was located. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Syntax

```
RemoveItem( key )
```

SetElement

The SetElement method is identical to SetValue, except that it uses an index value instead of a key. The index value is not related to the order in which items are added or removed. The order of items in Arguments is never guaranteed. This method is useful for sequentially iterating over all items in Arguments. Index is zero (0) based. Index should never be less than zero or greater than NumElements minus 1.

Syntax

```
SetElement( index, value )
```

SetValue

The SetValue method sets a value for a key. A True return value means the entry was located. Keys are not case sensitive. Leading and trailing spaces are always removed from the key.

Syntax

```
SetValue( key, value )
```

CILRefArg Class

Note The CILRefArg class is only available for C++.

The CILRefArg class is a subclass of the Arg class. Its main responsibility is to store a reference of a CCtiOsObject object. This class allows object references to be included in argument arrays. The object types that can be used are any of the following: CAgent, CCall, CskillGroup, CWaitObject or CctiOsSession.

In addition to the methods inherited from the Arg class, the CILRefArg class contains the methods listed in Table 8-4.

Table 8-4 CILRefArg Class Methods

Method	Description
SetValue	Encapsulates the pointer to CTI OS object into the CILRefArg object.
GetValue	Returns the encapsulated pointer in the object.
GetClassID	Returns eCILRefArg
GetType	Returns ARG_NOTSET

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.

Examples

```
CILRefArg * pCILRefArg = NULL;

 bool bRet = CILRefArg::CreateInstance(&pCILRefArg);

 if ((bRet) && (pCILRefArg))
 {
      pCILRefArg->SetValue((CCtiOsObject*)(*itObject).second);
     rArrayRef.AddItem((string)(*itObject).first, *pCILRefArg);
    pCILRefArg->Release();
   }
```

GetValue

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

Syntax

```
CCtiOsObject * GetValue();
```

Parameters

None.

Return Values

If successful, it returns the value in the object. Otherwise, it will raise a CCtiosException with iCode set to E_CTIOS_INVALID_ARGUMENT.

Examples

```
CILRefArg & rCILRefArg =
                              (CILRefArg &) GetValue(“CurrentAgent”);

CAgent * pctiAgent = (CAgent *) rCILRefArg.GetValue();

            if ( pctiAgent )
{
     pctiAgent->Logout();
}
```

GetClassID

The GetClassID method returns the object’s class id.

Syntax

```
int GetClassID()
```

Parameters

None.

Return Values

Returns eCILRefArg if the pointer is referencing a CILRefArg object.

Examples

```
int nClassID;

for (int nI = 1; nI < arEventParam.NumElements(); nI++)
{
     Arg & arParam =  arEventParam.GetelEment(nI);
    enType = arParam.GetClassID();

                 switch(nClassID)
                {
                      case  eArg:
                      break;
                      case eCILRefArg:
                            //Doing some work
                      break;
                }
    arParam.Release();
}
```

GetType

The CilRefArg class GetType method returns the ARG_NOTSET. It is defined and enumerated type to represent CTI OS Objects references. It has the same syntax as the Arg class GetType method.

CCtiOsException Class

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

The CCtiosException class it is normally used within the Arguments class. It provides access to additional information when errors are generated, such as what parameter is in error, memory allocation failed, and so on.

Table 8-5 lists the available CCtiOsException class methods.

Table 8-5 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

The CCtiosException constructor initializes an object of type CCtiosException class.

Syntax

```
CCtiosException(const char *pMsg, int iCode, int iStatus);
CCtiosException(const string& rstrMsg, int iCode, int iStatus);
```

Input Parameters

pMsg

Pointer to string that holds a description of an error.

iCode

Number that identifies an error.

iStatus

Status of an error.

rstrMsg

An STL string that holds a description of an error.

Return Values

None.

Example

```
String estr;

estr.Format(_T("Arguments::GetValue, Arg with key %d not found."),
nKey);
throw CCtiosException(estr, 0, 0);
```

GetCode

The GetCode method returns the error code that generated the exception.

Syntax

```
int GetCode();
```

Parameters

None.

Return Values

Returns an integer error code that generated the exception.

Example

```
Int nRetCode = 0;
String estr;
estr.Format(_T("Arguments::GetValue, Arg with key %d not found."),
nKey);
CctiosException HighException(estr, 1, 0);

nRetCode = HighException.GetCode( );
```

GetStatus

The GetStatus method returns the error status that generated the exception.

Syntax

```
int GetStatus ();
```

Parameters

None.

Return Values

Returns an integer error status that generated the exception.

Example

```
Int nRetStatus = 0;
String estr;
estr.Format(_T("Arguments::GetValue, Arg with key %d not found."),
nKey);
CctiosException HighException(estr, 1, 0);

nRetStatus = HighException. GetStatus ( );
```

GetString

The GetString method returns a text string containing the description of the exception.

Syntax

```
const char* GetString();
```

Parameters

None.

Return Values

Returns a text string containing the description of the exception.

Example

```
String estr;
estr.Format(_T("Arguments::GetValue, Arg with key %d not found."),
nKey);
CctiosException HighException(estr, 1, 0);

cout << HighException. GetString ( ) << endl;
```

What

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

Syntax

```
const char* What();
```

Parameters

None.

Return Values

Returns a text string containing the description of the exception, the code of an error and the status.

Example

```
String estr;
estr.Format(_T("Arguments::GetValue, Arg with key %d not found."),
nKey);
CctiosException HighException(estr, 1, 0);

Cout << HighException. What ( ) << endl;
```

CCtiOsObject Class

Note The CCtiOsObject class is only available for C++.

CCtiOsObject is the principal base class for the CTI OS Client Library. It serves as the root for all classes such as CAgent, CCall , CSkillGroup, CSession and CWaitObject. CCtiOsObject provides basic services, including

- Object lifetime control using a reference counting mechanism
- Run-time class information
- Dynamic management of properties

Table 8-6lists the available CCtiOsObject class methods.

Table 8-6 CCtiOsObject Class Methods

Method	Description
DumpProperties	Dumps the properties and their values in a log file.
GetPropertyName	Returns a property name in a string format.
GetNumProperties	Returns the number of properties of an object.
GetValue GetValueInt GetValueString	Returns the value stored in the argument.
GetElement	Returns a value of an element.
GetPropertyAttribute	Returns information about a property.
SetValue	Sets the data of a specific property.
IsValid	Checks to see if the property of an object is valid.

DumpProperties

The DumpProperties method dumps the properties and their values in a log file.

Syntax

```
string  DumpProperties();
```

Parameters

None.

Return Values

Returns a text string containing the properties and their values.

Example

```
// Get a valid CCtiOsObject pointer
cout << pCCtiOsObject-> DumpProperties().c_str() << endl;
```

GetPropertyName

The GetPropertyName method returns a property name in a string format.

C++

```
string  GetPropertyName(int nProperty);
```

Input Parameters

nProperty

Property or index number

Return Values

Returns a property name in a string format.

Example

```
string strPropertyName;
int Index = 1;   // Get a property of index name 1
// Get a valid CCtiOsObject pointer
strPropertyName = pCCtiOsObject->GetPropertyName(Index);
```

GetNumProperties

The GetNumProperties method returns the number of properties of an object.

Syntax

```
int  GetNumProperties();
```

Parameters

None.

Return Values

Returns the number of properties of an object.

Example

```
int nNumbProp = 0;
// Get a valid CCtiOsObject pointer
nNumbProp = pCCtiOsObject-> GetNumProperties ();
```

GetValueType

The GetValueType method returns data of a specific property.

Syntax

```
Arg &GetValue( string& sKey );
Arg &GetValue( char * pKey );
intGetValueInt( string& sKey);
intGetValueInt( char * pKey );
string   GetValueString( string& sKey  );
string   GetValueString( char * pKey );
```

Input Parameters

sKey

An STL character string containing the property name.

pKey

A pointer to a null terminated string containing the property name.

Return Values

Returns the number of properties of an object.

Example

```
int       iPeripheral
            string  strInstrument;

//Getting peripheral using first implementation
Arg & arPeripheral = m_Agent.GetValue(_T("PeripheralID"));
            iPeripheral = arPeripheral.GetValueInt();

            //Getting instrument using second implementation
 strInstrument = m_Agent.GetValueString(_T("Instrument"));

//Getting peripheral using third implementation
 iPeripheral = m_Agent. GetValueInt (_T("PeripheralID"));

arPeripheral.Release();
```

GetElement

The GetElement method is intended to access information in a property that was defined as an array. It returns the information stored in position *nElement*.

Syntax

```
Arg&    GetElement( string& sKey , int nElement);
Arg&    GetElement( int nKey, int nElement);
Arg&    GetElement( char * pKey, int nElement);
```

Input Parameters

sKey

An STL string containing the property name

pKey

A pointer to a null terminated string containing the property name nKey

An enumerated integer that identifies the property.

nElement

A one-based index that corresponds to the n-th element in the array.

Return Values

Returns the data of an element.

GetPropertyAttribute

The GetPropertyAttribute method returns information about a property. See Chapter 1, "Introduction" for more information.

Syntax

```
Arg&    GetPropertyAttribute( string& strPropName,
enumCTIOS_Attribute nAttribute);
Arg&    GetPropertyAttribute( int nPropName,
enumCTIOS_Attribute nAttribute);
Arg&    GetPropertyAttribute( char * pPropName,
enumCTIOS_Attribute nAttribute);
```

Input Parameters

strPropName

An STL string containing the property name.

pPropName

A pointer to a null terminated string containing the property name.

nAttribute

An enumerated integer that identifies the property.

Return Values

Returns information about a property of an object.

SetValue

The SetValue method sets the value of a property.

Syntax

```
bool SetValue( string& sKey, string& sValue );
bool SetValue( string& sKeyValuePair  );
bool SetValue( string& sKey, int nValue );

bool SetValue( const char * pKey, const char * pValue );
bool SetValue( const char * pKeyValuePair  );
bool SetValue( const char * pKey, int nValue );
```

Input Parameters

sKey

An STL string containing the property name

pKey

A pointer to a null terminated string containing the property name

sKeyValuePair

An STL string containing the property name and the value to assign. The format of this parameter is: “Key=Value”

pValue

A pointer to a null terminated string containing the value to set in the property;

pKeyValuePair

A pointer to a null terminated string containing the property name and the value to assign. The format of this parameter is: “Key=Value” .

nValue

A numeric value to set in the property.

Return Values

Returns true if the method succeeds, false otherwise.

Example

```
//Setting peripheral
bool bRet =  m_Agent. SetValue (_T("PeripheralID"), 5000);

//Setting instrument
 bool bRet =  m_Agent. SetValue (_T("Instrument"), _T("2229"));

//Setting Agent ID
 bool bRet =  m_Agent. SetValue (_T("AgentID=23840"));
```

IsValid

The IsValid method checks to see if the property of an object is valid.

Syntax

```
bool    IsValid( char * pKey );
bool    IsValid( string& sKey );
```

Input Parameters

sKey

An STL string containing the property name

pKey

A pointer to a null terminated string containing the property name

Return Values

Returns true if the method succeeds, false otherwise.

Example

```
//setting peripheral
if (  m_Agent. IsValid (_T("PeripheralID")))
{
// Do something if it is valid
}
```

APPENDIX A

CTI OS CIL Messages

This appendix lists the messages in the CTI OS CIL message set. This appendix is divided into two sections.

- CIL message equivalents to CTI Server messages
- CIL CTI OS-specific messages

CIL CTI Server Message Equivalents

The following CTI OS message requests, confirmations, and events correspond to the CTI Server messages defined in the *Cisco ICM Software CTI Server Message Reference Guide (Protocol Version 8)*.

Call Requests

eSetCallDataRequest

eReleaseCallRequest

eAlternateCallRequest

eAnswerCallRequest

eClearCallRequest

eClearConnectionRequest

eConferenceCallRequest

eConsultationCallRequest

eDeflectCallRequest

eHoldCallRequest

eReconnectCallRequest

eRetrieveCallRequest

eTransferCallRequest

eSnapshotCallRequest

eSendDTMFRequest

Agent Requests

eQueryAgentStateRequest

eSetAgentStateRequest

eMakeCallRequest

eMakePredictiveCallRequest

eSnapshotDeviceRequest

eUserMessageRequest

eQueryAgentStatisticsRequest

Skill Group Requests

eQuerySkillGroupStatisticsRequest

Supervisor Requests

eSessionMonitorStartRequest

eSessionMonitorStopRequest

eMonitorAgentTeamStartRequest

eMonitorAgentTeamStopRequest

eSupervisorAssistRequest

eEmergencyCallRequest

eSuperviseCallRequest

eAgentTeamConfigRequest

eSetAppDataRequest

eAgentDeskSettingsRequest

eListAgentTeamRequest

eBadCallRequest

eSetDeviceAttributesRequest

eStartRecordingRequest

eStopRecordingRequest

Generic CTI Server Confirmation Events

eControlFailureConf

Call Confirmation Events

eSetCallDataConf

eReleaseCallConf

eAlternateCallConf

eAnswerCallConf

eClearCallConf

eClearConnectionConf

eConferenceCallConf

eConsultationCallConf

eDeflectCallConf

eHoldCallConf

eReconnectCallConf

eRetrieveCallConf

eTransferCallConf

eSnapshotCallConf

eSendDTMFConf

Agent Confirmation Events

eQueryAgentStateConf

eSetAgentStateConf

eSnapshotDeviceConf

eUserMessageConf

eQueryAgentStatisticsConf

eMakeCallConf

eMakePredictiveCallConf

Supervisor Confirmation Events

eSessionMonitorStartConf

eSessionMonitorStopConf

eMonitorAgentTeamStartConf

eMonitorAgentTeamStopConf

eSupervisorAssistConf

eEmergencyCallConf

eSuperviseCallConf

eAgentTeamConfigConf

eSetAppDataConf

eAgentDeskSettingsConf

eListAgentTeamConf

eBadCallConf

eSetDeviceAttributesConf

eCallStartRecordingConf

eCallStopRecordingConf

Error and Failure Events

eFailureConf

eFailureEvent

Call Events

eCallDeliveredEvent

eCallEstablishedEvent

eCallHeldEvent

eCallRetrievedEvent

eCallClearedEvent

eCallConnectionClearedEvent

eCallOriginatedEvent

eCallFailedEvent

eCallConferencedEvent

eCallTransferredEvent

eCallDivertedEvent

eCallServiceInitiatedEvent

eCallQueuedEvent

eCallTranslationRouteEvent

eCallBeginEvent

eCallEndEvent

eCallDataUpdateEvent

eCallReachedNetworkEvent

eCallDequeuedEvent

eAgentPrecallEvent

eAgentPrecallAbortEvent

Call Recording Events

eRTPStartedEvent

eRTPStoppedEvent

Agent Events

eAgentStateEvent

eUserMessageEvent

eOnNewAgentStatisticsEvent

Skill Group Events

eOnNewSkillGroupStatisticsEvent

Supervisor Events

eEmergencyCallEvent

eAgentTeamConfigEvent

CIL CTI OS-specific messages

The following messages, requests, and events are specific to CTI OS. Chapters 4 through 7 include descriptions and field definitions for the most commonly occurring messages.

Session Requests

eOpenConnection

eCloseConnection

eSetSessionModeRequest

eGlobalSettingsDownloadRequest

Single Step Transfer/Conference Requests

eSingleStepTransferRequest

eSingleStepConferenceRequest

Supervisor Requests

eGetAllSupervisedAgentsRequest

eSetMonitoredAgentStateRequest

eMonitorAllAgentTeamsRequest

Statistics Requests

eEnableAgentStatisticsRequest

eEnableSkillGroupStatisticsRequest

eDisableAgentStatisticsRequest

eDisableSkillGroupStatisticsRequest

eSubscribeForStatisticsRequest

eUnsubscribeForStatisticsRequest

Timer Service Requests

eStartTimerRequest

eStopTimerRequest

Team Maintenance Requests

eAssignToTeamRequest

eRemoveFromTeamRequest

CTI OS Specific Confirmation Events

eSingleStepTransferConf

eSingleStepConferenceConf

CTI OS Specific Events

eSendChatMessageReq

eSendChatMessageConf

eChatMessageEvent

Session Events

eOnConnection

eOnConnectionClosed

eOnConnectionRejected

eOnConnectionFailure

eOnCILError

eCTIOSFailureEvent

eOnHeartbeat

eOnMissingHeartbeat

eOnMonitorModeEstablished

eOnCurrentCallChanged

eSetAgentModeEvent

eOnCurrentAgentReset

eGlobalSettingsDownloadConf

ePreLogoutEvent

ePostLogoutEvent

eLogoutFailedEvent

Button Enablement Events

eButtonEnablementMaskChange

Supervisor Events

eSupervisorNotifyAgent

eMonitoredAgentStateChange

eMonitoredCallEvent

eNewAgentTeamMember

eSupervisorButtonChange

eStartMonitoringAgentEvent

eDropMonitoredAgentEvent

Filter Events

eAddFilterEvent

eRemoveFilterEvent

Name Lookup Service Requests

eGetAgentNameRequest

Name Lookup Service Event

eAgentInfoEvent

eMonitoredAgentInfoEvent

Timeout Events

eAgentPrecallEventTimeout

Supervised Calls

All the supervised call events are sent to an agent mode application only when the agent logged in the session is a supervisor. Each of the event has its corresponding CTI Server event sent to the agent being supervised.

eMonitoredCallDeliveredEvent

eMonitoredCallEstablishedEvent

eMonitoredCallHeldEvent

eMonitoredCallRetrievedEvent

eMonitoredCallClearedEvent

eMonitoredCallConnectionClearedEvent

eMonitoredCallOriginatedEvent

eMonitoredCallFailedEvent

eMonitoredCallConferencedEvent

eMonitoredCallTransferredEvent

eMonitoredCallDivertedEvent

eMonitoredCallServiceInitiatedEvent

eMonitoredCallQueuedEvent

eMonitoredCallTranslationRouteEvent

eMonitoredCallBeginEvent

eMonitoredCallEndEvent

eMonitoredCallDataUpdateEvent

eMonitoredCallReachedNetworkEvent

eMonitoredCallDequeuedEvent

eMonitoredAgentPrecallEvent

eMonitoredAgentPrecallAbortEvent

Supervised Agent Events

All the supervised agent events are sent to an agent mode application only when the agent logged in the session is a supervisor. Each of the event has its corresponding CTI Server event sent to the agent being supervised.

eMonitoredAgentStateEvent

Team Maintenance Events

Contains up-to-date information about a team composition.

eTeamSkillGroupDataUpdateEvent,

APPENDIX B

CTI OS Keywords

Table B-1 lists all the valid keywords that can be used in CTI OS for the following purposes:

- To refer to a property in a CTI OS Object
- To refer to a parameter on a message request
- To create expressions used in a filter

The table shows the three possible forms of a CTI OS keyword.

- The first column is the keyword in text string format. Any CIL implementation can use this form directly provided the keyword is enclosed in double quotes.
- The second column is a constant symbols that maps directly to a text string representation. This form is recommended over the string text representation because if changes to the keywords occur in future versions of CIL, you need only rebuild the executable.
- The third column is an enumerated integer value. This form eliminates the need for an application programmer to deal with character strings when referring to properties or parameters.

Note Constant symbols and enumerated integers cannot be used to build expressions in a message filter.

Table B-1 CTI OS Keywords

Text String	Constant String	Enumerated Integer Value
InvokeID	INVOKEID	ekwInvokeID
VersionNumber	VERSIONNUMBER	ekwVersionNumber
IdleTimeout	IDLETIMEOUT	ekwIdleTimeout
PeripheralID	PERIPHERALID	ekwPeripheralID
ServiceMask	SERVICEMASK	ekwServiceMask
CallMsgMask	CALLMSGMASK	ekwCallMsgMask
AgentStateMask	AGENTSTATEMASK	ekwAgentStateMask
Reserved	RESERVED	ekwReserved
PGStatus	PGSTATUS	ekwPGStatus
PeripheralType	PERIPHERALTYPE	ekwPeripheralType
AgentState	AGENTSTATE	ekwAgentState
Status	STATUS	ekwStatus
NumCTIClients	NUMCTICLIENTS	ekwNumCTIClients
ConnectionCallID	CONNECTIONCALLID	ekwConnectionCallID
CallType	CALLTYPE	ekwCallType
ConnectionDeviceIDType	CONNECTIONDEVICEIDTYPE	ekwConnectionDeviceIDType
ServiceNumber	SERVICENUMBER	ekwServiceNumber
ServiceID	SERVICEID	ekwServiceID
AlertingDeviceType	ALERTINGDEVICETYPE	ekwAlertingDeviceType
CallingDeviceType	CALLINGDEVICETYPE	ekwCallingDeviceType
CalledDeviceType	CALLEDDEVICETYPE	ekwCalledDeviceType
LastRedirectDeviceType	LASTREDIRECTDEVICETYPE	ekwLastRedirectDeviceType
LocalConnectionState	LOCALCONNECTIONSTATE	ekwLocalConnectionState
EventCause	EVENTCAUSE	ekwEventCause
SkillGroupNumber	SKILLGROUPNUMBER	ekwSkillGroupNumber
SkillGroupID	SKILLGROUPID	ekwSkillGroupID
AnsweringDeviceType	ANSWERINGDEVICETYPE	ekwAnsweringDeviceType
HoldingDeviceType	HOLDINGDEVICETYPE	ekwHoldingDeviceType

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
RetrievingDeviceType	RETRIEVINGDEVICETYPE	ekwRetrievingDeviceType
ReleasingDeviceType	RELEASINGDEVICETYPE	ekwReleasingDeviceType
FailingDeviceType	FAILINGDEVICETYPE	ekwFailingDeviceType
PrimaryDeviceIDType	PRIMARYDEVICEIDTYPE	ekwPrimaryDeviceIDType
PrimaryCallID	PRIMARYCALLID	ekwPrimaryCallID
NumParties	NUMPARTIES	ekwNumParties
SecondaryDeviceIDType	SECONDARYDEVICEIDTYPE	ekwSecondaryDeviceIDType
SecondaryCallID	SECONDARYCALLID	ekwSecondaryCallID
ControllerDeviceType	CONTROLLERDEVICETYPE	ekwControllerDeviceType
AddedPartyDeviceType	ADDEDPARTYDEVICETYPE	ekwAddedPartyDeviceType
TransferringDeviceType	TRANSFERRINGDEVICETYPE	ekwTransferringDeviceType
TransferredDeviceType	TRANSFERREDDEVICETYPE	ekwTransferredDeviceType
DivertingDeviceType	DIVERTINGDEVICETYPE	ekwDivertingDeviceType
StateDuration	STATEDURATION	ekwStateDuration
QueueDeviceType	QUEUEDEVICETYPE	ekwQueueDeviceType
NumQueued	NUMQUEUED	ekwNumQueued
SystemEventID	SYSTEMEVENTID	ekwSystemEventID
SystemEventArg1	SYSTEMEVENTARG1	ekwSystemEventArg1
SystemEventArg2	SYSTEMEVENTARG2	ekwSystemEventArg2
SystemEventArg3	SYSTEMEVENTARG3	ekwSystemEventArg3
LineHandle	LINEHANDLE	ekwLineHandle
LineType	LINETYPE	ekwLineType
NewConnectionCallID	NEWCONNECTIONCALLID	ekwNewConnectionCallID
NewConnectionDeviceIDType	NEWCONNECTIONDEVICEIDTYPE	ekwNewConnectionDeviceIDType
SkillGroupPriority	SKILLGROUPPRIORITY	ekwSkillGroupPriority
ObjectState	OBJECTSTATE	ekwObjectState
TrunkUsedDeviceType	TRUNKUSEDDEVICETYPE	ekwTrunkUsedDeviceType
FailureCode	FAILURECODE	ekwFailureCode
NumSkillGroups	NUMSKILLGROUPS	ekwNumSkillGroups

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
EventReasonCode	EVENTREASONCODE	ekwEventReasonCode
ActiveConnectionCallID	ACTIVECONNECTIONCALLID	ekwActiveConnectionCallID
ActiveConnectionDevice IDType	ACTIVECONNECTIONDEVICEIDTYPE	ekwActiveConnectionDeviceID Type
ActiveConnectionDeviceID	ACTIVECONNECTIONDEVICEID	ekwActiveConnectionDeviceID
OtherConnectionCallID	OTHERCONNECTIONCALLID	ekwOtherConnectiomCallID
OtherConnectionDeviceID	OTHERCONNECTIONDEVICEID	ekwOtherConnectionDeviceID
OtherConnectionDeviceIDType	OTHERCONNECTIONDEVICEIDTYPE	ekwOtherConnectionDeviceID Type
HeldConnectionCallID	HELDCONNECTIONCALLID	ekwHeldConnectionCallID
HeldConnectionDeviceID	HELDCONNECTIONDEVICEID	ekwHeldConnectionDeviceID
HeldConnectionDeviceIDType	HELDCONNECTIONDEVICEIDTYPE	ekwHeldConnectionDeviceIDType
CallPlacementType	CALLPLACEMENTTYPE	ekwCallPlacementType
CallMannerType	CALLMANNERTYPE	ekwCallMannerType
ConsultType	CONSULTTYPE	ekwConsultType
FacilityType	FACILITYTYPE	ekwFacilityType
Priority	PRIORITY	ekwPriority
PostRoute	POSTROUTE	ekwPostRoute
Reservation	RESERVATION	ekwReservation
AlertRings	ALERTRINGS	ekwAlertRings
AlertCallerFirst	ALERTCALLERFIRST	ekwAlertCallerFirst
AnsweringMachine	ANSWERINGMACHINE	ekwAnsweringMachine
AllocationState	ALLOCATIONSTATE	ekwAllocationState
TypeOfDevice	TYPEOFDEVICE	ekwTypeOfDevice
ClassOfDevice	CLASSOFDEVICE	ekwClassOfDevice
NumLines	NUMLINES	ekwNumLines
NumCallDevices	NUMCALLDEVICES	ekwNumCallDevices
NumCalls	NUMCALLS	ekwNumCalls
ToneDuration	TONEDURATION	ekwToneDuration

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
PauseDuration	PAUSEDURATION	ekwPauseDuration
SkillGroupState	SKILLGROUPSTATE	ekwSkillGroupState
QueryDeviceDataType	QUERYDEVICEDATATYPE	ekwQueryDeviceDataType
PeripheralErrorCode	PERIPHERALERRORCODE	ekwPeripheralErrorCode
AgentWorkMode	AGENTWORKMODE	ekwAgentWorkMode
MonitorID	MONITORID	ekwMonitorID
MonitoredDeviceType	MONITOREDDEVICETYPE	ekwMonitoredDeviceType
CallOption	CALLOPTION	ekwCallOption
DestinationCountry	DESTINATIONCOUNTRY	ekwDestinationCountry
AnswerDetectMode	ANSWERDETECTMODE	ekwAnswerDetectMode
AnswerDetectTime	ANSWERDETECTTIME	ekwAnswerDetectTime
AnswerDetectControl1	ANSWERDETECTCONTROL1	ekwAnswerDetectControl1
AnswerDetectControl2	ANSWERDETECTCONTROL2	ekwAnswerDetectControl2
ClientPort	CLIENTPORT	ekwClientPort
SessionID	SESSIONID	ekwSessionID
ServicesGranted	SERVICESGRANTED	ekwServicesGranted
PeripheralOnline	PERIPHERALONLINE	ekwPeripheralOnline
Distribution	DISTRIBUTION	ekwDistribution
CallsQNow	CALLSQNOW	ekwCallsQNow
CallsQTimeNow	CALLSQTIMENOW	ekwCallsQTimeNow
LongestCallQNow	LONGESTCALLQNOW	ekwLongestCallQNow
AvailTimeToHalf	AVAILTIMETOHALF	ekwAvailTimeToHalf
LoggedOnTimeToHalf	LOGGEDONTIMETOHALF	ekwLoggedOnTimeToHalf
NotReadyTimetoHalf	NOTREADYTIMETOHALF	ekwNotReadyTimetoHalf
AgentOutCallsToHalf	AGENTOUTCALLSTOHALF	ekwAgentOutCallsToHalf
AgentOutCallsTalkTimeToHalf	AGENTOUTCALLSTALKTIMETOHALF	ekwAgentOutCallsTalkTimeToHalf
AgentOutCallsTimeToHalf	AGENTOUTCALLSTIMETOHALF	ekwAgentOutCallsTimeToHalf
AgentOutCallsHeldToHalf	AGENTOUTCALLSHELDTOHALF	ekwAgentOutCallsHeldToHalf

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
AgentOutCallsHeldTime ToHalf	AGENTOUTCALLSHELDTIMETOHALF	ekwAgentOutCallsHeldTime ToHalf
HandledCallsToHalf	HANDLEDCALLSTOHALF	ekwHandledCallsToHalf
HandledCallsTalkTimeToHalf	HANDLEDCALLSTALKTIMETOHALF	ekwHandledCallsTalkTimeToHalf
HandledCallsTimeToHalf	HANDLEDCALLSTIMETOHALF	ekwHandledCallsTimeToHalf
IncomingCallsHeldToHalf	INCOMINGCALLSHELDTOHALF	ekwIncomingCallsHeldToHalf
IncomingCallsHeldTimeToHalf	INCOMINGCALLSHELDTIMETOHALF	ekwIncomingCallsHeldTime ToHalf
InternalCallsRcvdToHalf	INTERNALCALLSRCVDTOHALF	ekwInternalCallsRcvdToHalf
InternalCallsRcvdTimeToHalf	INTERNALCALLSRCVDTIMETOHALF	ekwInternalCallsRcvdTimeToHalf
InternalCallsHeldToHalf	INTERNALCALLSHELDTOHALF	ekwInternalCallsHeldToHalf
InternalcallsHeldTimeToHalf	INTERNALCALLSHELDTIMETOHALF	ekwInternalcallsHeldTimeToHalf
CallsQHalf	CALLSQHALF	ekwCallsQHalf
CallsQTimeHalf	CALLSQTIMEHALF	ekwCallsQTimeHalf
LongestCallQHalf	LONGESTCALLQHALF	ekwLongestCallQHalf
AvailTimeToday	AVAILTIMETODAY	ekwAvailTimeToday
LoggedOnTimeToday	LOGGEDONTIMETODAY	ekwLoggedOnTimeToday
NotReadyTimeToday	NOTREADYTIMETODAY	ekwNotReadyTimeToday
AgentOutCallsToday	AGENTOUTCALLSTODAY	ekwAgentOutCallsToday
AgentOutCallsTalkTimeToday	AGENTOUTCALLSTALKTIMETODAY	ekwAgentOutCallsTalkTimeToday
AgentOutCallsTimeToday	AGENTOUTCALLSTIMETODAY	ekwAgentOutCallsTimeToday
AgentOutCallsHeldToday	AGENTOUTCALLSHELDTODAY	ekwAgentOutCallsHeldToday
AgentOutCallsHeldTimeToday	AGENTOUTCALLSHELDTIMETODAY	ekwAgentOutCallsHeldTimeToday
HandledCallsToday	HANDLEDCALLSTODAY	ekwHandledCallsToday
HandledCallsTalkTimeToday	HANDLEDCALLSTALKTIMETODAY	ekwHandledCallsTalkTimeToday
HandledCallsTimeToday	HANDLEDCALLSTIMETODAY	ekwHandledCallsTimeToday
IncomingCallsHeldToday	INCOMINGCALLSHELDTODAY	ekwIncomingCallsHeldToday
IncomingCallsHeldTimeToday	INCOMINGCALLSHELDTIMETODAY	ekwIncomingCallsHeldTimeToday
InternalCallsRecvdToday	INTERNALCALLSRECVDTODAY	ekwInternalCallsRecvdToday

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
InternalCallsRecvdTimeToday	INTERNALCALLSRECVDTIMETODAY	ekwInternalCallsRecvdTimeToday
InternalCallsHeldToday	INTERNALCALLSHELDTODAY	ekwInternalCallsHeldToday
InternalCallsHeldTimeToday	INTERNALCALLSHELDTIMETODAY	ekwInternalCallsHeldTimeToday
CallsQToday	CALLSQTODAY	ekwCallsQToday
CallsQTimeToday	CALLSQTIMETODAY	ekwCallsQTimeToday
LongestCallQToday	LONGESTCALLQTODAY	ekwLongestCallQToday
AvailTimeSession	AVAILTIMESESSION	ekwAvailTimeSession
LoggedOnTimeSession	LOGGEDONTIMESESSION	ekwLoggedOnTimeSession
NotReadyTimeSession	NOTREADYTIMESESSION	ekwNotReadyTimeSession
AgentOutCallsSession	AGENTOUTCALLSSESSION	ekwAgentOutCallsSession
AgentOutCallsTalkTime Session	AGENTOUTCALLSTALKTIMESESSION	ekwAgentOutCallsTalkTim eSession
AgentOutCallsTimeSession	AGENTOUTCALLSTIMESESSION	ekwAgentOutCallsTimeSession
AgentOutCallsHeldSession	AGENTOUTCALLSHELDSESSION	ekwAgentOutCallsHeldSession
AgentOutCallsHeldTime Session	AGENTOUTCALLSHELDTIMESESSION	ekwAgentOutCallsHeldTime Session
HandledCallsSession	HANDLEDCALLSSESSION	ekwHandledCallsSession
HandledCallsTalkTimeSession	HANDLEDCALLSTALKTIMESESSION	ekwHandledCallsTalkTimeSession
HandledCallsTimeSession	HANDLEDCALLSTIMESESSION	ekwHandledCallsTimeSession
IncomingCallsHeldSession	INCOMINGCALLSHELDSESSION	ekwIncomingCallsHeldSession
IncomingCallsHeldTime Session	INCOMINGCALLSHELDTIMESESSION	ekwIncomingCallsHeldTime Session
InternalCallsSession	INTERNALCALLSSESSION	ekwInternalCallsSession
InternalCallsTimeSession	INTERNALCALLSTIMESESSION	ekwInternalCallsTimeSession
InternalCallsRecvdSession	INTERNALCALLSRECVDSESSION	ekwInternalCallsRecvdSession
InternalCallsRcvdTimeSession	INTERNALCALLSRCVDTIMESESSION	ekwInternalCallsRcvdTimeSession
InternalCallsHeldSession	INTERNALCALLSHELDSESSION	ekwInternalCallsHeldSession
InternalCallsHeldTimeSession	INTERNALCALLSHELDTIMESESSION	ekwInternalCallsHeldTimeSession
InternalCallsToday	INTERNALCALLSTODAY	ekwInternalCallsToday

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
InternalCallsTimeToday	INTERNALCALLSTIMETODAY	ekwInternalCallsTimeToday
NumNamedVariables	NUMNAMEDVARIABLES	ekwNumNamedVariables
NumNamedArrays	NUMNAMEDARRAYS	ekwNumNamedArrays
HandledCallsAfterCallTime Session	HANDLEDCALLSAFTERCALLTIME SESSION	ekwHandledCallsAfterCallTime Session
HandledCallsAfterCallTime ToHalf	HANDLEDCALLSAFTERCALL TIMETOHALF	ekwHandledCallsAfterCallTime ToHalf
HandledCallsAfterCallTime Today	HANDLEDCALLSAFTERCALL TIMETODAY	ekwHandledCallsAfterCallTime Today
MaxActiveCalls	MAXACTIVECALLS	ekwMaxActiveCalls
MaxHeldCalls	MAXHELDCALLS	ekwMaxHeldCalls
MaxDevicesInConference	MAXDEVICESINCONFERENCE	ekwMaxDevicesInConference
TransferConferenceSetup	TRANSFERCONFERENCESETUP	ekwTransferConferenceSetup
CallEventsSupported	CALLEVENTSSUPPORTED	ekwCallEventsSupported
CallControlSupported	CALLCONTROLSUPPORTED	ekwCallControlSupported
OtherFeaturesSupported	OTHERFEATURESSUPPORTED	ekwOtherFeaturesSupported
CCTimestamp	CCTIMESTAMP	ekwCCTimestamp
CallVariableMask	CALLVARIABLEMASK	ekwCallVariableMask
CalledPartyDisposition	CALLEDPARTYDISPOSITION	ekwCalledPartyDisposition
AgentsLoggedOn	AGENTSLOGGEDON	ekwAgentsLoggedOn
AgentsAvail	AGENTSAVAIL	ekwAgentsAvail
AgentsNotReady	AGENTSNOTREADY	ekwAgentsNotReady
AgentsReady	AGENTSREADY	ekwAgentsReady
AgentsTalkingIn	AGENTSTALKINGIN	ekwAgentsTalkingIn
AgentsTalkingOut	AGENTSTALKINGOUT	ekwAgentsTalkingOu
AgentsTalkingOther	AGENTSTALKINGOTHER	ekwAgentsTalkingOther
AgentsWorkNotReady	AGENTSWORKNOTREADY	ekwAgentsWorkNotReady
AgentsWorkReady	AGENTSWORKREADY	ekwAgentsWorkReady
AgentsBusyOther	AGENTSBUSYOTHER	ekwAgentsBusyOther

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
AgentsReserved	AGENTSRESERVED	ekwAgentsReserved
AgentsHold	AGENTSHOLD	ekwAgentsHold
AvailTimeTo5	AVAILTIMETO5	ekwAvailTimeTo
LoggedOnTimeTo5	LOGGEDONTIMETO5	ekwLoggedOnTimeTo5
NotReadyTimeTo5	NOTREADYTIMETO5	ekwNotReadyTimeTo5
AgentOutCallsTo5	AGENTOUTCALLSTO5	ekwAgentOutCallsTo
AgentOutCallsTalkTimeTo5	AGENTOUTCALLSTALKTIMETO5	ekwAgentOutCallsTalkTimeTo5
AgentOutCallsTimeTo5	AGENTOUTCALLSTIMETO5	ekwAgentOutCallsTimeTo5
AgentOutCallsHeldTo5	AGENTOUTCALLSHELDTO5	ekwAgentOutCallsHeldTo5
AgentOutCallsHeldTimeTo5	AGENTOUTCALLSHELDTIMETO5	ekwAgentOutCallsHeldTimeTo5
HandledCallsTo5	HANDLEDCALLSTO5	ekwHandledCallsTo5
HandledCallsTalkTimeTo5	HANDLEDCALLSTALKTIMETO5	ekwHandledCallsTalkTimeTo5
HandledCallsAfterCallTimeTo5	HANDLEDCALLSAFTERCALLTIMETO5	ekwHandledCallsAfterCallTimeTo5
HandledCallsTimeTo5	HANDLEDCALLSTIMETO5	ekwHandledCallsTimeTo5
IncomingCallsHeldTo5	INCOMINGCALLSHELDTO5	ekwIncomingCallsHeldTo5
IncomingCallsHeldTimeTo5	INCOMINGCALLSHELDTIMETO5	ekwIncomingCallsHeldTimeTo5
InternalCallsRcvdTo5	INTERNALCALLSRCVDTO5	ekwInternalCallsRcvdTo5
InternalCallsRcvdTimeTo5	INTERNALCALLSRCVDTIMETO5	ekwInternalCallsRcvdTimeTo5
InternalCallsHeldTo5	INTERNALCALLSHELDTO5	ekwInternalCallsHeldTo5
InternalCallsHeldTimeTo5	INTERNALCALLSHELDTIMETO5	ekwInternalCallsHeldTimeTo5
CallsQ5	CALLSQ5	ekwCallsQ5
CallsQTime5	CALLSQTIME5	ekwCallsQTime5
LongestCallQ5	LONGESTCALLQ5	ekwLongestCallQ5
SupervisoryAction	SUPERVISORYACTION	ekwSupervisoryAction
TeamID	TEAMID	ekwTeamID
NumberOfAgents	NUMBEROFAGENTS	ekwNumberOfAgents
ConfigOperation	CONFIGOPERATION	ekwConfigOperation
Direction	DIRECTION	ekwDirection
RTPType	RTPTYPE	ekwRTPType

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
BitRate	BITRATE	ekwBitRate
EchoCancellation	ECHOCANCELLATION	ekwEchoCancellation
PacketSize	PACKETSIZE	ekwPacketSize
PayloadType	PAYLOADTYPE	ekwPayloadType
AgentConnectionCallID	AGENTCONNECTIONCALLID	ekwAgentConnectionCallID
AgentConnectionDeviceID Type	AGENTCONNECTIONDEVICEID TYPE	ekwAgentConnectionDeviceID Type
SupervisorConnectionCallID	SUPERVISORCONNECTIONCALLID	ekwSupervisorConnectionCallID
SupervisorConnectionDevice IDType	SUPERVISORCONNECTIONDEVICE IDTYPE	ekwSupervisorConnectionDevice IDType
DeskSettingsMask	DESKSETTINGSMASK	ekwDeskSettingsMask
WrapupDataIncomingMode	WRAPUPDATAINCOMINGMODE	ekwWrapupDataIncomingMode
WrapupDataOutgoingMode	WRAPUPDATAOUTGOINGMODE	ekwWrapupDataOutgoingMode
LogoutNonActivityTime	LOGOUTNONACTIVITYTIME	ekwLogoutNonActivityTime
QualityRecordingRate	QUALITYRECORDINGRATE	ekwQualityRecordingRate
RingNoAnswerTime	RINGNOANSWERTIME	ekwRingNoAnswerTime
WorkModeTimer	WORKMODETIMER	ekwWorkModeTimer
RingNoAnswerDN	RINGNOANSWERDN	ekwRingNoAnswerDN
NumberOfAgentTeams	NUMBEROFAGENTTEAMS	ekwNumberOfAgentTeams
More	MORE	ekwMore
LatSupervisorID	LATSUPERVISORID	ekwLatSupervisorID
SilentMonitorWarningMessage	SILENTMONITORWARNING MESSAGE	ekwSilentMonitorWarningMessage
SilentMonitorAudibleIndication	SILENTMONITORAUDIBLE INDICATION	ekwSilentMonitorAudibleIndication
SupervisorAssistCallMethod	SUPERVISORASSISTCALLMETHOD	ekwSupervisorAssistCallMethod
EmergencyCallMethod	EMERGENCYCALLMETHOD	ekwEmergencyCallMethod
AutoRecordOnEmergency	AUTORECORDONEMERGENCY	ekwAutoRecordOnEmergency
RecordingMode	RECORDINGMODE	ekwRecordingMode
SegmentNumber	SEGMENTNUMBER	ekwSegmentNumber

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
AutoOutCallsSession	AUTOOUTCALLSSESSION	ekwAutoOutCallsSession
AutoOutCallsTalkTimeSession	AUTOOUTCALLSTALKTIMESESSION	ekwAutoOutCallsTalkTimeSession
AutoOutCallsTimeSession	AUTOOUTCALLSTIMESESSION	ekwAutoOutCallsTimeSession
AutoOutCallsHeldSession	AUTOOUTCALLSHELDSESSION	ekwAutoOutCallsHeldSession
AutoOutCallsHeldTimeSession	AUTOOUTCALLSHELDTIMESESSION	ekwAutoOutCallsHeldTimeSession
AutoOutCallsToday	AUTOOUTCALLSTODAY	ekwAutoOutCallsToday
AutoOutCallsTalkTimeToday	AUTOOUTCALLSTALKTIMETODAY	ekwAutoOutCallsTalkTimeToday
AutoOutCallsTimeToday	AUTOOUTCALLSTIMETODAY	ekwAutoOutCallsTimeToday
AutoOutCallsHeldToday	AUTOOUTCALLSHELDTODAY	ekwAutoOutCallsHeldToday
AutoOutCallsHeldTimeToday	AUTOOUTCALLSHELDTIMETODAY	ekwAutoOutCallsHeldTimeToday
AutoOutCallsToHalf	AUTOOUTCALLSTOHALF	ekwAutoOutCallsToHalf
AutoOutCallsTalkTimeToHalf	AUTOOUTCALLSTALKTIMETOHALF	ekwAutoOutCallsTalkTimeToHalf
AutoOutCallsTimeToHalf	AUTOOUTCALLSTIMETOHALF	ekwAutoOutCallsTimeToHalf
AutoOutCallsHeldToHalf	AUTOOUTCALLSHELDTOHALF	ekwAutoOutCallsHeldToHalf
AutoOutCallsHeldTimeToHalf	AUTOOUTCALLSHELDTIMETOHALF	ekwAutoOutCallsHeldTimeToHalf
AutoOutCallsTo5	AUTOOUTCALLSTO5	ekwAutoOutCallsTo
AutoOutCallsTalkTimeTo5	AUTOOUTCALLSTALKTIMETO5	ekwAutoOutCallsTalkTimeTo5
AutoOutCallsTimeTo5	AUTOOUTCALLSTIMETO5	ekwAutoOutCallsTimeTo5
AutoOutCallsHeldTo5	AUTOOUTCALLSHELDTO5	ekwAutoOutCallsHeldTo
AutoOutCallsHeldTimeTo5	AUTOOUTCALLSHELDTIMETO5	ekwAutoOutCallsHeldTimeTo5
PreviewCallsSession	PREVIEWCALLSSESSION	ekwPreviewCallsSession
PreviewCallsTalkTimeSession	PREVIEWCALLSTALKTIMESESSION	ekwPreviewCallsTalkTimeSession
PreviewCallsTimeSession	PREVIEWCALLSTIMESESSION	ekwPreviewCallsTimeSession
PreviewCallsHeldSession	PREVIEWCALLSHELDSESSION	ekwPreviewCallsHeldSession
PreviewCallsHeldTimeSession	PREVIEWCALLSHELDTIMESESSION	ekwPreviewCallsHeldTimeSession
PreviewCallsToday	PREVIEWCALLSTODAY	ekwPreviewCallsToday
PreviewCallsTalkTimeToday	PREVIEWCALLSTALKTIMETODAY	ekwPreviewCallsTalkTimeToday
PreviewCallsTimeToday	PREVIEWCALLSTIMETODAY	ekwPreviewCallsTimeToday
PreviewCallsHeldToday	PREVIEWCALLSHELDTODAY	ekwPreviewCallsHeldToday

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
PreviewCallsHeldTimeToday	PREVIEWCALLSHELDTIMETODAY	ekwPreviewCallsHeldTimeToday
PreviewCallsToHalf	PREVIEWCALLSTOHALF	ekwPreviewCallsToHalf
PreviewCallsTalkTimeToHalf	PREVIEWCALLSTALKTIMETOHALF	ekwPreviewCallsTalkTimeToHalf
PreviewCallsTimeToHalf	PREVIEWCALLSTIMETOHALF	ekwPreviewCallsTimeToHalf
PreviewCallsHeldToHalf	PREVIEWCALLSHELDTOHALF	ekwPreviewCallsHeldToHalf
PreviewCallsHeldTimeToHalf	PREVIEWCALLSHELDTIMETOHALF	ekwPreviewCallsHeldTimeToHalf
PreviewCallsTo5	PREVIEWCALLSTO5	ekwPreviewCallsTo
PreviewCallsTalkTimeTo5	PREVIEWCALLSTALKTIMETO5	ekwPreviewCallsTalkTimeTo5
PreviewCallsTimeTo5	PREVIEWCALLSTIMETO5	ekwPreviewCallsTimeTo5
PreviewCallsHeldTo5	PREVIEWCALLSHELDTO5	ekwPreviewCallsHeldTo
PreviewCallsHeldTimeTo5	PREVIEWCALLSHELDTIMETO5	ekwPreviewCallsHeldTimeTo5
ReserveCallsSession	RESERVECALLSSESSION	ekwReserveCallsSession
ReserveCallsTalkTimeSession	RESERVECALLSTALKTIMESESSION	ekwReserveCallsTalkTimeSession
ReserveCallsTimeSession	RESERVECALLSTIMESESSION	ekwReserveCallsTimeSession
ReserveCallsHeldSession	RESERVECALLSHELDSESSION	ekwReserveCallsHeldSession
ReserveCallsHeldTimeSession	RESERVECALLSHELDTIMESESSION	ekwReserveCallsHeldTimeSession
ReserveCallsToday	RESERVECALLSTODAY	ekwReserveCallsToday
ReserveCallsTalkTimeToday	RESERVECALLSTALKTIMETODAY	ekwReserveCallsTalkTimeToday
ReserveCallsTimeToday	RESERVECALLSTIMETODAY	ekwReserveCallsTimeToday
ReserveCallsHeldToday	RESERVECALLSHELDTODAY	ekwReserveCallsHeldToday
ReserveCallsHeldTimeToday	RESERVECALLSHELDTIMETODAY	ekwReserveCallsHeldTimeToday
ReserveCallsToHalf	RESERVECALLSTOHALF	ekwReserveCallsToHalf
ReserveCallsTalkTimeToHalf	RESERVECALLSTALKTIMETOHALF	ekwReserveCallsTalkTimeToHalf
ReserveCallsTimeToHalf	RESERVECALLSTIMETOHALF	ekwReserveCallsTimeToHalf
ReserveCallsHeldToHalf	RESERVECALLSHELDTOHALF	ekwReserveCallsHeldToHalf
ReserveCallsHeldTimeToHalf	RESERVECALLSHELDTIMETOHALF	ekwReserveCallsHeldTimeToHalf
ReserveCallsTo5	RESERVECALLSTO5	ekwReserveCallsTo
ReserveCallsTalkTimeTo5	RESERVECALLSTALKTIMETO5	ekwReserveCallsTalkTimeTo5
ReserveCallsTimeTo5	RESERVECALLSTIMETO5	ekwReserveCallsTimeTo5

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
ReserveCallsHeldTo5	RESERVECALLSHELDTO5	ekwReserveCallsHeldTo
ReserveCallsHeldTimeTo5	RESERVECALLSHELDTIMETO5	ekwReserveCallsHeldTimeTo5
BargeInCallsSession	BARGEINCALLSSESSION	ekwBargeInCallsSession
BargeInCallsToday	BARGEINCALLSTODAY	ekwBargeInCallsToday
BargeInCallsToHalf	BARGEINCALLSTOHALF	ekwBargeInCallsToHalf
BargeInCallsTo5	BARGEINCALLSTO5	ekwBargeInCallsTo5
InterceptCallsSession	INTERCEPTCALLSSESSION	ekwInterceptCallsSession
InterceptCallsToday	INTERCEPTCALLSTODAY	ekwInterceptCallsToday
InterceptCallsToHalf	INTERCEPTCALLSTOHALF	ekwInterceptCallsToHalf
InterceptCallsTo5	INTERCEPTCALLSTO5	ekwInterceptCallsTo
MonitorCallsSession	MONITORCALLSSESSION	ekwMonitorCallsSession
MonitorCallsToday	MONITORCALLSTODAY	ekwMonitorCallsToday
MonitorCallsToHalf	MONITORCALLSTOHALF	ekwMonitorCallsToHalf
MonitorCallsTo5	MONITORCALLSTO5	ekwMonitorCallsTo
WhisperCallsSession	WHISPERCALLSSESSION	ekwWhisperCallsSession
WhisperCallsToday	WHISPERCALLSTODAY	ekwWhisperCallsToday
WhisperCallsToHalf	WHISPERCALLSTOHALF	ekwWhisperCallsToHalf
WhisperCallsTo5	WHISPERCALLSTO5	ekwWhisperCallsTo
EmergencyCallsSession	EMERGENCYCALLSSESSION	ekwEmergencyCallsSession
EmergencyCallsToday	EMERGENCYCALLSTODAY	ekwEmergencyCallsToday
EmergencyCallsToHalf	EMERGENCYCALLSTOHALF	ekwEmergencyCallsToHalf
EmergencyCallsTo5	EMERGENCYCALLSTO5	ekwEmergencyCallsTo5
AgentsTalkingAutoOut	AGENTSTALKINGAUTOOUT	ekwAgentsTalkingAutoOut
AgentsTalkingPreview	AGENTSTALKINGPREVIEW	ekwAgentsTalkingPreview
AgentsTalkingReservation	AGENTSTALKINGRESERVATION	ekwAgentsTalkingReservation
RouterCallsQNow	ROUTERCALLSQNOW	ekwRouterCallsQNow
ServerMode	SERVERMODE	ekwServerMode
RegisteredServiceID	REGISTEREDSERVICEID	ekwRegisteredServiceID
ServerData	SERVERDATA	ekwServerData

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
LongestRouterCallQNow	LONGESTROUTERCALLQNOW	ekwLongestRouterCallQNow
CampaignID	CAMPAIGNID	ekwCampaignID
QueryRuleID	QUERYRULEID	ekwQueryRuleID
TaskID	TASKID	ekwTaskID
MRDID	MRDID	ekwMRDID
ICMAgentID	ICMAGENTID	ekwICMAgentID
AgentMode	AGENTMODE	ekwAgentMode
IsAvailable	ISAVAILABLE	ekwIsAvailable
MaxTaskLimit	MAXTASKLIMIT	ekwMaxTaskLimit
MaxTasks	MAXTASKS	ekwMaxTasks
MakeRoutable	MAKEROUTABLE	ekwMakeRoutable
Reason	REASON	ekwReason
PreviousICMTaskID	PREVIOUSICMTASKID	ekwPreviousICMTaskID
ICMDisposition	ICMDISPOSITION	ekwICMDisposition
ApplicationDisposition	APPLICATIONDISPOSITION	ekwApplicationDisposition
DoThisWithTaskOnTheWay	DOTHISWITHTASKONTHEWAY	ekwDoThisWithTaskOnTheWay
IsInterrupted	ISINTERRUPTED	ekwIsInterrupted
InterruptingMRDID	INTERRUPTINGMRDID	ekwInterruptingMRDID
InterruptingInvokeID	INTERRUPTINGINVOKEID	ekwInterruptingInvokeID
IsNotReady	ISNOTREADY	ekwIsNotReady
AvailableDuration	AVAILABLEDURATION	ekwAvailableDuration
NumTasks	NUMTASKS	ekwNumTasks
Paused	PAUSED	ekwPaused
TaskState	TASKSTATE	ekwTaskState
OfferDuration	OFFERDURATION	ekwOfferDuration
BeginDuration	BEGINDURATION	ekwBeginDuration
InterruptDuration	INTERRUPTDURATION	ekwInterruptDuration
WrapupDuration	WRAPUPDURATION	ekwWrapupDuration
ReadyCount	READYCOUNT	ekwReadyCount

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
Type	TYPE	ekwType
ClientID	CLIENTID	ekwClientID
ClientPassword	CLIENTPASSWORD	ekwClientPassword
ClientSignature	CLIENTSIGNATURE	ekwClientSignature
AgentExtension	AGENTEXTENSION	ekwAgentExtension
AgentID	AGENTID	ekwAgentID
AgentLastName	AGENTLASTNAME	ekwAgentLastName
AgentFirstName	AGENTFIRSTNAME	ekwAgentFirstName
AgentInstrument	AGENTINSTRUMENT	ekwAgentInstrument
ANI	ANI	ekwANI
UserToUserInfo	USERTOUSERINFO	ekwUserToUserInfo
DNIS	DNIS	ekwDNIS
DialedNumber	DIALEDNUMBER	ekwDialedNumber
CallerEnteredDigits	CALLERENTEREDDIGITS	ekwCallerEnteredDigits
CallVariable1	CALLVARIABLE1	ekwCallVariable1
CallVariable2	CALLVARIABLE2	ekwCallVariable2
CallVariable3	CALLVARIABLE3	ekwCallVariable3
CallVariable4	CALLVARIABLE4	ekwCallVariable4
CallVariable5	CALLVARIABLE5	ekwCallVariable5
CallVariable6	CALLVARIABLE6	ekwCallVariable6
CallVariable7	CALLVARIABLE7	ekwCallVariable7
CallVariable8	CALLVARIABLE8	ekwCallVariable8
CallVariable9	CALLVARIABLE9	ekwCallVariable9
CallVariable10	CALLVARIABLE10	ekwCallVariable10
CTIClientSignature	CTICLIENTSIGNATURE	ekwCTIClientSignature
CTIClientTimestamp	CTICLIENTTIMESTAMP	ekwCTIClientTimestamp
ConnectionDeviceID	CONNECTIONDEVICEID	ekwConnectionDeviceID
AlertingDeviceID	ALERTINGDEVICEID	ekwAlertingDeviceID
CallingDeviceID	CALLINGDEVICEID	ekwCallingDeviceID

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
CalledDeviceID	CALLEDDEVICEID	ekwCalledDeviceID
LastRedirectDeviceID	LASTREDIRECTDEVICEID	ekwLastRedirectDeviceID
AnsweringDeviceID	ANSWERINGDEVICEID	ekwAnsweringDeviceID
HoldingDeviceID	HOLDINGDEVICEID	ekwHoldingDeviceID
RetrievingDeviceID	RETRIEVINGDEVICEID	ekwRetrievingDeviceID
ReleasingDeviceID	RELEASINGDEVICEID	ekwReleasingDeviceID
FailingDeviceID	FAILINGDEVICEID	ekwFailingDeviceID
PrimaryDeviceID	PRIMARYDEVICEID	ekwPrimaryDeviceID
SecondaryDeviceID	SECONDARYDEVICEID	ekwSecondaryDeviceID
ControllerDeviceID	CONTROLLERDEVICEID	ekwControllerDeviceID
AddedPartyDeviceID	ADDEDPARTYDEVICEID	ekwAddedPartyDeviceID
ConnectedPartyCallID	CONNECTEDPARTYCALLID	ekwConnectedPartyCallID
ConnectedPartyDeviceIDType	CONNECTEDPARTYDEVICEIDTYPE	ekwConnectedPartyDeviceIDType
ConnectedPartyDeviceID	CONNECTEDPARTYDEVICEID	ekwConnectedPartyDeviceID
TransferringDeviceID	TRANSFERRINGDEVICEID	ekwTransferringDeviceI
TransferredDeviceID	TRANSFERREDDEVICEID	ekwTransferredDeviceI
DivertingDeviceID	DIVERTINGDEVICEID	ekwDivertingDeviceI
QueueDeviceID	QUEUEDEVICEID	ekwQueueDeviceID
CallWrapupData	CALLWRAPUPDATA	ekwCallWrapupData
UserText	USERTEXT	ekwUserText
NewConnectionDeviceID	NEWCONNECTIONDEVICEID	ekwNewConnectionDeviceID
ObjectName	OBJECTNAME	ekwObjectName
TrunkUsedDeviceID	TRUNKUSEDDEVICEID	ekwTrunkUsedDeviceID
FltSkillGroupNumber	FLTSKILLGROUPNUMBER	ekwFltSkillGroupNumber
FltSkillGroupID	FLTSKILLGROUPID	ekwFltSkillGroupID
FltSkillGroupPriority	FLTSKILLGROUPPRIORITY	ekwFltSkillGroupPriority
FltSkillGroupState	FLTSKILLGROUPSTATE	ekwFltSkillGroupState
AgentPassword	AGENTPASSWORD	ekwAgentPassword
PositionID	POSITIONID	ekwPositionID

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
SupervisorID	SUPERVISORID	ekwSupervisorID
ActiveConnDeviceID	ACTIVECONNDEVICEID	ekwActiveConnDeviceID
OtherConnDeviceID	OTHERCONNDEVICEID	ekwOtherConnDeviceID
HeldConnDeviceID	HELDCONNDEVICEID	ekwHeldConnDeviceID
FacilityCode	FACILITYCODE	ekwFacilityCode
FltLineHandle	FLTLINEHANDLE	ekwFltLineHandle
FltLineType	FLTLINETYPE	ekwFltLineType
CallConnectionCallID	CALLCONNECTIONCALLID	ekwCallConnectionCallID
CallConnectionDeviceIDType	CALLCONNECTIONDEVICEIDTYPE	ekwCallConnectionDeviceIDType
CallConnectionDeviceID	CALLCONNECTIONDEVICEID	ekwCallConnectionDeviceID
CallDeviceType	CALLDEVICETYPE	ekwCallDeviceType
CallDeviceID	CALLDEVICEID	ekwCallDeviceID
CallDeviceConnState	CALLDEVICECONNSTATE	ekwCallDeviceConnState
DTMFString	DTMFSTRING	ekwDTMFString
RouterCallKeyDay	ROUTERCALLKEYDAY	ekwRouterCallKeyDay
RouterCallKeyCallID	ROUTERCALLKEYCALLID	ekwRouterCallKeyCallID
DataStructure	DATASTRUCTURE	ekwDataStructure
CallState	CALLSTATE	ekwCallState
MonitoredDeviceID	MONITOREDDEVICEID	ekwMonitoredDeviceID
AuthorizationCode	AUTHORIZATIONCODE	ekwAuthorizationCode
AccountCode	ACCOUNTCODE	ekwAccountCode
OriginatingDeviceID	ORIGINATINGDEVICEID	ekwOriginatingDeviceID
OriginatingLineID	ORIGINATINGLINEID	ekwOriginatingLineID
ClientAddress	CLIENTADDRESS	ekwClientAddress
ExpandedCallContxt	EXPANDEDCALLCONTXT	ekwExpandedCallContxt
CallControlTable	CALLCONTROLTABLE	ekwCallControlTable
SupervisorInstrument	SUPERVISORINSTRUMENT	ekwSupervisorInstrument
AtcAgentID	ATCAGENTID	ekwAtcAgentID
AgentFlags	AGENTFLAGS	ekwAgentFlags

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
AtcAgentState	ATCAGENTSTATE	ekwAtcAgentState
AtcStateDuration	ATCSTATEDURATION	ekwAtcStateDuration
AgentConnectionDeviceID	AGENTCONNECTIONDEVICEID	ekwAgentConnectionDeviceID
SupervisorConnectionDeviceID	SUPERVISORCONNECTIONDEVICEID	ekwSupervisorConnectionDeviceID
ListTeamID	LISTTEAMID	ekwListTeamID
DefaultDevicePortAddress	DEFAULTDEVICEPORTADDRESS	ekwDefaultDevicePortAddress
ServiceName	SERVICENAME	ekwServiceName
CustomerPhoneNumber	CUSTOMERPHONENUMBER	ekwCustomerPhoneNumber
CustomerAccountNumber	CUSTOMERACCOUNTNUMBER	ekwCustomerAccountNumber
ApplicationPathID	APPLICATIONPATHID	ekwApplicationPathID
ApplicationData	APPLICATIONDATA	ekwApplicationData
ScriptSelector	SCRIPTSELECTOR	ekwScriptSelector
ApplicationString1	APPLICATIONSTRING1	ekwApplicationString1
ApplicationString2	APPLICATIONSTRING2	ekwApplicationString2
AgentInfo	AGENTINFO	ekwAgentInfo
WrapupData	WRAPUPDATA	ekwWrapupData
RouteSelected	ROUTESELECTED	ekwRouteSelected
SkillGroups	SKILLGROUPS	ekwSkillGroups
EnablementMask	ENABLEMENTMASK	ekwEnablementMask
DeviceID	DEVICEID	ekwDeviceID
UniqueObjectID	UNIQUEOBJECTID	ekwUniqueObjectID
ServerObjectID	SERVEROBJECTID	ekwServerObjectID
NewUniqueObjectID	NEWUNIQUEOBJECTID	ekwNewUniqueObjectID
Monitored	MONITORED	ekwMonitored
CallStatus	CALLSTATUS	ekwCallStatus
EventID	EVENTID	ekwEventID
MessageID	MESSAGEID	ekwMessageID
MessageType	MESSAGETYPE	ekwMessageType

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
DriverKey	DRIVERKEY	ekwDriverKey
ecc	ECC	ekwecc
Indicator	INDICATOR	ekwIndicator
ForwardType	FORWARDTYPE	ekwForwardType
ForwardDeviceType	FORWARDDEVICETYPE	ekwForwardDeviceType
Forwarding	FORWARDING	ekwForwarding
LastNumberType	LASTNUMBERTYPE	ekwLastNumberType
ICMCentralControllerTime	ICMCENTRALCONTROLLERTIME	ekwICMCentralControllerTime
Statistics	STATISTICS	ekwStatistics
SupervisorMode	SUPERVISORMODE	ekwSupervisorMode
IncomingOrOutgoing	INCOMINGOROUTGOING	ekwIncomingOrOutgoing
WrapupOKEnabled	WRAPUPOKENABLED	ekwWrapupOKEnabled
Instrument	INSTRUMENT	ekwInstrument
Password	PASSWORD	ekwPassword
State	STATE	ekwState
Text	CLIENT_MESSAGE_TEXT	ekwClientMessageText
BMUDataField	BMUDATAFIELD	ekwBMUDataField
MakeCallSetup	MAKECALLSETUP	ekwMakeCallSetup
CallDeviceConnectionState	CALLDEVICECONNECTIONSTATE	ekwCallDeviceConnectionState
InternalCallsRcvdToday	INTERNALCALLSRCVDTODAY	ekwInternalCallsRcvdToday
InternalCallsRcvdTimeToday	INTERNALCALLSRCVDTIMETODAY	ekwInternalCallsRcvdTimeToday
InternalCallsRcvdSession	INTERNALCALLSRCVDSESSION	ekwInternalCallsRcvdSession
ReservationCallsSession	RESERVATIONCALLSSESSION	ekwReservationCallsSession
ReservationCallsTalkTimeSession	RESERVATIONCALLSTALKTIMESESSION	ekwReservationCallsTalkTimeSession
ReservationCallsTimeSession	RESERVATIONCALLSTIMESESSION	ekwReservationCallsTimeSession
ReservationCallsHeldSession	RESERVATIONCALLSHELDSESSION	ekwReservationCallsHeldSession
ReservationCallsHeldTimeSession	RESERVATIONCALLSHELDTIMESESSION	ekwReservationCallsHeldTimeSession

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
ReservationCallsToday	RESERVATIONCALLSTODAY	ekwReservationCallsToday
ReservationCallsTalkTime Today	RESERVATIONCALLSTALKTIME TODAY	ekwReservationCallsTalkTime Today
ReservationCallsHeldTime Today	RESERVATIONCALLSHELDTIME TODAY	ekwReservationCallsHeldTim eToday
ReservationCallsTimeToday	RESERVATIONCALLSTIMETODAY	ekwReservationCallsTimeToday
ReservationCallsHeldToday	RESERVATIONCALLSHELDTODAY	ekwReservationCallsHeldToday
ReservationCallsTo5	RESERVATIONCALLSTO5	ekwReservationCallsTo5
ReservationCallsTalkTimeTo5	RESERVATIONCALLSTALKTIMETO5	ekwReservationCallsTalkTimeTo5
ReservationCallsHeldTimeTo5	RESERVATIONCALLSHELDTIMETO5	ekwReservationCallsHeldTimeTo5
ReservationCallsTimeTo5	RESERVATIONCALLSTIMETO5	ekwReservationCallsTimeTo5
ReservationCallsHeldTo5	RESERVATIONCALLSHELDTO5	ekwReservationCallsHeldTo5
ReservationCallsToHalf	RESERVATIONCALLSTOHALF	ekwReservationCallsToHalf
ReservationCallsTalkTime ToHalf	RESERVATIONCALLSTALKTIME TOHALF	ekwReservationCallsTalkTime ToHalf
ReservationCallsHeldTime ToHalf	RESERVATIONCALLSHELDTIMETO HALF	ekwReservationCallsHeldTimeTo Half
ReservationCallsTimeToHalf	RESERVATIONCALLSTIMETOHALF	ekwReservationCallsTimeToHalf
ReservationCallsHeldToHalf	RESERVATIONCALLSHELDTOHALF	ekwReservationCallsHeldToHalf
EventTime	EVENTTIME	ekwEventTime
CurrentServer	CURRENTSERVER	ekwCurrentServer
CurrentPort	CURRENTPORT	ekwCurrentPort
ReasonCode	REASONCODE	ekwReasonCode
FailedServer	FAILEDSERVER	ekwFailedServer
HeartBeatInterval	HEARTBEATINTERVAL	ekwHeartBeatInterval
MissedHeartBeats	MISSEDHEARTBEATS	ekwMissedHeartbeats
ClientAgentTemporaryID	CLIENTAGENTTEMPORARYID	ekwClientAgentTemporaryID
ClassIdentifier	CLASSIDENTIFIER	ekwClassIdentifier
IsSupervisor	ISSUPERVISOR	ekwIsSupervisor
Extension	EXTENSION	ekwExtension

Table B-1 CTI OS Keywords (continued)

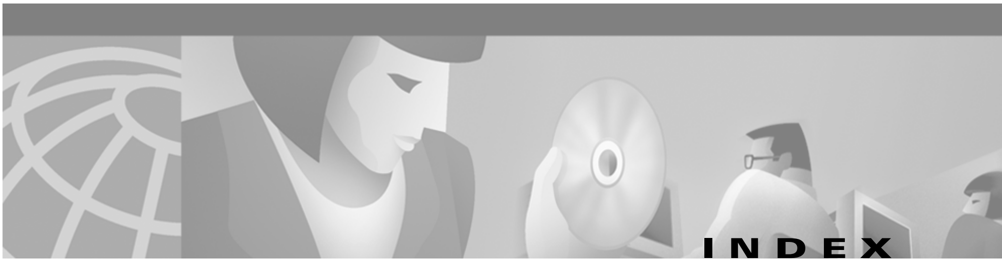
Text String	Constant String	Enumerated Integer Value
LastError	LASTERROR	ekwLastError
CallReference	CALLREFERENCE	ekwCallReference
AgentReference	AGENTREFERENCE	ekwAgentReference
AgentCallReference	AGENTCALLREFERENCE	ekwAgentCallReference
MonitorFlag	MONITORFLAG	ekwMonitorFlag
CtiOsA	CTIOSA	ekwCtiOsA
CtiOsB	CTIOSB	ekwCtiOsB
PortA	PORTA	ekwPortA
PortB	PORTB	ekwPortB
Heartbeat	HEARTBEAT	ekwHeartbeat
MaxHeartbeats	MAXHEARTBEATS	ekwMaxHeartbeats
CurrentAgent	CURRENTAGENT	ekwCurrentAgent
ActiveCall	ACTIVECALL	ekwActiveCall
CurrentFilter	CURRENTFILTER	ekwCurrentFilter
Agents	AGENTS	ekwAgents
Calls	CALLS	ekwCalls
WaitObjects	WAITOBJECTS	ekwWaitObjects
ObjectReferences	OBJECTREFERENCES	ekwObjectReferences
DesktopSettings	DESKTOPSETTINGS	ekwDesktopSettings
CallVariables	CALLVARIABLES	ekwCallVariables
CallReferenceObjectID	CALLREFERENCEOBJECTID	ekwCallReferenceObjectID
RecordEnable	RECORDENABLE	ekwRecordEnable
TimeOfDay	TIMEOFDAY	ekwTimeOfDay
ConnectionMode	CONNECTIONMODE	ekwConnectionMode
TryingServer	TRYINGSERVER	ekwTryingServer
TryingPort	TRYINGPORT	ekwTryingPort
TryingSince	TRYINGSINCE	ekwTryingSince
MessageFilter	MESSAGEFILTER	ekwMessageFilter
DesktopType	DESKTOPTYPE	ekwDesktopType

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
SupervisorBtnEnablementMask	SUPERVISORBTNENABLEMENT MASK	ekwSupervisorBtnEnablement Mask
ErrorMessage	ERRORMESSAGE	ekwErrorMessage
Mode	MODE	ekwMode
Call	CALL	ekwCall
SkillGroupName	SKILLGROUPNAME	ekwSkillGroupName
OldUniqueObjectID	OLDUNIQUEOBJECTID	ekwOldUniqueObjectID
EmbeddedArgs	EMBEDDEDARGS	ekwEmbeddedArgs
EventMask	EVENTMASK	ekwEventMask
ID	ID	ekwID
Filter	FILTER	ekwFilter
CILConnectionID	CILCONNECTIONID	ekwCILConnectionID
IncomingWrapupStrings	INCOMINGWRAPUPSTRINGS	ekwIncomingWrapupStrings
OutgoingWrapupStrings	OUTGOINGWRAPUPSTRINGS	ekwOutgoingWrapupStrings
NotReadyReasonCodes	NOTREADYREASONCODES	ekwNotReadyReasonCodes
LogoutReasonCodes	LOGOUTREASONCODES	ekwLogoutReasonCodes
DriverID	DRIVERID	ekwDriverID
SkillGroupNumbers	SKILLGROUPNUMBERS	ekwSkillGroupNumbers
Target	TARGET	ekwTarget
Source	SOURCE	ekwSource
SupervisorKey	SUPERVISORKEY	ekwSupervisorKey
BargedInCallID	BARGEDINCALLID	ekwBargedInCallID
TeamUniqueID	TEAMUNIQUEID	ekwTeamUniqueID
ValidCalls	VALIDCALLS	ekwValidCalls
StatusBarMessage	STATUSBARMESSAGE	ekwStatusBarMessage
FailoverRequired	FAILOVERREQUIRED	ekwFailoverRequired
CTIOSSystemEventID	CTIOSSYSTEMEVENTID	ekwCTIOSSystemEventID
AutoLogin	AUTOLOGIN	ekwAutoLogin
SavedAgentState	SAVEDAGENTSTATE	ekwSavedAgentState

Table B-1 CTI OS Keywords (continued)

Text String	Constant String	Enumerated Integer Value
WaitingforRecovery	WAITINGFORRECOVERY	ekwWatingForRecovery
SavedLoginInfo	SAVEDLOGININFO	ekwSavedLoginInfo
EventType	EVENTTYPE	ekwEventType
RequestID	REQUESTID	ekwRequestID
NonClientRequest	NONCLIENTREQUEST	ekwNonClientRequest
TimerInterval	TIMERINTERVAL	ekwTimerInterval
TimerCount	TIMERCOUNT	ekwTimerCount
CTIClients	CTICLIENTS	ekwCTIClients



INDEX