

CTI Toolkit Version 4.0 Developer Guide

Version 4.0, 1





CONTENTS

Introduction to Salestorce CRM Call Center
About This Guide
System Requirements
Chapter 1: System Architecture
About CTI Systems
About CTI Adapters
About CTI Connectors
About SoftPhone Connectors
Processing Events from a CTI System
Processing Commands from a SoftPhone
Registering a CTI adapter with a CTI System
About Salesforce
About SoftPhones
Chapter 2: The Salesforce CTI Toolkit
Contents of the CTI Toolkit Code Package
The Demo Adapter
Setting Up the Demo Adapter
Using the Demo Adapter
Customizing the Demo Adapter
CTI Connector Classes
Best Practices for Coding with the CTI Toolkit
The CTIConstants.h File
The 'L' Literal String and Character Prefix
Method Name in CCTIUserInterface
Specifying a Valid CTI Client Key
Chapter 3: Customizing Salesforce CRM Call Center
Customizing a CTI Connector
Building a New CTI Connector
Setting Up a New CTI Connector Solution
Building a CTI Connector Solution in Visual Studio
Adding a COM Base Class to a CTI Adapter Project
Instantiating a CCTIUserInterface Subclass
Referencing Your New CTI Adapter in BrowserConnector
Configuring app.config
Writing an Event Sink

Contents

Determining the CTI System Events that Require an Event Sink Handler
Writing an Event Sink Handler
SoftPhone Modification Options
Using the Virtual Keyword in Your CCTIUserInterface .h File
Implementing Call Center User Command Messages
Writing the Initialize Method for CCTIUserInterface
Enabling Call Center User Login
Enabling One-Step Transfers and Conferences
Enabling a Set of Buttons
Changing the Display Order of SoftPhone Buttons
Adding a Button
Enabling Reason Codes
Mapping CTI System Agent States to Salesforce CRM Call Center User States
Displaying Salesforce CRM Call Center User States
Adding a New Salesforce CRM Call Center User State
Adding a Custom Logo
Modifying Displayed Call Information
Customizing Automatically-Generated Call Logs
Defining Custom SoftPhone Labels
Translating Custom SoftPhone Labels
Chapter 4: Call Center Definition Files
Call Center Definition File XML Format
Required Call Center Elements and Attributes
Specifying Values for <item> Elements</item>
Sample Call Center Definition File
Chapter 5: Packaging and Publishing a CTI Adapter
Packaging a CTI Adapter
Selling a CTI Adapter on the AppExchange
Chapter 6: Configuring HTTPS for a CTI Adapter
Adding HTTPS to a Call Center
Adding HTTPS to CTI Adapters
Creating and Installing a Security Certificate
Chapter 7: CTI Adapter Log Files
Chapter 8: Salesforce CRM Call Center API Reference
describeSoftphoneLayout()
CallCenter
AdditionalNumber
Frequently Asked Questions
Q: What is the difference between Salesforce CRM Call Center and the CTI Toolkit?

Contents

Q: Why does Salesforce CRM Call Center involve a client-side application? Isn't Salesforce the
"No Software" company?
Q: Does the CTI adapter support multiple telephone lines?
Q: Is it possible to have multiple CTI adapters working in parallel against the same switch? 66
Q: Does a CTI adapter require any investment or changes to a switch?
Q: Does Salesforce CRM Call Center require VoIP?
Q: How long does it usually take for a partner to write a custom CTI adapter?
Q: Is there a list of the telephony platforms that are currently covered?
Q: How can I demo a CTI adapter?
Q: I'm seeing 'L' prefixes in front of all the string and character literals in the CTI Toolkit code.
Why are these present?
Q: My SoftPhone is not behaving the way that I expect it to. How can I troubleshoot it? 68
Glossary
Index

INTRODUCTION TO SALESFORCE CRM CALL CENTER

(1)

Important: CTI Toolkit, also known as the Desktop CTI, is retired. The CTI Toolkit is no longer supported, and any adapters built on the CTI Toolkit won't work. To continue using CTI functionality, migrate to Salesforce Open CTI.

Call Center integrates Salesforce with third-party computer-telephony integration (CTI) systems. Call center users can see Salesforce information for incoming calls, make out-going calls directly from Salesforce, and report on call outcome, duration, and more.

After a call center is set up, call center users can make and receive calls with a softphone. Each softphone looks and behaves differently because each CTI implementation is unique. In the console and in Lightning Experience, softphones appear in a footer. In Salesforce Classic, softphones appear in the left sidebar of every Salesforce page.

The call center is all about customization. You can modify softphone layouts and assign specific layouts to selected user profiles. You can also add phone numbers to call center directories so your users all have access to key phone numbers. As your needs change, your call center can be customized and changed too. As an admin, some customization you can do yourself. However, you might want to work with your developers or partners to make functionality changes.

About This Guide

The CTI Toolkit 4.0 Developer's Guide is a reference for developers who want to customize Salesforce CRM Call Center beyond the scope of what is currently offered by Salesforce. This guide includes the following information:

Chapter	Description
Introduction to Salesforce CRM Call Center	Salesforce CRM Call Center overview, documentation, and system requirements
	See About Salesforce CRM Call Center Documentation on page 2 and System Requirements on page 2 for details.
System Architecture	Salesforce CRM Call Center system architecture, including CTI systems, CTI adapters, the role of the Salesforce database, and SoftPhones
	See System Architecture on page 4 for details.
The CTI Toolkit	Descriptions of the types of files that are included in a CTI adapter code package and notes on CTI Toolkit coding practices
	See The Salesforce CTI Toolkit on page 9 for details.
Customizing Salesforce CRM Call Center	Options for customizing Salesforce CRM Call Center, including project setup, event sink implementation, and SoftPhone modification options
	See Customizing Salesforce CRM Call Center on page 19 for details.
Call Center Definition Files	How to define a default XML call center definition file for a new CTI adapter
	See Call Center Definition Files for details.

Steps for bundling new CTI adapter code libraries into an installer and publishing it on Force.com AppExchange.
See Packaging and Publishing a CTI Adapter on page 53 for details.
Steps for setting up secure hypertext transfer protocol (HTTPS) for CTI adapters and call centers.
See Configuring HTTPS for a CTI Adapter on page 54 for details.
A description of the location and format of CTI adapter log files
See CTI Adapter Log Files on page 56 for details.
Force.com API objects and methods that relate to Salesforce CRM Call Center
See Salesforce CRM Call Center API Reference on page 57 for details.
A set of frequently-asked questions regarding troubleshooting and CTI adapter architecture
See Frequently Asked Questions on page 66 for details.
A glossary of terms related to Salesforce CRM Call Center
See Glossary on page 69 for details.

About Salesforce CRM Call Center Documentation

For a complete understanding of Salesforce CRM Call Center, read the CTI Toolkit 4.0 Developer's Guide.

The following additional documentation might also deepen your understanding of Salesforce CRM Call Center:

- Getting Started with your SoftPhone A tip sheet that provides an overview of Salesforce CRM Call Center functionality for call center
 users
- Getting Started with Setting Up Call Centers A tip sheet that provides an overview of how an administrator can configure Salesforce CRM Call Center for an organization.
- Using the Salesforce CRM Call Center Demo Adapter A tip sheet that provides an overview of the Salesforce CRM Call Center demo adapter.
- Salesforce CTI Toolkit Code Reference An online help system that provides detailed information about the objects and methods in the Salesforce CRM Call Center source code that Salesforce provides.

Standard Salesforce CRM Call Center functionality is also fully documented in the Salesforce online help.

System Requirements

Salesforce CRM Call Center requires the installation of a light-weight computer-telephony integration (CTI) adapter on every Salesforce user's machine. The *minimum* system requirements for an adapter are:

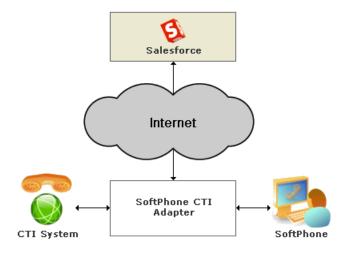
For adapters built with CTI version 4.0 or higher:

- Microsoft® Internet Explorer® 8; Mozilla® Firefox® 3.5; Apple® Safari® 4; Google Chrome™ 10.0 (Internet Explorer 11 isn't supported)
- Microsoft Windows® XP (with Microsoft .NET framework)
- For adapters built with previous versions of CTI:
 - Internet Explorer 7 or 8; Firefox 3.5 or 3.6 (Safari, Chrome, and Internet Explorer 11 aren't supported)
 - For Windows 7 32-bit, 32-bit Internet Explorer 8; Firefox 3.5 or 3.6
 - For Windows 7 64-bit, 64-bit Internet Explorer 8; Firefox 3.5 or 3.6
 - 256 MB of RAM (512 MB recommended)
 - 20 MB of disk space minimum
 - Intel® Pentium® II processor, 500 MHz or above
 - Windows XP

Pre-built adapters for several different CTI systems are available on http://sites.force.com/appexchange.

CHAPTER 1 System Architecture

This diagram shows the system architecture of Salesforce CRM Call Center:



Salesforce CRM Call Center uses the following components:

CTI system

A set of telephony hardware and software that supports integration with other computer systems. The CTI system provides the base framework for the calls that are made and received through a SoftPhone

For more information, see About CTI Systems on page 5.

CTI adapter

A lightweight software program that controls the appearance and behavior of a Salesforce softphone. The adapter acts as an intermediary between a third-party computer telephony integration (CTI) system, Salesforce, and a Salesforce CRM Call Center user. It must be installed on any machine that needs access to Salesforce CRM Call Center functionality.

For more information, see About CTI Adapters on page 5.

Salesforce

The source of call-related data and SoftPhone layout instructions. In addition to providing access to records that are related to an incoming call, Salesforce stores call center connection information and directories, SoftPhone layouts, and automatic call logs.

For more information, see About Salesforce on page 7.

SoftPhone

A customizable call control tool that appears in the sidebar of every Salesforce page. A SoftPhone requires a connection to a third-party CTI system to make or receive phone calls.

For more information, see About SoftPhones on page 7.

System Architecture About CTI Systems

About CTI Systems

A Computer-Telephony Integration (CTI) system is a set of telephony hardware and software that supports integration with other computer systems. The CTI system provides the base framework for the calls that are made and received through a SoftPhone.

Salesforce CRM Call Center supports any CTI system that operates using an event model, in which a CTI system sends an event to all registered listeners for any action that occurs in the phone system. For example, when a user's telephone rings, a CTI system broadcasts a "RINGING" event. A CTI adapter, the Salesforce CRM Call Center component that acts as a listener, receives this event and updates the SoftPhone as appropriate.

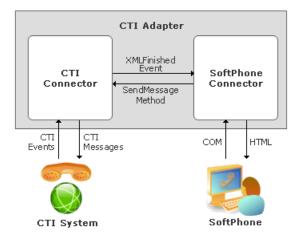
About CTI Adapters

A CTI adapter is a lightweight software program that controls the appearance and behavior of a Salesforce softphone. The adapter acts as an intermediary between a third-party computer telephony integration (CTI) system, Salesforce, and a Salesforce CRM Call Center user. It must be installed on any machine that needs access to Salesforce CRM Call Center functionality.

Because a CTI adapter communicates directly with an individual CTI system, an organization must use a different CTI adapter for each type of CTI system that is in use. For example, if an organization wants to integrate one call center that runs Cisco IPCC Enterprise[™] and one call center that runs Cisco IPCC Express[™], the organization must have two CTI adapters available. A call center user's machine only requires the SoftPhone CTI adapter for the call center to which it connects.

A number of prebuilt CTI adapters for different CTI systems are available on Force.com AppExchange at http://sites.force.com/appexchange. If your CTI system is not currently supported, you can still use Salesforce CRM Call Center, but you must either build a new CTI adapter, or customize sample CTI adapter code provided by Salesforce.

As illustrated in this diagram, a CTI adapter consists of two components: a CTI connector that maintains an XML representation of the SoftPhone and communicates directly with the CTI system, and a SoftPhone connector that converts SoftPhone XML to HTML and distributes it to a call center user's browser.



About CTI Connectors

A CTI connector is the component of a CTI adapter that controls the functionality and appearance of a call center user's SoftPhone. The CTI connector receives CTI system events, displays SoftPhone user interface elements, and searches for Salesforce records that are related

System Architecture About SoftPhone Connectors

to incoming calls. During a call the CTI adapter generates an automatic call log, and when a call is over it allows the call center user to enter reason codes for why the call ended, or for why the user wishes to log out of the CTI system.

The CTI connector works by maintaining an XML representation of the SoftPhone that it updates whenever it receives relevant CTI system events or SoftPhone connector messages. After the CTI connector updates its XML, it fires a UIRefresh event. This event signals the SoftPhone connector that the SoftPhone needs to be updated in the call center user's browser.

You can customize a CTI connector implementation with new functionality, or build a new CTI connector if you want to support a CTI system that does not already have an adapter. Although a CTI connector can be written in any language that supports COM, Salesforce only provides code for CTI connectors that are written in managed C++ using the CTI library (CTIAdapterLib) that Salesforce provides. When you customize a CTI connector, you extend the objects and methods already defined in CTIAdapterLib.

For more information on CTI connector customization options, see Customizing a CTI Connector on page 19.

About SoftPhone Connectors

A SoftPhone connector is the component of a CTI adapter that converts SoftPhone XML into HTML and distributes it to a call center user's browser. Every time a call center user begins a new Salesforce session, the SoftPhone connector downloads SoftPhone labels and layouts from Salesforce. The SoftPhone connector uses these labels and layouts to generate SoftPhone HTML from the XML passed in by the CTI connector's UIRefresh event.

When the SoftPhone connector receives a new set of XML from the CTI connector through the UIRefresh event:

- 1. The SoftPhone connector converts the attached XML into HTML. All labels and layouts are resolved during this step.
- 2. The call center user's browser maintains a continuous link with the SoftPhone connector through AJAX. When the browser connector generates new SoftPhone HTML, it updates the SoftPhone in the user's browser.

Processing Events from a CTI System

CTI system events are messages from a CTI system to its registered listeners, such as a CTI adapter. An adapter handles events from a CTI system as follows:

- 1. When a CTI system broadcasts an event, the adapter's CTI connector receives it through its event sink class. The event sink converts any data that is attached to the event into a standard format, and then calls the corresponding event handler in the CTI connector's CCTIUserInterface class. CTI system events that do not affect the SoftPhone, such as those that provide statistical data, are ignored by the event sink.
- 2. The CCTIUserInterface event handler updates the XML representation of the SoftPhone by hiding and revealing individual SoftPhone components, such as buttons and dial pads, and by performing searches for related Salesforce records, such as contacts and accounts. When finished, the CTI connector broadcasts a UIRefresh event to the SoftPhone connector.
- 3. The SoftPhone connector uses the XML attached to the UIRefresh event to render an HTML version of the SoftPhone.
- **4.** The call center user's browser maintains a continuous link with the SoftPhone connector through AJAX. When the browser connector generates new SoftPhone HTML, it updates the SoftPhone in the user's browser.

Processing Commands from a SoftPhone

Commands are messages from a call center user's SoftPhone to a CTI system. An adapter handles SoftPhone commands as follows:

1. When a call center user clicks a button in the SoftPhone (for example, **Hang Up**), the SoftPhone sends an HTML command message to the SoftPhone connector that includes the ID of the button that was clicked (for example, http://localhost:7332/HANGUP).

- 2. The SoftPhone connector translates the button click into an XML message, and then sends it to the CTI connector with the UIAction method.
- 3. When the CTI connector receives the message from the SoftPhone connector, one of two actions takes place:
 - The UIHandleMessage method in CCTIUserInterface forwards the message to the CTI system and no other changes occur until the CTI system sends a new event.
 - For example, the HANGUP message is passed to the CTI system so that the connected call can be terminated. Once the call ends in the phone system, the CTI system broadcasts a hang-up confirmation event and the CTI connector proceeds as described in Processing Events from a CTI System on page 6.
 - The UIHandleMessage method in CCTIUserInterface updates the CTI connector's XML representation of the SoftPhone by hiding and revealing individual SoftPhone components, such as buttons and dial pads, and by performing searches for related Salesforce records, such as contacts and accounts. When finished, the CTI connector broadcasts a UIRefresh event to the SoftPhone connector. The SoftPhone connector receives the XML and uses it to render an updated SoftPhone.

Registering a CTI adapter with a CTI System

When a call center user logs in to Salesforce, the CTI adapter that is installed on the user's machine must first register with a CTI system before it can receive CTI events. Registration occurs as follows:

- 1. Upon logging in to Salesforce, the browser acquires a session ID that authenticates the user for the duration of their Salesforce session. The CTI adapter uses this session ID to query Salesforce for information related to the user's assigned call center. All data associated with the call center is returned to the adapter.
- 2. The adapter's CTILogin method uses the call center data to connect with the specified CTI system. In most cases the call center user must provide authentication information to the CTI system. After logging in for the first time, users have the option of saving their CTI system credentials within Salesforce for automatic login in the future.

About Salesforce

The Salesforce database stores call center connection information, SoftPhone formatting instructions, and call center directory numbers. In addition, the CTI connector component of a CTI adapter uses Salesforce Object Query Language (SOQL) and Salesforce Object Search Language (SOSL) to retrieve Salesforce records that are associated with data attached to incoming calls, such as an account number (SOQL) or the phone number from which a customer is calling (SOSL).

About SoftPhones

A softphone is a customizable call-control tool that appears to users assigned to a call center. Although administrators and developers can customize SoftPhones, they usually include the following components when built with version 3.0 of the CTI Toolkit:

System Architecture About SoftPhones



Call center state area

Includes a drop-down list that lets you specify whether you're ready to receive calls.

Phone line header

Shows the status of the phone line. A status icon changes colors and blinks (), and provides a text description. You can click the name of the line (Line 1) to show or hide the line's dial pad and call information area.

Call information area

Shows data related to the call, such as the phone number the customer used to dial, the duration of the call, and links to any records associated with the call.

Call button area

Shows buttons that let you make call commands, such as dialing, hanging up, putting a caller on hold, transferring, conferencing, and opening a second line while on a call.

My Calls Today report

Opens a report of all the calls you've made or received in the last day.

SoftPhone logo

Displays a customizable logo for each CTI adapter.

CHAPTER 2 The Salesforce CTI Toolkit

(1) Important: CTI Toolkit, also known as the Desktop CTI, is retired. The CTI Toolkit is no longer supported, and any adapters built on the CTI Toolkit won't work. To continue using CTI functionality, migrate to Salesforce Open CTI.

The Salesforce CTI Toolkit provides you with all of the source code, libraries, and files you need to develop your own custom CTI adapter.

There are three versions of the CTI Toolkit. Each version provides users with different Salesforce CRM Call Center functionality. However, Salesforce only distributes CTI Toolkit version 4.0 or higher. The following table lists the functionality available in CTI adapters built with each CTI Toolkit:

Functionality	Version 1.0 or Higher	Version 2.0 or Higher	Version 3.0 or Higher	Version 4.0 or Higher
Change the fields and order of fields that display in a SoftPhone	✓	<u>~</u>	<u>~</u>	✓
Change the objects and order of links to objects that display in a SoftPhone	✓	~	~	~
Specify the fields that display in the SoftPhone if a single record for a particular object is found	✓	~	~	~
Specify screen pop settings for inbound calls with single, multiple, or no record matches		~	~	✓
Specify screen pops for inbound calls to display in browser windows that are already open, or in new browser windows or tabs		▽	▽	~
Specify screen pops to Visualforce pages for inbound calls		▽	▽	<u>~</u>
Specify screen pops to search pages for inbound calls with multiple record matches		▽	✓	▼
View a call center's version in a Version		V	<u>~</u>	~

Functionality	Version 1.0 or Higher	Version 2.0 or Higher	Version 3.0 or Higher	Version 4.0 or Higher
field (from Setup, enter Call Centers in the Quick Find box, then select Call Centers and choose a call center)				
View an enhanced SoftPhone user-interface in the footer of the Salesforce console			~	~
Log calls in the customizable interaction log of the Salesforce console			✓	✓
Support browsers that are cross-domain messaging compatible				
Reduce CTI adapter size and complexity				~

You can download the CTI Toolkit by visiting developer.salesforce.com.

Contents of the CTI Toolkit Code Package

All CTI toolkit code packages include the following components:

SoftPhone connector executable (SalesforceCTI.exe)

This component runs the SoftPhone connector portion of a SoftPhone CTI adapter. It comes as a pre-compiled executable file with the 😚 logo.

CTI connector code package (<cti_system_adapter_name>.Primary Output)

This Visual Studio.NET 2008 code package contains the classes that make up a CTI connector for the specified CTI system. Once you have customized this code package, it is compiled into a .dll file. See CTI Connector Classes on page 12 for a description of the classes in this package.

Demo CTI connector code package (DemoAdapter.Primary Output)

This Visual Studio.NET 2008 code package contains the classes that make up the CTI connector for the Salesforce CRM Call Center Demo Adapter. For more information, see The Demo Adapter on page 11.

Note: A compiled version of the demo adapter is also available on Salesforce Developers for quick installation.

The Salesforce Office Toolkit Library (SF MSApi4.dll)

This .dll component is required for access to the Salesforce Force.com API.

Microsoft XML Library 6 (msxml6.dll and msxml6r.dll)

These .dll files allow the SoftPhone connector to translate the SoftPhone user interface XML into HTML.

The Salesforce CTI Toolkit

The Demo Adapter

Any dynamically-linked libraries that are required for the CTI system

Most CTI adapter implementations require additional CTI-system-specific .dll files to enable communication.

Default call center definition files (<cti_system_adapter_name>.xml or DemoAdapter.xml)

These XML files provide the default definition format for a call center that uses the specified adapter. For more information, see Call Center Definition Files on page 48.

For a complete list of the files that are included with a particular CTI adapter code package, see its associated Adapter Guide.

The Demo Adapter

Included with the Salesforce CTI Toolkit is the Salesforce CRM Call Center demo adapter, an executable that can be used to mimic SoftPhone functionality without requiring an operational CTI system. Users can manually trigger CTI system events, such as an incoming call with an automatic number identification (ANI) value, an incoming call without an ANI value, an incoming conference call, and an incoming transfer request. The SoftPhone operates as if these events had been triggered by a valid CTI system and allows the user to view associated records and generate call logs as if they had been real. As a developer, you can use the demo adapter to quickly prototype new SoftPhone functionality, or as a template for a new CTI connector implementation.

Setting Up the Demo Adapter

To install and set up a Salesforce CRM Call Center demo adapter:

- 1. Download the demo adapter installation package from the Salesforce Developers website at https://developer.salesforce.com/page/CTI_Toolkit.
- 2. From the demo adapter's installation directory, run Setup.exe as a Windows administrator user.
- **3.** Log in to Salesforce as an administrator user.
- **4.** From Setup, enter *Call Centers* in the Quick Find box, then select **Call Centers**.
- 5. Click Import.
- 6. Click **Browse** and navigate to the call center definition file in your demo adapter installation directory. In default installations, this file is located at C:\Program Files\salesforce.com\Demo Adapter\DemoAdapter.xml.
- 7. Click Import.
- 8. Click Demo Call Center Adapter.
- 9. Click Manage Call Center Users.
- 10. Click Add More Users.
- 11. Enter search criteria to find a Salesforce user who can be a demo adapter user. Click Find.
- 12. Select the checkbox next to the name of one or more users with access to the demo adapter. Click Add to Call Center.
- 13. Log in to Salesforce as a demo adapter user.
- 14. Log in to the SoftPhone. The demo adapter's login screen accepts any values as valid credentials.

Using the Demo Adapter

Once you have logged in to the demo adapter, you can use it like a normal SoftPhone. To trigger a CTI system event while you are using the demo adapter SoftPhone:

1. Right-click the CTI adapter system tray icon (😭) in the lower right corner of your computer screen.

- 2. Select Go To Wrapup After Call if you want to view wrap-up reasons after ending a demo call.
- **3.** Select one of these demo CTI system events:

Demo CTI System Event	Description
Call From 415-555-1212	An inbound call from (415) 555-1212 that does not include interactive voice response (IVR) data.
Auto Answer Call From 415-555-1212	An inbound call from (415) 555-1212 that answers automatically.
Call Via IVR (Case 1001)	An inbound call from (415) 555-1212 that includes IVR data identifying case 1001.
Transfer From x8120	A call that has been transferred from another call center user at extension x8120
Conference From x8120	A conference request from another call center user at extension x8120



Note: When the demo adapter receives a call, your computer plays a ring tone. When the demo adapter initiates a call via click-to-dial, your computer plays a dialing sound.

Customizing the Demo Adapter

You can customize the demo adapter system tray menu and the logo that displays at the bottom of the SoftPhone by modifying demo_menu.xml in the installation directory.

• To customize the first system tray menu item, edit the menu item label or automatic number identification (ANI) attributes in the following line:

```
<ITEM ID="MENU_CALL_FROM_PHONE" LABEL="Call From 415-555-1212" ANI="4155551212"/>
```

To customize the second system tray menu item, edit the menu item label, ANI, search field, or search value attributes in the following
line:

```
<ITEM ID="MENU_CALL_FROM_IVR" LABEL="Call Via IVR (Case 1001)" ANI="4155551212"
SEARCHFIELD="Case.CaseNumber" SEARCHVALUE="00001001"/>
```

- Note: For the SEARCHFIELD attribute, any object-field pair can be used. For example, Case.CaseNumber, Account.AccountNumber, and Support Program c.Program Number c are all valid.
- To customize the SoftPhone logo, edit the image URL in the following line:

<LOGO IMAGE_URL="https://yourInstance.salesforce.com/servlet/servlet.ImageServer
?oid=00D000000000062&id=01530000007JX6"/>

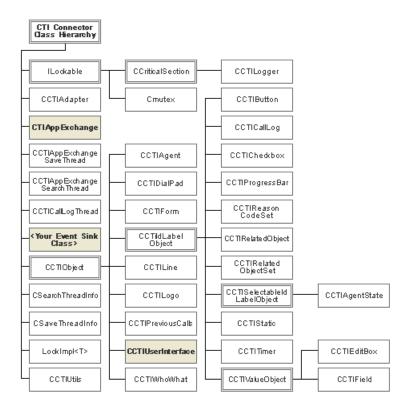


Note: To ensure that SoftPhone logo images are available on any machine, save your logo image as an externally available document in the Documents tab of any Salesforce organization.

CTI Connector Classes

When a developer wants to customize a CTI adapter, he or she modifies its CTI connector component. This diagram shows the classes that make up a CTI connector. Descriptions of each class follow:

The Salesforce CTI Toolkit CTI Connector Classes



ILockable

A container for classes that can lock and release resources.

CCritical Section

A wrapper class for the Win32 CRITICAL_SECTION object. Objects that must be thread-safe should derive from this class.

CCTILogger

A class that logs CTI Adapter events to a file and classifies them by level (Low, Medium and High). Low-level events are used for errors only, medium-level events are used for informational messages, and high-level events are used for granular information such as the specific XML that was sent back and forth in messages.

CMutex

A wrapper class for the Win32 MUTEX object.

CCTIAdapter

An interface between the CCTIUserInterface class and the base COM class of the CTI connector. This class provides a common way for methods in CCTIUserInterface to fire a UIRefresh event.

CCTIAppExchange

An interface between CCTIUserInterface and the Force.com API.

CCTIAppExchangeSaveThread

Encapsulates a thread for saving items to the Salesforce database with the Force.com API. When using this thread, data can be saved in the background without freezing the Salesforce user interface. For example, this class is used to create and update task objects for call logs, and saves user login parameters.

The Salesforce CTI Toolkit CTI Connector Classes

CCTIAppExchangeSearchThread

Encapsulates a thread for searching for items in the Salesforce database with the Force.com API. When using this thread, a search can occur in the background without freezing the Salesforce user interface. For example, this class is used to search for automatic number identifiers (ANIs) and interactive voice response (IVR) data.

CCTICallLogThread

Encapsulates a thread for creating and updating task objects for call logs using the Force.com API. When using this thread, tasks can be created in the background without freezing the Salesforce user interface.

<Your Event Sink Class>

Normally a subclass of a library provided by a CTI system vendor, an event sink captures interesting events broadcast by a CTI system. This class receives events, populates attached data into a PARAM_MAP structure, and calls the corresponding event handlers in CCTIUserInterface.

CCTIObject

A base class for all visual objects in a SoftPhone. It provides common functionality, such as determining when an object is hidden or visible, attribute handling, and serialization to XML.

CCTIAgent

A class that represents the call center state picklist in a SoftPhone. A call center state indicates whether the user is ready to receive calls.

CCTIDialpad

A class that represents the dial pad for a single phone line in a SoftPhone.

CCTIForm

A class that represents a set of related combo boxes, edit boxes, check boxes, buttons, and other user interface elements in a SoftPhone. When this form is rendered, all element values are sent in a group when a user presses a submit button.

CCTIIdLabelObject

A class that represents any CCTIObject that requires a label and ID. At runtime, the browser controller uses an object's ID to translate any text to the proper language for the call center user. If the browser controller does not have a mapping for a particular ID, the label associated with the object is used instead.

CCTIButton

A class that represents a button in a SoftPhone.

CCTICallLog

A class that represents a call log for a current or previous call in a SoftPhone.

CCTICheckbox

A class that represents a checkbox in a SoftPhone.

CCTINoRelatedObjectSet

A class that represents the absence of any related objects for incoming search parameters.

Available in the CTI Toolkit version 2.0 or higher.

CCTIPayload

A class that represents a data passing from a call to a Visualforce page via URL parameters.

Available in the CTI Toolkit version 2.0 or higher.

CCTIPayloadData

A class that represents the name value pairs of payload data passed from a call to a Visualforce page.

Available in the CTI Toolkit version 2.0 or higher.

The Salesforce CTI Toolkit CTI Connector Classes

CCTIProgressBar

A class that represents a progress bar control that indicates processing is occurring. This class can exist within the context of CCTIUserInterface or CCTILine.

CCTIReasonCodeSet

A class that represents a set of reason codes in a SoftPhone. A reason code is an explanation that users can select when they want to enter wrap-up mode after a call, set their user status to Not Ready for Calls, or log out of a CTI system entirely. Organizations can define sets of reason codes to track the activities of their call center users. This class does not distinguish between wrap-up, not ready, and logout reason codes.

CCTIRelatedObject

A class that represents a single related record (like an individual contact or account) in a SoftPhone.

CCTIRelatedObjectSet

A class that represents a set of related records (like contacts or accounts) in a SoftPhone.

CCTISelectableIdLabelObject

A base class that represents any object with an ID, Label, and Selected parameter in a SoftPhone.

CCTIAgentState

A class that represents a single call center state in a SoftPhone, such as Not Ready for Calls.

CCTIStatic

A class that represents a static text element in a SoftPhone.

CCTITimer

A class that represents a timer that ticks in real-time in a SoftPhone. A CCTITimer can only be attached to a phone line.

CCTIValueObject

A class that represents any CCTIIdLabelObject that also requires a value.

CCTIEditBox

A class that represents an edit field in a SoftPhone.

CCTIField

A class that represents a single field from a CTIRelatedObject in a SoftPhone.

CCTILine

A class that represents a single phone line in a SoftPhone.

CCTILogo

A class that represents the logo section of a SoftPhone.

CCTIPreviousCalls

A class that represents a single phone line in a SoftPhone.

CCTIUserInterface

A class that encapsulates the connection between a CTI server and a SoftPhone. All CTI connectors should subclass this class, overriding the command methods to actually perform the described actions. CCTIUserInterface should also include methods for events that are captured in an event sink.

CCTIWhoWhat

A class that holds data about any record that is related to a call log. A call log can be related to one contact, lead, or person account record (the "who"), and one additional record of any other type (the "what").

CSearchThreadInfo

A class that provides information to CCTIAppExchangeSearchThread. This class generally does not need to be instantiated by classes other than CCTIAppExchangeSearchThread.

CSaveThreadInfo

A class that provides information to CCTIAppExchangeSaveThread. This class generally does not need to be instantiated by classes other than CCTIAppExchangeSaveThread.

LockImpl<T>

A resource grabber class that automatically locks a resource on construction and releases the resource on destruction.

CCTIUtils

A utility class for convenience methods, such as conversions to and from COM data types.

For more information about the methods and attributes associated with these classes, see the Salesforce CTI Toolkit Code Reference.

Best Practices for Coding with the CTI Toolkit

The following practices are recommended for coding with the CTI Toolkit.

The CTIConstants.h File

In addition to the .h files that are included for every class, the CTI Toolkit also includes CTIConstants.h. This file contains all of the constant values that are used to control the display of the SoftPhone, as well as constants for all commonly used terms that appear in the CTI Toolkit code.

The 'L' Literal String and Character Prefix

The Salesforce CTI Toolkit includes an 'L' character in front of all literal strings and characters. The 'L' prefix indicates that the string will be stored as an array of "wide" (2-byte) characters, which are necessary to support Unicode character encoding. Salesforce enforces the use of this prefix on all string values to support localization.

For example, in this constant definition, the 'L' is placed in front of the constant's literal string value:

```
#define CTI_CLIENT_KEY L"/cti/1.0/"
```

If an 'L' prefix is not placed in front of a string's first " character, the following type of error appears when you attempt to compile:

```
CTICallLog.cpp(193) : error C2665: 'CCTILogger::Log' : none of the 11 overloads can convert parameter 2 from type 'const char [93]'

c:\dev\144\clients\Salesforce_CTI\CTIAdapterLib\include\CTILogger.h(96): could be 'void CCTILogger::Log(int,const wchar_t *,const wchar_t *,const wchar_t *,const wchar_t *,const wchar_t *)'

c:\dev\144\clients\Salesforce_CTI\CTIAdapterLib\include\CTILogger.h(101):or 'void CCTILogger::Log(int,const wchar_t *,long,const wchar_t *,const wchar_t *)'

c:\dev\144\clients\Salesforce_CTI\CTIAdapterLib\include\CTILogger.h(106): or 'void CCTILogger::Log(int,const wchar_t *,long,long,const wchar_t *)'

c:\dev\144\clients\Salesforce_CTI\CTIAdapterLib\include\CTILogger.h(111):or 'void CCTILogger::Log(int,const wchar_t *,const wchar_t *,long,const wchar_t *)'

c:\dev\144\clients\Salesforce_CTI\CTIAdapterLib\include\CTILogger.h(111):or 'void CCTILogger::Log(int,const wchar_t *,const wchar_t *,long,const wchar_t *)'

c:\dev\144\clients\Salesforce_CTI\CTIAdapterLib\include\CTILogger.h(121): or 'void CCTILogger::Log(int,const wchar_t *,const std::wstring &,const std::wstring &)'
```

```
c:\dev\144\clients\Salesforce_CTI\CTIAdapterLib\include\CTILogger.h(126): or 'void
CCTILogger::Log(int,const wchar_t *,const std::wstring &,const wchar_t *)'

c:\dev\144\clients\Salesforce_CTI\CTIAdapterLib\include\CTILogger.h(141): or 'void
CCTILogger::Log(int,const wchar_t *,long,const std::wstring &)'

c:\dev\144\clients\Salesforce_CTI\CTIAdapterLib\include\CTILogger.h(146): or 'void
CCTILogger::Log(int,const wchar_t *,const std::wstring &,long)'

while trying to match the argument list '(int, const char [93], std::wstring, std::wstring)'
```

Method Name in CCTIUserInterface

This table shows the naming convention for most methods in CCTIUserInterface:

Naming Convention	Description
CallXxx (for example, CallInitiate)	A method that initiates or somehow affects an active call
CreateXxx (for example, CreateParty)	A method that allocates memory for a new object
CTIXxx (for example, CTIChangeAgentState)	A method that affects the overall state of the call center user in the CTI system, but does not pertain to any specific call
DestroyXxx (for example, DestroyVirtualLine)	A method that deallocates memory that was created for a specific object
GetXxx (for example, GetAppExchange)	A method that provides access to a property or sub-object of CCTIUserInteface
OnXxx (for example, OnAgentStateChange)	A CTI event
OnCallXxx (for example, OnCallAttemptFailed)	A CTI event pertaining to a specific call
QueueXxx (for example, QueueChangeAgentState)	A method that implements the specific action after some precondition has completed
SetXxx (for example, SetAllowDialpad ForAllLines)	A method that sets a property on CCTIUserInterface or on one of its sub-objects
UIXxx (for example, UIRefresh)	A method that renders the SoftPhone user interface or handles input from it

Specifying a Valid CTI Client Key

If you are developing a CTI adapter that will be published on Force.com AppExchange, you must provide a valid client key in CTIAppExchange. h. A client key allows an organization to download and deploy your CTI adapter even if the organization does not normally have access to the Force.com API. All CTI adapters require access to the API in order to communicate with the Salesforce database.

- If you are creating a CTI adapter from a third-party vendor's adapter source code, a valid client key is already specified in CTIAppExchange.h. You do not need to provide a new client key.
- If you are creating a CTI adapter directly from a Salesforce CTI Toolkit, request a client key from Salesforce Support and use it to update the following constant in CTIAppExchange.h:

#define CTI_CLIENT_KEY L"<your_client_key_value_here>"

CHAPTER 3 Customizing Salesforce CRM Call Center

While Force.com AppExchange offers prebuilt CTI adapters for several CTI systems, many organizations want to implement additional features for these adapters or use a CTI system for which an adapter has not yet been built. In either case, an organization can customize a CTI adapter by modifying the CTI connector source code provided by Salesforce. You can either customize an existing CTI connector or build a new one for a CTI system that is not yet supported.

Customizing a CTI Connector

Salesforce allows you to customize CTI adapters that have already been built and posted on AppExchange. To do so:

- 1. Download the CTI adapter code package. (See The Salesforce CTI Toolkit on page 9 for information.)
- 2. Read the entirety of this guide to understand how a CTI connector should work with other CTI adapter components, and how it can be customized.
- **3.** Use Visual Studio.NET 2008 to customize the CTI connector code base. For example, you can:
 - Use any CTI system event to update the SoftPhone for a user. For example, a warning icon can blink if customers are forced to be on hold for more than five minutes. (See Writing an Event Sink Handler on page 27 for information.)
 - Define new commands for call center users. For example, you can add a **Record** button that allows a user to record his or her phone conversations. (See Implementing Call Center User Command Messages on page 30 for information.)
 - Customize the buttons that are displayed in a SoftPhone. For example, you can modify the length, icon, and placement of any button. (See Adding a Button on page 37 and Changing the Display Order of SoftPhone Buttons on page 36 for information.)
 - Display custom Salesforce fields in a SoftPhone layout. For example, you can display a field that shows the local time of the person who is calling. (See Modifying Displayed Call Information on page 43 for information.)
 - Customize the call center user login screen. For example, an organization that allows free seating might require their users to enter a desk number when they log in. (See Enabling Call Center User Login on page 32 for information.)
 - Enable one-step (blind) transfers and conference calls. For example, a call center user might wish to transfer a caller to another party without speaking to them first. (See Enabling One-Step Transfers and Conferences on page 35 for information.)
 - Manage wrap-up, not-ready, and logout reason codes. For example, you can ensure that a user specifies why they cannot take a new call before they enter the Not Ready for Calls state. (See Enabling Reason Codes on page 38 for information.)
 - Modify the logo at the bottom of the SoftPhone. For example, you can display the logo for your organization rather than the logo of the CTI system. (See Adding a Custom Logo on page 43 for information.)
 - Customize the search for call-related Salesforce records. For example, you can query other data repositories in your organization for information before searching Salesforce. (See Displaying Call-Related Records on page 44 for information.)
 - Customize automatically-generated call logs. For example, you can add a field to a call's activity record that shows the amount of time a caller was left on hold. (See Customizing Automatically-Generated Call Logs on page 46 for information.)
 - Translate SoftPhone labels. For example, you can display French labels for your call center in Paris and Japanese labels for your center in Tokyo. (See Translating Custom SoftPhone Labels on page 47 for information.)
- **4.** Update the CTI adapter's call center definition file as required. (See Call Center Definition Files on page 48 for information.)
- 5. Compile the CTI connector solution and test it with the Salesforce CRM Call Center SoftPhone connector.
- **6.** Deploy your adapter. (See Packaging and Publishing a CTI Adapter on page 53 for information.)

Building a New CTI Connector

If you want to use Salesforce CRM Call Center with a CTI system that does not currently have a CTI adapter, you must build a new CTI connector. Building a new CTI connector is similar to customizing an existing CTI connector, except that in addition to performing the customizations outlined in the previous section, you must also build an event sink.

To build a new CTI connector:

- 1. Read the entirety of this guide to understand how a CTI connector should work with other CTI adapter components, and how it can be customized.
- 2. Download the CTI connector files from Salesforce. (See The Salesforce CTI Toolkit on page 9 for information.)
- **3.** Use Visual Studio.NET 2008 to build a CTI connector solution. (See Setting Up a New CTI Connector Solution on page 20 for information.)
- **4.** Write an event sink for your new CTI connector to handle events that are generated by your CTI system. (See Writing an Event Sink on page 26 for information.)
- **5.** Write subclasses of CCTIUserInterface and CCTIAppExchange to maintain a representation of a SoftPhone based on CTI system events and user commands. (See SoftPhone Modification Options on page 29 for information.)
- **6.** Update the CTI adapter's call center definition file as required. (See Call Center Definition Files on page 48 for information.)
- 7. Compile the CTI connector solution and test it with the Salesforce CRM Call Center SoftPhone connector.
- **8.** Deploy your adapter. (See Packaging and Publishing a CTI Adapter on page 53 for information.)
- Note: You can enable HTTPS between a call center and CTI adapters. For more information, see Configuring HTTPS for a CTI Adapter on page 54.

Setting Up a New CTI Connector Solution

If you are building a new CTI connector for a CTI system that does not currently have a CTI adapter, you must set up a new project in Visual Studio.NET 2008. If you simply want to customize an existing CTI connector, it is not necessary to perform these steps.

To set up a project for a new CTI connector:

- 1. Build a CTI connector solution in Visual Studio.NET 2008, as described in Building a CTI Connector Solution in Visual Studio on page 20.
- 2. Add the COM base class to your project, as described in Adding a COM Base Class to a CTI Adapter Project on page 22.
- 3. Instantiate a subclass of CCTIUserInterface, as described in Instantiating a CCTIUserInterface Subclass on page 24.
- **4.** Add the browser connector project to the CTI connector solution, as described in Referencing Your New CTI Adapter in BrowserConnector on page 25.
- 5. Optionally, update app.config, as described in Configuring app.config on page 25.

Building a CTI Connector Solution in Visual Studio

If your organization uses a CTI system that does not have a SoftPhone CTI adapter built by Salesforce, you must first build a CTI connector project in Visual Studio.NET 2008:

- 1. Download and install the Salesforce CRM Call Center libraries. See The Salesforce CTI Toolkit on page 9 for details.
- 2. Open the demo CTI connector solution from where it's installed. For example, from the Start menu, select **All Programs** > salesforce.com > CTI Toolkit 4.0 > Demo Adapter Source Code.

- **3.** In Visual Studio, click **File** > **New...** > **Project**.
- **4.** In the Project Types area, select **Other Languages** > **Visual C++ Projects** > **ATL**.
- 5. In the Templates area, select ATL Project.
- 6. Specify a Name and Location for your CTI connector.
- 7. Specify Add to Solution for Solution.
- 8. Click OK.
- **9.** In the ATL Project Wizard, click **Finish**. Two projects are created: one with the name you specified and another with the name you specified and a postfix of "PS" (for example, "MyCTlConnector" and "MyCTlConnectorPS"). The second "PS" project is not necessary and can be removed from your solution.
- **10.** Right-click the project you just created and select **Properties**.

11. Select **Configuration Properties** > **C/C++** > **General**.

- **a.** Set the Configuration picklist to All Configurations and make the following changes:
 - In the Additional Include Directories field, add the CTIAdapterLib include directory (for example, ..\CTIAdapterLib\include).
 - In the Debug Information Format field, specify Program Database for Edit & Continue (/ZI).
 - In the Warning Level field, specify Level 3 (/W3).
 - In the Detect 64-bit Portability Issues, specify Yes (/Wp64).
- **b.** Set the Configuration picklist to Release.
- c. Change the Debug Information Format field to Line Numbers Only (/Zd).

12. Select Configuration Properties > C/C++ > Code Generation.

- **a.** Set the Configuration picklist to All Configurations and make the following changes:
 - In the Enable Minimal Rebuild field, specify Yes (/Gm).
 - In the Basic Runtime Checks field, specify Both (/RTC1, equiv. to /RTCsu).
 - In the Runtime Library field, specify Multithreaded Debug DLL (/MDd).
 - In the Enable Function-Level Linking field, specify Yes (/Gy).
- **b.** Set the Configuration picklist to Release.
- c. Change the Runtime Library field to Multithreaded (/MT).

13. Select Configuration Properties > C/C++ > Language.

- **a.** Set the Configuration picklist to All Configurations.
- **b.** In the Enable Run-Time Type Info field, specify Yes (/GR).

14. Select **Configuration Properties** > **Linker** > **Input**.

- a. Set the Configuration picklist to Debug.
- **b.** In the Additional Dependencies field, include the debug version of the CTIAdapterLib library, CTIAdapterLibD.lib (for example, ...\CTIAdapterLib\Debug\CTIAdapterLibD.lib).
- **c.** Set the Configuration picklist to Release.
- **d.** In the Additional Dependencies field, include the release version of the CTIAdapterLib library, CTIAdapterLib.lib (for example, ..\CTIAdapterLib\Debug\CTIAdapterLib.lib).

- e. Click **OK** to close the Properties window and save your changes.
- 15. In your project, open stdafx.h and add the following lines to the bottom of the file if they are not already there:

```
#include <atlbase.h>
#include <atlcom.h>
#include <atlwin.h>
#include <atltypes.h>
#include <atlctl.h>
#include <atlctl.h>
#include <comutil.h>

using namespace ATL;
```

Adding a COM Base Class to a CTI Adapter Project

Once you have built a CTI adapter project, you must add the base adapter class that implements the provided COM interface:

- 1. Right-click your adapter project and select Add Class.
- 2. Choose the ATL Simple Object template, and click Open.
- 3. Click Names in the left navigation pane. Specify object names according to the following guidelines:
 - Choose a Class and ProgID name that represent the CTI system with which you are integrating.
 - In the Interface field, specify ISalesforceCTIAdapter.
 - Make a note of the value that you choose for ProgID. You will need it again when you define a call center that uses your custom CTI adapter.

To serve as examples in the remainder of the procedure, suppose you specified the following names:

- Short name = MyAdapterBase
- .h file = MyAdapterBase.h
- Class = CMyAdapterBase
- .cpp file = MyAdapterBase.cpp
- Type = MyAdapterBase Class
- Interface = |SalesforceCT|Adapter
- ProgID = MyAdapter.MyAdapter
- **4.** Click **Options** in the left navigation pane.
- **5.** Select the Connection points checkbox, and click **Finish**.
- **6.** By default, the wizard automatically creates a new interface called ISalesforceCTIAdapter. You will need the version of ISalesforceCTIAdapter that has been defined by Salesforce for your new CTI connector. To use Salesforce's version, open the .h file that the wizard generated (MyAdapterBase.h in the example above). In this file:
 - **a.** Delete the entry for __interface ISalesforceCTIAdapter : IDispatch and its corresponding attributes and comments.
 - **b.** Add the following lines to the #include statement section:

```
#include "CTIAdapter.h"
#include "ISalesforceCTIAdapter.h"
```

c. Edit the class so that it inherits from ISalesforceCTIAdapter and CCTIAdapter, as follows:

```
class ATL_NO_VTABLE CMyAdapterBase :
  public ISalesforceCTIAdapter,
  public CCTIAdapter
```

d. Reset a default event linkage that the wizard created. In the example above, the wizard created:

```
__event __interface _IMyAdapterBaseEvents;

Replace that line with:

__event __interface _ISalesforceCTIAdapterEvents;
```

e. Add the following four methods to the public: section of the class header. Your class must implement each:

```
public:
   * Returns the name and author of the adapter (for
   * example "Salesforce.com CTI Adapter For Cisco IPCC
   * Enterprise").
   * @param bsName Contains the return value.
  STDMETHOD(GetAdapterName)(BSTR* bsName);
  /**
   * Returns the version of the adapter, (for example "1.01b").
   ^{\star} @param bsName Contains the return value.
  STDMETHOD(GetAdapterVersion)(BSTR* bsVersion);
  * Receives an inbound XML-formatted message via COM
  * from the browser controller. It should be formatted
  * as (with as many parameters as needed):
  * <MESSAGE ID="MESSAGE ID">
    <PARAMETER NAME="PARAM1" VALUE="VALUE1"/>
    <PARAMETER NAME="PARAM2" VALUE="VALUE2"/>
  * </MESSAGE>
  ^{\star} @param message The XML-formatted message to handle
  * @return An HRESULT indicating whether the message
  * was successfully received and parsed
  */
 STDMETHOD (UIAction) (BSTR message);
 /**
  * A method that takes in an XML string and generates a
  * COM UIRefresh event with it.
  ^{\star} @param xml The XML to include with the event.
 virtual void SendUIRefreshEvent( bstr t xml);
```

- **f.** Save your changes and close the file.
- 7. Open the .cpp file that the wizard generated (MyAdapterBase.cpp in the example above). Add bodies for the following methods:

```
STDMETHODIMP CMyAdapterBase::GetAdapterName(BSTR* bsName)
{
  *bsName = SysAllocString(L"My Sample CTI Adapter");
  return S_OK;
}
STDMETHODIMP CMyAdapterBase::GetAdapterVersion(BSTR* bsName)
{
  *bsName = SysAllocString(L"1.0 Candidate 2");
  return S_OK;
}
STDMETHODIMP CMyAdapterBase::UIAction(BSTR message)
{
  //Do something with the incoming XML message here
  return S_OK;
}
void CMyAdapterBase::SendUIRefreshEvent(_bstr_t xml)
{
  CCTILogger::Log(LOGLEVEL_HIGH, "Sending XML (len %d): %s", xml.length(), (wchar_t*)xml);
  _ISalesforceCTIAdapterEvents_UIRefresh(xml);
}
```

8. Compile the project. You now have a functioning COM object that can send and receive XML.

Instantiating a CCTIUserInterface Subclass

The CTI connector class that provides the link between Salesforce, the end user, and a CTI system is called CCTIUserInterface. This class receives commands sent by the user, interprets events coming from the CTI server, and contains a representation of the SoftPhone user interface.

Every CTI connector must have a subclass of CCTIUserInterface instantiated in its base COM class. The subclass implements the command methods specified in CCTIUserInterface and calls the event methods of the base class when appropriate to update the SoftPhone.

Use the FinalConstruct method of the CTI connector's base class to instantiate a subclass of CCTIUserInterface. For example:

```
HRESULT CMyAdapterBase::FinalConstruct()
{
   m_pUI = new CMyUserInterface(this);
   m_pUI->Initialize();
}
```

Note: Always call the Initialize method after the CCTIUserInterface subclass is instantiated. This allows the SoftPhone to create all of its child user interface objects in advance.

The subclass's constructor passes a pointer to the CTI connector base class and the number of available phone lines to the superclass constructor. For example:

```
CMyUserInterface: CMyUserInterface (CMyAdapterBase* pAdapter) :CCTIUserInterface(pAdapter,2)
```

In this example, CMyUserInterface supports two phone lines. Your CMyUserInterface subclass needs to specify the maximum number of lines that can be used by a single call center user. This allows CCTIUserInterface to create those phone lines in advance.

Use the FinalRelease method of the CTI connector's base class to delete your instance of CCTIUserInterface. This action terminates all related threads, allowing the .dll to be unloaded. For example:

```
void CMyAdapterBase::FinalRelease()
{
  if (m_pUI) {
   delete m_pUI;
  }
  CoUninitialize();
}
```

Referencing Your New CTI Adapter in BrowserConnector

BrowserConnctor generates SalesforceCTI.exe when compiled. Call center users run SalesforceCTI.exe to start their SoftPhones. Once you have added the base adapter class, instantiated a subclass of CCTIUserInterface, and compiled your CTI adapter.dll, you can reference the .dll in your BrowserConnector project:

- 1. In Visual Studio, go to the Solution Explorer and right-click **BrowserConnector**.
- 2. Select Add Reference.
- **3.** Click the Browser tab.
- 4. Navigate to your new CTI adapter .dll file and select it.
- 5. Click OK.
- 6. From the Solution Explorer, click **BrowserConnector** > **AdapterLink** and select AdapterLink.cs.
- 7. Change using DemoAdapter; to using your new CTI connecter.dll name;.
- 8. Change adapter = new CDemoAdapterBaseClass(); to adapter = new
 YourNewCTIConnecterClassName();.
- 9. Click Save.

Configuring app.config

app.config provides an XML file to change two CTI configurations without having to recompile the CTI connector solution. Once you have built a CTI connector solution, you can specify the value of these two optional configurations in app.config:

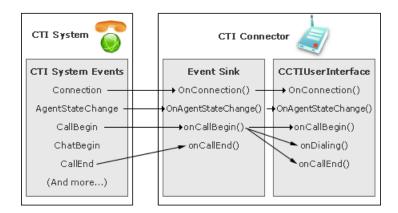
- 1. In Visual Studio, go to the Solution Explorer and click **BrowserConnector** > **Properties** > **Settings**.settings.
- **2.** Update the value of the following fields as necessary:

Configuration	Description
AdapterUrl	The location of where the CTI adapter is hosted. For example, http://localhost:11000.
AdapterExitWaitTime	The time at which users are automatically logged out of their SoftPhones due to SoftPhone inactivity. For example, 10000 (milliseconds).

3. Click Save.

Writing an Event Sink

A CTI connector requires an event sink class that receives events from its associated CTI system and calls the appropriate event methods in the CCTIUserInterface subclass. As illustrated in this diagram, an event sink only requires a handler method for CTI system events that affect the appearance or functionality of a SoftPhone. These handler methods can then call one or more methods in CCTIUserInterface:



Event sink implementations vary for each CTI system depending on the interfaces that a vendor makes available. For each implementation:

- 1. Determine the CTI system events that your event sink must capture. See Determining the CTI System Events that Require an Event Sink Handler on page 26 for information.
- 2. For each CTI system event that must be captured, write an event handler that calls the appropriate methods in your CCTIUserInterface subclass. See Writing an Event Sink Handler on page 27 for sample event sink handler code.

Determining the CTI System Events that Require an Event Sink Handler

In general, any CTI system event that affects a user's SoftPhone requires a handler in your event sink. While events vary from system to system, some common events that need a handler include:

- Events related to a user's CTI server connection
- Events related to a user's state (Ready for Calls, On a Call, Not Ready for Calls, and so on)
- Events related to the types of buttons that should be enabled

• Call events that affect the SoftPhone, such as the start and end of a call, or the fact that a caller has been put on hold

Although the following types of events do not require event sink handlers for default Salesforce CRM Call Center functionality, you may want to add event sink handlers for some of these events if you are implementing a new SoftPhone feature:

- Statistical events that appraise a user of CTI system usage, such as the number of call center users that are currently on a call
- Chat, email, or other channel events that Salesforce CRM Call Center does not currently support
- Informational events that apprise a user of activity on the system, such as the fact that another call center user has just logged in
- Call events that do not affect the SoftPhone, such as the fact that a call has been gueued

Writing an Event Sink Handler

Event handlers in an event sink pass data related to an event directly to a comparable method in CCTIUserInterface. Some event handlers consist of single one-line calls to CCTIUserInterface, while others require deeper levels of processing.

Simple Event Handler: OnCallEnd

The following simple event handler passes an OnCallEnd event directly to the OnCallEnd method in CCTIUserInterface. The first parameter value indicates the ID of the call that is affected, while the second specifies whether the log for this call should be moved to the Previous Calls section of the SoftPhone:

```
void CMyEventSink::OnCallEnd(EventArguments* pArguments)
{
  std::wstring sCallObjectId = pArguments->GetValue("CallObjectID");
  m_pUI->OnCallEnd(sCallObjectId, true);
}
```

Complex Event Handler: OnCallRinging

More complex event handlers perform additional processing and often must pass one or more PARAM_MAP parameters into CCTIUserInterface. A PARAM_MAP is a map of the form std::map<std::wstring,std::wstring>. Elements of a PARAM MAP can be obtained by calling map["paramName"].

For example, OnCallRinging is a CTI system event that occurs when a call has arrived on any phone line. For incoming calls, OnCallRinging marks the start of a phone conversation and maps to the OnCallRinging method in CCTIUserInterface. For outgoing calls, OnCallRinging marks the start of the dialing process and maps to the OnCallDialing method in CCTIUserInterface.

In addition to their other parameters, both the OnCallRinging and OnCallDialing methods in CCTIUserInterface require a PARAM_MAP attribute called mapInfoFields. In addition, OnCallRinging requires a second PARAM_MAP attribute called mapAttachedData:

- mapInfoFields includes information that pertains to the incoming call, such as the automatic number identification (ANI,the number from which the caller is calling), dialed number identification service (DNIS, the number the caller dialed), and any other custom fields that you add. This data displays in the SoftPhone.
- mapAttachedData includes all other data attached to an OnCallRinging event, such as account data generated from an interactive voice response (IVR) system. This data is used for performing SOSL and SOQL searches of Salesforce data and generally is not displayed in the SoftPhone interface.

The following sample code implements the OnCallRinging handler in an event sink. For more information about the parameters in these calls, see the Salesforce CTI Toolkit Code Reference.

```
void CMyEventSink::OnCallRinging(EventArguments* pArguments)
PARAM MAP mapInfoFields;
PARAM MAP mapAttachedData;
 //First convert the attached data to a parameter map so that we
 //can later pass it to OnCallRinging() in CCTIUserInterface.
 //To simplify this example, this conversion is represented by
 //a call to a method called ResolveAttachedData().
 //In practice, salesforce.com adapters handle this with several
 //lines of inline code. See the OnCallRinging() implementation
 //for your adapter for details.
ResolveAttachedData(pArguments, mapAttachedData);
 //Determine the call object ID
 std::wstring sCallObjectId = pArguments->GetValue("CallObjectID");
 //We always perform a search
bool bPerformSearch = true;
 //We don't log internal calls
bool bLogCall = !pArguments->IsInternal();
 //Check whether the call is incoming or outbound
if (pArguments->IsIncoming()) {
 //Assume that the call is from an external number (since those
    //calls do not have an associated call type), but if it is
    //internal, change it to CALLTYPE INTERNAL
 int nCallType = CALLTYPE INBOUND;
 if (pArguments->IsInternal()) nCallType = CALLTYPE INTERNAL;
    //Pull the DNIS and ANI (the number from which the
    //caller is dialing), and put these values in mapInfoFields
 std::wstring sANI = pArguments->GetValue("ANI");
 mapInfoFields[KEY ANI]=sANI;
 std::wstring sDNIS = pArguments->GetValue("DNIS");
 mapInfoFields[KEY DNIS] = sDNIS;
    //Call the OnCallRinging() event handler in CCTIUserInterface
m pUI->OnCallRinging(sCallObjectId,nCallType,bPerformSearch,bLogCall,mapInfoFields,mapAttachedData);
} else {
    //If outbound, call the OnCallDialing() event handler in
   //CCTIUserInterface, and attach the mapInfoFields PARAM MAP.
 m pUI->OnCallDialing(sCallObjectId, mapInfoFields, bPerformSearch, bLogCall);
}
```

SoftPhone Modification Options

You can modify CCTIUserInterface and CCTIAppExchange to customize the appearance and functionality of a SoftPhone. Whether you are building a new CTI connector or customizing one that has already been built by Salesforce, you should read the following topics to understand how the code works and what elements are required to make a SoftPhone operational:

- Using the Virtual Keyword in Your CCTIUserInterface .h File on page 29
- Implementing Call Center User Command Messages on page 30
- Writing the Initialize Method for CCTIUserInterface on page 32
- Enabling Call Center User Login on page 32
- Enabling One-Step Transfers and Conferences on page 35
- Enabling a Set of Buttons on page 35
- Changing the Display Order of SoftPhone Buttons on page 36
- Adding a Button on page 37
- Enabling Reason Codes on page 38
- Mapping CTI System Agent States to Salesforce CRM Call Center User States on page 40
- Adding a Custom Logo on page 43
- Modifying Displayed Call Information on page 43
- Customizing Automatically-Generated Call Logs on page 46
- Translating Custom SoftPhone Labels on page 47

For more detailed information about the objects and methods in the Salesforce CRM Call Center source code that Salesforce provides, see the Salesforce CTI Toolkit Code Reference.

Using the Virtual Keyword in Your CCTIUserInterface .h File

When you write the .h file for your CCTIUserInterface subclass, you must use the virtual keyword for all the methods that you override, other than the constructor. This ensures that CTILib and other parts of the CTI connector call the correct method for your extended objects, regardless of how they are declared.

For example, the interface for a subclass of CCTIUserInterface might include a constructor, destructor, and two additional overridden methods:

```
#pragma once
#include ctiuserinterface.h"

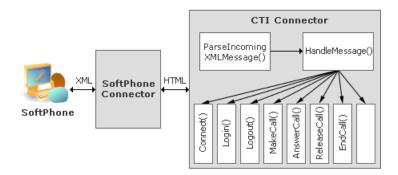
class CMyAdapterBase;
class CMyAdapterUserInterface :
  public CCTIUserInterface
{
  public:
    CMyAdapterUserInterface(CMyAdapterBase* pAdapter);
    virtual ~CMyAdapterUserInterface(void);
    virtual void Initialize();
    virtual void CallInitiate(PARAM_MAP& parameters);
};
```

If the following method were to receive a MyAdapterUserInterface object instead of the CCTIUserInterface object that the method declaration expects, the virtual keyword prevents the code from calling the wrong implementation of the CallInitiate method:

```
void foo(CCTIUserInterface *x, PARAM_MAP parameters)
{
    x->CallInitiate(parameters);
}
```

Implementing Call Center User Command Messages

As illustrated below, a CTI connector receives call center user command messages from the SoftPhone connector. These messages arrive formatted in XML and are received by the UIParseIncomingXMLMessage method of CCTIUserInterface. This method parses the incoming command message into a command ID and a set of parameters, and then passes the data to its UIHandleMessage method. UIHandleMessage routes the command message to the proper handler based on its command ID.



If you need to override UIHandleMessage to add a call to a custom command message, your implementation should handle your custom message first and then make a call to the base UIHandleMessage method. For example, the following UIHandleMessage override handles a custom command message called "EXIT." Once "EXIT" is processed, the base UIHandleMessage method in CCTIUserInterface is called:

```
void CDemoUserInterface::UIHandleMessage(std::wstring& message, PARAM_MAP& parameters)
{
    if (message=="EXIT")
    {
        m_pEventSink->SetShellVisible(false);
    }
    CCTIUserInterface::UIHandleMessage(message,parameters);
}
```

Commands That Require Implementation in a CCTIUserInterface Subclass

At minimum, a subclass of CCTIUserInterface must implement the following command methods:

- virtual void CTIConnect (PARAM_MAP ¶meters);
- virtual void CTIDisconnect (PARAM MAP ¶meters);
- virtual void CTILogin (PARAM MAP ¶meters);

```
    virtual void CTILogout (PARAM MAP &parameters);

    virtual void CallAlternate (PARAM MAP &parameters);

    virtual void CallInitiate (PARAM MAP &parameters);

    virtual void CallRelease (PARAM MAP &parameters);

virtual void CTIChangeAgentState (PARAM MAP &parameters);

    virtual void CallAnswer (PARAM MAP &parameters);

    virtual void CallHold (PARAM MAP &parameters);

    virtual void CallRetrieve (PARAM MAP &parameters);

    virtual void CallSetWrapupCode (PARAM MAP &parameters);

    virtual void CallSaveWrapup();

    virtual void CallInitiateTransfer (PARAM MAP &parameters);

    virtual void CallInitiateConference (PARAM MAP &parameters);

    virtual void CallOneStepTransfer (PARAM MAP &parameters);

    virtual void CallOneStepConference (PARAM MAP &parameters);

    virtual void CallCompleteTransfer (PARAM MAP &parameters);

    virtual void CallCompleteConference (PARAM MAP &parameters);

    virtual void CallAttachData (std::wstring sCallObjectId, PARAM MAP mapAttachedData);

    virtual CCTIForm* CreateLoginForm();
```

Note: You do not need to implement a command method if it is not supported by your CTI system.

In most methods above, a PARAM_MAP is included that contains parameters relevant to the command. A PARAM_MAP is a map of the form std::map<std::wstring,std::wstring>. Elements of a PARAM_MAP can be obtained by calling map["paramName"]. For details on any required values in an individual command's PARAM_MAP, see the CCTIUserInterface documentation in the Salesforce CTI Connector Code Reference.

(1) Important: With the exception of UIHandleMessage, all subclass method implementations must first call their base class method implementation before performing any additional processing. For example:

```
void MyAdapterUserInterface::CallInitiateTransfer(PARAM_MAP& parameters)
{
    CCTIUserInterface::CallInitiateTransfer(parameters);
    <remaining code...>
}
```

In most cases UIHandleMessage should also call its base class, but if it is intercepting a message for special handling, it does not need to be called.

Commands That Do Not Require Implementation in a CCTIUserInterface Subclass

The following commands generally do not require implementation by a subclass of CCTIUserInterface. These commands only affect the SoftPhone and are not intended to have any effect on the underlying CTI system:

```
• UIUpdateSid (PARAM MAP &parameters)
```

- CallUpdateComments (PARAM MAP ¶meters)
- ToggleComments (int nLineNumber)

- UIShowDialpad (int nLineNumber, int nDialpadType, bool bUpdateXML)
- UIHideDialpad (int nLineNumber, int nDialpadType, bool bUpdateXML)

Writing the Initialize Method for CCTIUserInterface

In general, the Initialize method of CCTIUserInterface is responsible for the following tasks:

- Establishing a connection between the CTI system and a user's machine
- Logging the user into the CTI system
- Associating the CTI connector's event sink with the CTI system and user session, where applicable

In addition, Initialize is typically the method in which visual elements of the SoftPhone are defined, including the login form, the call center state area, reason codes, phone lines, the SoftPhone logo, and buttons. Once defined, these objects can then be shown or hidden by the other CTI connector methods as needed.

Enabling Call Center User Login

Two CCTIUserInterface methods are responsible for logging a call center user into a CTI system:

- CreateLoginForm() defines the user interface in which call center users enter their authentication information, such as a
 username and password.
- CTILogin () uses the data that a user provided in a login form to connect to the associated CTI system.

CreateLoginForm() Method

To implement a call center user login form, override the CreateLoginForm method in your CCTIUserInterface subclass. This method returns a CCTIForm, which is a container for the following types of form elements:

Form Element Type	Description
CCTIStatic	A static text element in the user interface, such as a welcome message
CCTIEditBox	A text box in which a call center user can enter a string
CCTICheckbox	A checkbox that a call center user can select and deselect
CCTIButton	A button that a call center user can click

An individual login form element is identified and labeled according to its SID. By default, Salesforce includes SIDs and standardized, translated labels for the following commonly-used login form elements:

- AGENT ID
- USER ID
- LOGIN NAME
- AGENT NAME
- PASSWORD
- DN
- POSITION ID

- QUEUE
- SKILL
- PLACE
- PERIPHERAL ID
- SWITCH
- EXTENSION
- ROUTE POINT
- DOMAIN
- LOGIN* (this SID is reserved for the mandatory "Login." button that is required on all Login forms)

If you require a login form element that does not appear in the list above, use one of the ten custom element SIDs (CUSTOM1, CUSTOM1) that Salesforce also provides, and specify its label with the Setlabel method. It is your responsibility to ensure that a custom login form element has the correct label for every language that your CTI adapter supports. For information, see Translating Custom SoftPhone Labels on page 47.



Warning: A Salesforce-provided SID must be used for every element in your login form to ensure that user login information is properly saved by the application.

Login form elements are displayed in the SoftPhone login form in the same order that they are added to the form. There must be at least one CCTIButton element with a SID of "LOGIN" that performs the actual login function.

For example, the following CreateLoginForm method implementation creates a form with text boxes for Agent ID, Password, Peripheral ID, and MyCustomLoginField. It also includes a Save Values check box and a large green login button:

```
CCTIForm* CMyAdapterUserInterface::CreateLoginForm()
CCTIForm* pForm = new CCTIForm();
std::wstring sAgentId;
std::wstring sPassword;
std::wstring sPeripheralId;
 std::wstring sMyCustomLoginField;
 //Add form elements in the order they should appear.
CCTIEditBox* pAgentId = pForm->AddEditBox("AGENT ID");
pAgentId->SetValue(sAgentId);
CCTIEditBox* pPassword = pForm->AddEditBox("PASSWORD");
pPassword->SetValue(sPassword);
pPassword->SetPassword(true);
CCTIEditBox* pPeripheralId = pForm->AddEditBox("PERIPHERAL ID");
pPeripheralId->SetValue(sPeripheralId);
 //Use one of the custom SIDs for the custom field
CCTIEditBox* pCustom = pForm->AddEditBox("CUSTOM1");
pCustom->SetValue(sMyCustomLoginField);
 //Use the SetLabel() method to specify the custom
 //field's label
 pCustom->SetLabel("My Custom Login Field Label");
```

```
pForm->AddCheckbox("SAVE_VALUES","",true);

CCTIButton* pLogin = pForm->AddButton("LOGIN");
pLogin->SetColor("GREEN");
pLogin->SetLongStyle(true);

return pForm;
}
```

CTILogin() Method

When your CTILogin implementation is called, its parameter map contains the SIDs and values of the visual elements that you specified in the login form. You can use this data to connect directly to the CTI server and perform other user initialization functions.

For example, the following CTILogin method implementation:

- 1. Performs some basic error checking
- 2. Initializes a user object and an event sink for receiving CTI system events
- 3. Makes a call toMyCTISystemLib::CTILogin to finish the user's login

```
void CMyAdapterUserInterface::CTILogin(PARAM MAP& parameters)
// Check to see if we are already connected
if (!GetConnected())
 CCTILogger::Log(LOGLEVEL LOW, "Must be connected first to log in!");
}
 // Check to see if we are already logged in
if (GetLoggedIn())
 CCTILogger::Log(LOGLEVEL LOW, "Login attempted while already logged in.");
 // Create a new call center user object to handle the login
if (m pAgent!=NULL) m pAgent.Release();
m pAgent.CreateInstance("MyCTISystemLib.ComAgent");
// Initialize the event sink
m pEventSink->SetAgentPtr(m pAgent);
// Set the call center user object
m pAgent->SetValue("AgentID", parameters["AGENT ID"]);
m pAgent->SetValue("AgentPassword", parameters["PASSWORD"]);
m pAgent->SetValue("AgentPeripheralId", parameters["PERIPHERAL ID"]);
   // Log in to the CTI system
m pMyCTISystemLib->CTILogin(m pAgent);}
```

Enabling One-Step Transfers and Conferences

In Salesforce CRM Call Center, performing a transfer or conference while on a call can be either a one- or two-step process. In a two-step process:

- 1. The user clicks **Transfer** or **Conference** in the SoftPhone. This action places the first call on hold and displays a dial pad in which the user can enter a phone number.
- 2. The user clicks **Dial** to connect with the new phone number. After speaking with the third party, the user can click **Complete Transfer** or **Complete Conference** to complete the operation.

In a one-step process, the user clicks **Transfer** or **Conference**, enters a number into the dial pad that is displayed, and clicks **One-Step Transfer** or **One-Step Conference** to complete the operation without speaking separately to the third party.



Note: Other common terms for "one-step" transfers or conferences include "blind," "mute," "cold," or "single-step."

Because not all CTI systems allow one-step transfer and conference functionality, CTI connector code disables one-step transfers and conferences by default. If you want to enable this functionality, call the SetOneStepTransferEnabled and SetOneStepConferenceEnabled methods at any time. For example, to call these methods in the Initialize method of your CCTIUserInterface subclass, use code similar to the following:

```
void CMyAdapterUserInterface::Initialize() {
   //First call the Initialize() method of the base class.
   CCTIUserInterface::Initialize();
   ...
   //Then enable one-step transfers and conferences later in
   //the method.
   CCTIUserInterface::SetOneStepTransferEnabled(TRUE);
   CCTIUserInterface::SetOneStepConferenceEnabled(TRUE);
   ...
}
```

When a user clicks **Transfer** or **Conference** in a SoftPhone, the CTI connector code uses the CallInitiateTransfer or CallInitiateConference methods to transfer or conference the call in two steps. When a user clicks **One-Step Transfer** or **One-Step Conference** in a SoftPhone, the CTI connector code uses the CallOneStepTransfer or CallOneStepConference methods to transfer or conference the call in a single step.

If your CTI system does not transmit the automatic number identifier (ANI) of the original call to the third party when a transfer or conference is initiated, you must attach this value to the call in CalloneStepTransfer, CallOneStepConference, CallInitiateTransfer, and CallInitiateConference. To do so, use the key "ANI" to attach the data as shown in this example:

```
CCTILine* pLine = GetLine(nLineNumber);
    CallAttachSingleItem(nLineNumber, "ANI", pLine->GetANI());
```

Attaching the ANI to the call in this manner allows the CTI adapter of the third party to interpret this value as the true ANI for the call.

Enabling a Set of Buttons

By default, a SoftPhone line is created with the following buttons (the ID that identifies each button in the CTI connector code is listed in parentheses):

- Answer (BUTTON ANSWER)
- Reject (BUTTON REJECT)
- Release (BUTTON_RELEASE)

- Hold (BUTTON HOLD)
- Retrieve (BUTTON RETRIEVE)
- Transfer (BUTTON TRANSFER)
- Accept Transfer (BUTTON ACCEPT TRANSFER)
- Complete Transfer (BUTTON COMPLETE TRANSFER)
- Conference (BUTTON CONFERENCE)
- Accept Conference (BUTTON ACCEPT CONFERENCE)
- Complete Conference (BUTTON COMPLETE CONFERENCE)

All buttons are created when a line is initialized and are shown when necessary using the OnButtonEnablementChange event in CCTIUserInterface. To display a set of buttons in the SoftPhone, call OnButtonEnablementChange with the number of the affected line and the list of button IDs that should be shown. The system automatically shows the buttons in the list and hides all other buttons.

For example, the following code displays the **Answer** and **Reject** buttons in line one:

```
std::list<int> listEnabledButtons;

listEnabledButtons.push_back(BUTTON_ANSWER);
listEnabledButtons.push_back(BUTTON_REJECT);

CCTIUserInterface::OnButtonEnablementChange(1, listEnabledButtons)
```

Changing the Display Order of SoftPhone Buttons

SoftPhone buttons are displayed whenever they are enabled through the OnButtonEnablementChange event in CCTIUserInterface. The order in which they are displayed is controlled by the index values they are assigned in CTIConstants.h:

```
#define BUTTON_ANSWER 10
#define BUTTON_REJECT 20
#define BUTTON_RELEASE 30
#define BUTTON_HOLD 40
#define BUTTON_RETRIEVE 50
#define BUTTON_TRANSFER 60
#define BUTTON_ACCEPT_TRANSFER 70
#define BUTTON_COMPLETE_TRANSFER 80
#define BUTTON_COMPLETE_TRANSFER 80
#define BUTTON_CONFERENCE 90
#define BUTTON_ACCEPT_CONFERENCE 100
#define BUTTON_COMPLETE_CONFERENCE 110
```

Buttons that have a lower index value are listed first in the SoftPhone, above buttons with greater index values. For example, if the **Conference**, **Hold**, **Release**, and **Transfer** buttons are enabled during a call, they appear in the following order:

- Release (ID=30)
- Hold (ID=40)
- Transfer (ID=60)
- Conference (ID=90)

Adding a Button

To add a new button to your SoftPhone:

- 1. Override the Initialize method of CCTIUserInterface:
 - a. Call your base Initialize method first.
 - **b.** Create a CCTIButton object with the ID and label that you want your button to have.
 - c. Specify the width of the button with the SetLongStyle method. By passing FALSE into this method, your button will be short and you will be able to display an icon with the label. By passing TRUE into this method, your button will be long and will not have an associated icon.
 - **d.** If you passed TRUE into the SetLongStyle method, specify the color of your button with the SetColor method. Color constant names are specified in CTIObject.h.
 - e. If you passed FALSE into the SetLongStyle method, specify the URL of the icon that should be displayed within the button with the SetIconURL method. Icons should be exactly 46 pixels wide by 17 pixels high.
 - Note: Salesforce recommends storing any button icons in your organization's Documents tab. See Adding a Custom Logo on page 43 for information.
 - f. Make a call to UIAddButtonToAllLines that includes the index of where the new button should be placed in relation to other buttons. For example, if you added a button with an index value of 15, it would appear between the **Answer** and **Reject** buttons, if both of those buttons were visible (see Changing the Display Order of SoftPhone Buttons on page 36).
- 2. Add a method to your CCTIUserInterface subclass that handles the message that is sent to the CTI connector when the new button is clicked.
- 3. Override the UIHandleMessage method of CCTIUserInterface so that it calls the method you specify when the new button is clicked. The SoftPhone connector will pass the button's ID as the command message when this button is clicked.

The following code sample shows an overridden Initialize method that adds a short **Begin Recording** button to the SoftPhone:

```
//Define the index value for the new button. A value of 15 places
//the new button just after the Answer button, or at the top of the
//button list if the Answer button is not visible. Note that default
//button indexes are stored in CTIObject.h.
#define BEGIN RECORDING 15
void CMyAdapterUserInterface::Initialize() {
 //First call the Initialize() method of the base class.
CCTIUserInterface::Initialize();
 //Then add the new button. Note that its ID, BEGIN RECORDING, is
 //the command message that will be passed to the UIHandleMessage
 //method whenever this button is clicked.
 CCTIButton pRecordButton("BEGIN_RECORDING", "Begin Recording");
 //Set the size and icon of the button.
 pRecordButton.SetLongStyle(FALSE);
precordButton.SetIcanlRl("https://yourInstance.salesforce.com/servlet/servlet.ImageServer?oid=00Dx0000000000Ps&id=015x00000000gs&")
 //Add a copy of this button to all phone lines, in the location specified by
```

```
//the BEGIN_RECORDING constant.
UIAddButtonToAllLines(BEGIN_RECORDING, &pRecordButton);
}
```

Enabling Reason Codes

A reason code is an explanation that users can select when they want to enter wrap-up mode after a call, set their user status to Not Ready for Calls, or log out of a CTI system entirely. Organizations can define sets of reason codes to track the activities of their call center users.

Enabling Wrap-Up Reason Codes

Wrap-up mode allows a call center user to complete work related to a call, such as filling out a call log, without interruption from a new inbound call. When wrap-up mode is available, some organizations require their users to enter reason codes before entering this mode so that their time can be more accurately tracked.

To enable wrap-up codes for your CTI SoftPhone adapter, you must initialize them either in the Initialization method of CCTIUserInterface or when receiving an appropriate event. To initialize wrap-up codes in one of these methods:

- 1. Convert the codes to a PARAM_MAP in which the keys are the wrap-up code IDs and the values are the wrap-up code labels. A PARAM_MAP is a map of the form std::map<std::wstring,std::wstring>. Elements of a PARAM_MAP can be obtained by calling map ["paramName"].
- 2. Call the SetWrapupReasonCodes method in CCTIUserInterface with the PARAM_MAP as its only parameter. If this method is called multiple times, the existing wrap-up codes are replaced by the newly-specified ones.

Once enabled, wrap-up codes are automatically displayed in the SoftPhone when a call center user's state is set to Wrap-Up.

Enabling Not-Ready Reason Codes

Not-ready reason codes allow an organization to track the reasons why a user cannot take a new inbound call. To enable not-ready reason codes:

- 1. In the CCTIUserInterface::Initialize method or just after user login, make a call to CCTIUserInterface::SetNotReadyReasonRequired. When this method is set to TRUE, a user is prompted to enter a reason code when he or she changes state to Not Ready for Calls.
- **2.** Specify the reason codes for changing to the Not Ready for Calls state:
 - a. Convert the not-ready reason codes to a PARAM_MAP in which the keys are the reason code IDs and the values are the reason code labels. A PARAM_MAP is a map of the form std::map<std::wstring, std::wstring>. Elements of a PARAM MAP can be obtained by calling map["paramName"].
 - **b.** Call CCTIUserInterface::SetNotReadyReasonCodes with the PARAM_MAP as its only parameter. If this method is called multiple times, the existing codes are replaced by the newly-specified ones.

Enabling Logout Reason Codes

Similarly to wrap-up and not-ready reason codes, logout codes allow an organization to track the reasons why a call center user is logging out of the CTI system. To enable logout reason codes:

1. In the CCTIUserInterface::Initialize method or just after user login, make a call to CCTIUserInterface::SetLogoutReasonRequired for logout reason codes. When this method is set to TRUE, a user is prompted to enter a reason code when he or she attempts to log out.

- 2. Specify the reason codes for logout:
 - a. Convert the logout reason codes to a PARAM_MAP in which the keys are the reason code IDs and the values are the reason code labels. A PARAM_MAP is a map of the form std::map<std::wstring,std::wstring>. Elements of a PARAM MAP can be obtained by calling map["paramName"].
 - **b.** Call CCTIUserInterface::SetLogoutReasonCodes with the PARAM_MAP as its only parameter. If this method is called multiple times, the existing codes are replaced by the newly-specified ones.

Example: Enabling Wrap-Up, Not-Ready, and Logout Reason Codes

The following example shows an event sink method that might be called just after a user logs in to a CTI system. The attached data (pArguments) specifies whether wrap-up, not-ready, and logout reasons are required, and if so, what the valid reason codes are:

```
void MyAdapterEventSink::OnJustLoggedIn(EventArguments* pArguments)
   PARAM MAP mapReasonCodes;
   // First extract the wrap-up codes from pArguments if they
   // exist, and put them in the mapReasonCodes PARAM MAP.
  EventArguments* pReasonCodes=pArguments->GetValueArray("IncomingWrapupStrings");
   if (pReasonCodes!=NULL) {
      for (int i=0;i<=pReasonCodes->NumElements();i++) {
         std::wstring sId = pReasonCodes->GetElementKey(i);
         std::wstring sLabel = pReasonCodes->GetElement(i);
        mapReasonCodes[sId]=sLabel;
      // Make the call to SetNotReadyReasonCodes with the
      // mapReasonCodes PARAM MAP.
 m pUI->SetWrapupReasonCodes(mapReasonCodes);
   // Next, extract the not-ready codes from pArguments if
   // they exist, and put them in the mapReasonCodes PARAM MAP.
   EventArguments* pNRReasonCodes=pArguments->GetValueArray("NotReadyReasonCodes");
   if (pNRReasonCodes!=NULL) {
     mapReasonCodes.clear();
      for (int i=1;i<=pNRReasonCodes->NumElements();i++) {
         std::wstring sId = pNRReasonCodes->GetElementKey(i);
         std::wstring sLabel = pNRReasonCodes->GetElement(i);
        mapReasonCodes[sId]=sLabel;
      // Make the call to SetNotReadyReasonCodes with the
      // mapReasonCodes PARAM MAP.
      m pUI->SetNotReadyReasonCodes(mapReasonCodes);
   // Finally, extract the logout codes from pArguments if
 // they exist, and put them in the mapReasonCodes PARAM_MAP.
  EventArguments* pLogoutReasonCodes=pArguments->GetValueArray("LogoutReasonCodes");
   if (pLogoutReasonCodes!=NULL) {
```

```
mapReasonCodes.clear();
for (int i=1;i<=pLogoutReasonCodes->NumElements();i++) {
    std::wstring sId = pLogoutReasonCodes->GetElementKey(i);
    std::wstring sLabel = pLogoutReasonCodes->GetElement(i);
    mapReasonCodes[sId]=sLabel;
}
// Make the call to SetLogoutReasonCodes with the
// mapReasonCodes PARAM_MAP.
m_pUI->SetLogoutReasonCodes (mapReasonCodes);
}
```

Mapping CTI System Agent States to Salesforce CRM Call Center User States

By default, a Salesforce CTI adapter allows call center users to be in one of the following states while using Salesforce CRM Call Center (the call center state ID that is used in the CTI connector code is also listed):

Call Center State	Call Center State ID	Description
Log Out	AGENTSTATE_LOGOUT	The user is not logged in to a CTI system.
Ready for Calls	AGENTSTATE_READY	The user is not currently on a call, and is prepared to accept the next inbound call.
Not Ready for Calls	AGENTSTATE_NOT_READY	The user is not currently on a call, and is not prepared to receive the next inbound call.
On a Call	AGENTSTATE_BUSY	The user is currently connected to a caller.
Wrap-Up	AGENTSTATE_WRAPUP	The user is currently connected to a caller, and wishes to go directly to the Not Ready for Calls state when the call is complete.
Logged In	AGENTSTATE_LOGGED_IN	The user is logged into the system. Note that this state is meant to be used in those CTI systems that do not support call center user states.

All CTI system agent states must be mapped to one of the Salesforce CRM Call Center states for the SoftPhone to behave appropriately. For example, a CTI system might have a state for talking on the phone and another state for waiting on hold. In both cases, these states should be mapped to the Salesforce CRM Call Center Busy state so that the SoftPhone does not display functionality that should not be available when a user is on a call.

To map CTI system agent states to Salesforce CRM Call Center user states, modify the AgentStateToString method in your CCTIUserInterface subclass. AgentStateToString is used by the OnAgentStateChange event handler to map your CTI system agent state values to the correctSalesforce CRM Call Center user state. For example:

```
std::wstring MyAdapterUserInterface::AgentStateToString(enumMY_AgentState State)
{
    switch( State )
    {
        case Logout:
```

```
return AGENTSTATE LOGOUT;
      case NotReady:
      case Login:
        return AGENTSTATE NOT READY;
      case Available:
       return AGENTSTATE READY;
     case BusyOther:
     case Talking:
      case Reserved:
     case Hold:
     case Unknown:
       return AGENTSTATE BUSY;
      case WorkNotReady:
      case WorkReady:
        return AGENTSTATE WRAPUP;
      default:
        return "";
}
```

Displaying Salesforce CRM Call Center User States

All call center states are created when a line is initialized and are selectively displayed in the SoftPhone's call center state drop-down list using the OnAgentStateEnablementChange event in CCTIUserInterface. To change the set of call center states that are available in the SoftPhone at any one time, call OnButtonEnablementChange with the list of call center states that should be available and the ID of the currently selected state. The SoftPhone automatically updates the call center state drop-down list.

For example, the following event sink method creates a list of enabled agent states depending on the current agent state and then makes a call to OnAgentStateEnablementChange:

```
void CCTIOSEventSink::SetAgentStateEnablement(std::wstring& sAgentState) {
 std::list<std::wstring> listEnabledAgentStates;
if (sAgentState == AGENTSTATE LOGOUT) {
 // do nothing
} else if (sAgentState == AGENTSTATE NOT READY || sAgentState == AGENTSTATE READY) {
 listEnabledAgentStates.push back(AGENTSTATE READY);
 listEnabledAgentStates.push back(AGENTSTATE NOT READY);
 listEnabledAgentStates.push back(AGENTSTATE LOGOUT);
} else if (sAgentState == AGENTSTATE BUSY) {
 // if an agent is busy (on a call), show busy status and wrapup (allow agent to queue
wrapup)
 listEnabledAgentStates.push back(AGENTSTATE BUSY);
 listEnabledAgentStates.push back(AGENTSTATE WRAPUP);
} else if (sAgentState == AGENTSTATE WRAPUP) {
 listEnabledAgentStates.push back(AGENTSTATE WRAPUP);
 } else if (sAgentState == AGENTSTATE LOGGED IN) {
 // do nothing
m pUI->OnAgentStateEnablementChange(listEnabledAgentStates, sAgentState);
```

Adding a New Salesforce CRM Call Center User State

To add a new call center state to your SoftPhone:

- 1. Override the Initialize method of CCTIUserInterface:
 - a. Call your base Initialize () method first.
 - **b.** Make a call to UIAddAgentState () with the ID, display order, and label for your new call center state.

Display order is specified as an integer with the lowest number displaying first in the list. The display order values for default call center states are:

AGENTSTATE_READY	0
AGENTSTATE_NOT_READY	1
AGENTSTATE_WRAPUP	2
AGENTSTATE_BUSY	3
AGENTSTATE_LOGOUT	4

If two or more call center states are assigned the same display order number, the state that was most recently defined appears first. For example, if you add a new call center state with display order "2," the new call center state is displayed between AGENTSTATE_NOT_READY and AGENTSTATE_WRAPUP, since the default call center states were already defined in the base Initialize () method.

- 2. Override the CTIChangeAgentState method of CCTIUserInterface so that the SoftPhone displays the correct behavior when your new call center state is selected.
- 3. Override the CTIIsOccupiedAgentState method of CCTIUserInterface so that the method returns the appropriate boolean value when your new call center state is selected.

On Agent State Change uses CTIIs Occupied Agent State to determine whether it is time to move an open call log to the Previous Calls section of the Soft Phone:

- When CTIIsOccupiedAgentState returns True, the call log remains open for editing.
- When CTIIsOccupiedAgentState returns False, the call log is saved and moved to the Previous Calls section.

By default, the two states that return True are AGENTSTATE_BUSY ("On a Call") and AGENTSTATE_WRAPUP. If your new call center state indicates that a phone line is engaged, then it should also return True when passed in to this method.

The following code sample shows an overridden Initialize method that adds a "Processing Account" call center state:

```
void CMyAdapterUserInterface::Initialize() {
   //First call the Initialize() method of the base class.
   CCTIUserInterface::Initialize();

   //Then add the new call center state. Note that its ID, PROCESSING_ACCOUNT, is
   //the command message that will be passed to the CTIChangeAgentState()
   //method whenever this call center state is selected.
   UIAddAgentState(L"PROCESSING_ACCOUNT",5,L"Processing Account");
}
```

Adding a Custom Logo

You can customize the logo that appears at the bottom of a SoftPhone by calling SetLogoImageUrl from an overridden Initialize method in your CCTIUserInterface subclass. For example:

Logos must be no more than 116 pixels wide by 31 pixels high and must be stored on a server that is accessible to all Salesforce CRM Call Center users. Salesforce recommends that you store the image as an externally available document in the Documents tab for your organization. For information on how to create an externally available document, see "Uploading and Replacing Documents" in the Salesforce online help.

To determine the URL of a logo that is stored in the Documents tab:

- 1. View the document detail page in Salesforce.
- 2. Right-click on the logo image and select **Properties**. The logo URL is located next to the Address (URL) or Location parameter in the dialog. It should be of the form

https://yourInstance.salesforce.com/servlet/servlet.ImageServer?oid=00Dx000000000Ps&id=015x000000000qsK

Modifying Displayed Call Information

By default, two types of call-related information are displayed in the SoftPhone when a call begins:

- Call information fields show information about the call itself, such as the number from which the caller is dialing (the automatic
 number identification or ANI) and the number that was dialed (the dialed number identification service or DNIS). These fields are
 specified in the &mapInfoFields parameter of CCTIUserInterface::OnCallRinging. For information about
 modifying these fields, see Adding a Call Information Field on page 43.
- Call-related records show Salesforce records that are related to the call, such as contacts, leads, activities, or accounts. These records
 are discovered by running the CTIAppExchange::Search method on the ANI and any data in the &mapAttachedData
 parameter from CCTIUserInterface::OnCallRinging. For more information, see Displaying Call-Related Records on
 page 44.

Adding a Call Information Field

To add a custom call information field to a SoftPhone, simply add the field to the mapInfoFields PARAM_MAP in CCTIUserInterface::OnCallRinging. For example, the following implementation of OnCallRinging adds fields for the number from which the caller is dialing (the automatic number identification or ANI), and the number that was dialed (the dialed number identification service or DNIS). You can use parguments to add any other field that is passed in from your CTI system:

```
void CMyEventSink::OnCallRinging(EventArguments* pArguments)
{
   PARAM_MAP mapInfoFields;
   PARAM_MAP mapAttachedData;
```

```
//First convert the attached data to a parameter map so that we
 //can later pass it to OnCallRinging() in CCTIUserInterface.
 //To simplify this example, this conversion is represented by
 //a call to a method called ResolveAttachedData().
 //In practice, salesforce.com adapters handle this with several
 //lines of inline code. See the OnCallRinging() implementation
 //for your adapter for details.
ResolveAttachedData(pArguments, mapAttachedData);
 //Determine the call object ID
 std::wstring sCallObjectId = pArguments->GetValue("CallObjectID");
 //We always perform a search
bool bPerformSearch = true;
 //We don't log internal calls
bool bLogCall = !pArguments->IsInternal();
 //Check whether the call is incoming or outbound
if (pArguments->IsIncoming()) {
 //Assume that the call is from an external number (since those
    //calls do not have an associated call type), but if it is
    //internal, change it to CALLTYPE INTERNAL
 int nCallType = CALLTYPE INBOUND;
 if (pArguments->IsInternal()) nCallType = CALLTYPE INTERNAL;
    //Pull the DNIS and ANI (the number from which the
    //caller is dialing), and put these values in mapInfoFields
 std::wstring sANI = pArguments->GetValue("ANI");
 mapInfoFields[KEY ANI]=sANI;
 std::wstring sDNIS = pArguments->GetValue("DNIS");
 mapInfoFields[KEY DNIS]=sDNIS;
    //Call the OnCallRinging() event handler in CCTIUserInterface
m pUI->OnCallRinging(sCallObjectId,nCallType,bPerformSearch,bLogCall,mapInfoFields,mapAttachedData);
} else {
   //If outbound, call the OnCallDialing() event handler in
   //CCTIUserInterface, and attach the mapInfoFields PARAM MAP.
 m pUI->OnCallDialing(sCallObjectId, mapInfoFields, bPerformSearch, bLogCall);
}
```

Displaying Call-Related Records

When a call arrives, the CTI connector searches Salesforce for any related records using the Force.com API. The CTI connector first searches based on data attached to the call, such as an account number that was entered by the caller during an interactive voice response (IVR) session. If that search fails, the CTI connector then searches based on the automatic number identification (ANI, the phone number from

which the caller is dialing). Any related records that are found during either of these searches are displayed in the user interface of the line containing the call.

Handling Attached Data

A CTI connector built by Salesforce implements a default method of handling data attached to calls. If you are writing a new IVR script, you can either follow this default method to format attached data or modify the CTI connector code to handle the attached data in the format your existing script uses.

The default Salesforce method expects attached data in the form of key-value pairs:

- The keys should be of the form Object.FieldName, where Object corresponds to the developer name of an object in the Force.com API, and FieldName corresponds to the developer name of a field on that object. For example, Case.CaseNumber would be a valid key.
- The values should correspond to the format of the field in Salesforce. For example, for Case. CaseNumber, "00001001" would be a valid value.



Note: Values must be exact matches for the Salesforce record. For example, if "1001" is specified but the case number is actually "00001001," the record is not returned.

If your IVR attaches data that is not in this format, but you wish to trigger a search using the attached data, you can resolve it in your implementation of OnCallRinging. In this method you can convert the data into a format that Salesforce CRM Call Center can use. OnCallRinging should return a PARAM MAP that contains valid key-value pairs.

Examples of Attached Data Search Queries

Salesforce CRM Call Center uses attached data to generate a Salesforce Obejct Query Language (SOQL) query. For example, if the attached data contains Case. CaseNumber="00001001", Salesforce CRM Call Center generates and executes the following query:

```
SELECT <fields> FROM Case WHERE CaseNumber='00001001'
```

where the value of <fields> is defined by the user's SoftPhone layout.

If the attached data contains two or more key-value pairs that pertain to the same object, Salesforce CRM Call Center generates a single query with all the conditions specified for that object. For example, if the attached data contains Case.CaseNumber="00001001" and Case.Priority="High", the search generates and executes the following query:

```
SELECT <fields> FROM Case WHERE CaseNumber='00001001' AND Priority='High'
```

If the attached data includes two or more key-value pairs that pertain to different objects, Salesforce CRM Call Center generates a query for each object and adds all of the results to the phone line's display. For example, if the attached data contains

Case.CaseNumber="00001001" and Account.AccountNumber="1234", the search generates and executes the following queries:

```
SELECT <fields> FROM Case WHERE CaseNumber='00001001'
SELECT <fields> FROM Account WHERE AccountNumber='1234'
```

Handling Automatic Number Identification (ANI) Search

If the search on attached data produces no results or if no attached data was found, Salesforce CRM Call Center performs a Salesforce Object Search Language (SOSL) search on the automatic number identification (ANI) field (the number from which a caller is dialing). The search is only performed on objects that are defined in the user's SoftPhone layout, and only returns the fields that the layout specifies.

For example, if the ANI is 650-555-1212 and the user's SoftPhone layout specifies that only the First Name, Last Name, and Email fields of contact records should be displayed, Salesforce CRM Call Center generates and executes the following SOSL query:

FIND {6505551212} IN PHONE FIELDS RETURNING Contact (Email, FirstName, LastName)



Note: ANIs should be stripped of all non-standard digits. For example, if your phone system generates ANIs preceded by a "1," like 1 415 555-1212, the first "1" should be stripped off the number for the search to return successfully.

Overriding Default Search Behavior

To override the default search behavior and implement your own query generators:

- 1. Create a subclass of CTIAppExchange.
- 2. Override the method CCTIUserInterface::GetSearch to return an instance of your Force.com search object. All calls to search will now use your search instead of the default. See the Salesforce CTI Toolkit Code Reference for details.
- 3. Override the method CCTIAppExchange::Search to implement your own query generators.

Customizing Automatically-Generated Call Logs

By default, the <code>UpsertCallLog</code> method of <code>CCTIAppExchange</code> automatically generates a closed task record for every completed SoftPhone call. This call log task record includes the following information about the call:

- The call object ID, a field that uniquely identifies a single call in Salesforce
- The date that the call occurred
- The duration of the call, in seconds
- The type of call that took place (inbound, outbound, or internal)
- The owner of the call record (typically the user who initiated an outbound or internal call, or the user who received an inbound call)
- The text, if any, that was entered in the SoftPhone comments text box

In addition, the call log task record can also include one link to an account, one link to either a contact, lead, or person account (a "who"), and one link to any other Salesforce record that is not a contact, lead, or person account (a "what").

Associating Call Logs with Multiple Records

Because there is a restriction on the number of records that can be associated with a task, AddCallLog must choose how to handle calls with multiple account, "who," or "what" records. For example, a support user might need to create two cases ("what" objects) while on a call because a customer has two separate problems. Since only one case can be associated with a single call log, AddCallLog must choose which case to associate. By default, AddCallLog handles this type of situation as follows:

- **Multiple accounts**: AddCallLog chooses one account record to associate with the call log, and nothing is associated with the other account objects.
- Multiple "who"s: AddCallLog chooses one "who" record to associate with the call log, and nothing is associated with the other "who" records.
- Multiple "what"s: AddCallLog does not associate the call log with any of the "what" records.

You can customize the strategy for handling multiple records by modifying the AddCallLog method in your own CTIAppExchange subclass. One strategy to consider is to make duplicate call log tasks for multiple records of a particular type. In the example above, AddCallLog would make two identical call log task records - one for each case that was generated during the call. When using this strategy, Salesforce CRM Call Center reports must be modified to group call logs by the Call Object ID field so that duplicate

call logs are not counted as two separate calls. In addition, reports that sum call duration must be filtered so that the same call is not included twice.

Defining Custom SoftPhone Labels

You can define custom labels for call information fields with the SetInfoFieldLabel() method of CCTIUserInterface. To do so, modify the Initialize() method of your CCTIUserInterface subclass with a call to SetInfoFieldLabel() for every label that you want to specify. SetInfoFieldLabel() takes two strings as parameters: the sld of the field that you want to rename, and the custom label.

For example, to change the label for the QUEUE info field to "Customer List," add the following line to your Initialize () method:

```
SetInfoFieldLabel("QUEUE", "Customer List");
```

When a field with the sId of "QUEUE" is placed in the mapInfoFields PARAM_MAP in CCTIUserInterface::OnCallRinging, the field is displayed in the SoftPhone with a label of "Customer List."

This table lists the default labels for call information fields that are built into Salesforce CTI adapters:

Call Information Field sld	Default Label
ANI	Caller ID #
DNIS	Dialed #
Queue	Queue
Segment	Segment
Customer Type	Customer Type

You can modify the labels for these default fields, or define labels for custom call information fields that you add to the mapInfoFields PARAM MAP of your CCTIUserInterface::OnCallRinging method implementation.

Translating Custom SoftPhone Labels

By default, all standard SoftPhone labels are translated and displayed in the language of the call center user. If you need to support localized labels for a custom SoftPhone component or call information fields, you can use one of two strategies:

- Create a different CTI adapter for each locale that you want to support

 With this strategy, the labels are hard-coded in the supported language of the adapter. While easy to implement, this method can lead to maintenance issues if you need to support adapters for several different languages.
- Create a single CTI adapter that supports all languages

With this strategy, the adapter performs a search for the user's language and then adjust the label text accordingly. This method can take longer to implement, but simplifies maintenance if you need to support adapters for several different languages.

CHAPTER 4 Call Center Definition Files

A call center definition file specifies a set of fields and values that are used to define a call center in Salesforce for a particular CTI system. Salesforce uses call center definition files in order to support the integration with multiple CTI system vendors.

By default, any CTI adapter installation package includes a default call center definition file that works specifically with that adapter. This XML file is located in the adapter installation directory and is named after the CTI system that it supports. For example, the Cisco IPCC Enterprise™ adapter's default call center definition file is named CiscoIPCCEnterprise7x.xml.

The first instance of a call center for a particular CTI adapter must be defined by importing the call center definition file into Salesforce. Subsequent call centers can be created by cloning the original call center that was created with the import.

If your organization modifies an adapter or builds a new one, you must customize the adapter's call center definition file so that it includes any additional call center information that is required. For example, if you are building a CTI adapter for a system that supports a backup server, your call center definition file should include fields for the backup server's IP address and port number. CTI adapters for systems that do not make use of a backup server do not need those fields in their associated call center definition files.



Note: Once a call center definition file has been imported into Salesforce, the set of fields that were specified in the file cannot be modified. The values assigned to those fields, however, can be changed within Salesforce.

If you have built a custom CTI adapter, you must write a call center definition file to support it. Use a text or XML editor to define an XML file according to the guidelines in the following topics.

Call Center Definition File XML Format

A call center definition file consists of three XML elements: callCenter, section, and item. The following list provides details about the properties and attributes of each element:

callCenter

This element represents a definition for a single call center phone system. At least one <callCenter> element must be included in every call center definition file. A <callCenter> element consists of one or more <section> elements.

section

This element represents a grouping of related data fields, such as server information or dialing prefixes. When a call center is edited in Salesforce, fields are organized by the section to which they are assigned. A <section> element belongs to a single <callCenter> element, and consists of one or more <item> elements.

Attributes:

Name	Туре	Required?	Description
sortOrder	Positive Integer	Required	The order in which the section should appear when the call center is edited in Salesforce. For example, a section with sortOrder="1" comes just before a section with sortOrder="2".
			The values for sortOrder must be non-negative integers, and no numbers can be skipped within a single call center definition. For example, if there are three section elements in a call center definition file, one <section> element must have sortOrder="0", one <section> element must have</section></section>

Name	Туре	Required?	Description	
			$\verb sortOrder="1", and one < \verb section> element must have \\ \verb sortOrder="2". $	
name	String	Required	The internal name of the section as defined in the Salesforce database. You can use this value to refer to the section when writing custom adapter or SoftPhone code.	
			Names must be composed of only alphanumeric characters with no white space or other punctuation. They are limited to 40 characters each.	
			Names beginning with req are reserved for required Salesforce sections only (see Required Call Center Elements and Attributes). Other reserved words that cannot be used for the name attribute include label, sortOrder, internalNameLabel, and displayNameLabel.	
label	String	Optional	The name of the section when viewed in Salesforce. Labels can be composed of any string of UTF-8 characters. They are limited to 1000 characters each.	

item

This element represents a single field in a call center definition, such as the IP address of a primary server or the dialing prefix for international calls. When call centers are edited in Salesforce, each <item> element is listed under the section to which it belongs. You can have multiple <item> elements in a <section> element.

Attributes:

Name	Туре	Required?	Description
sortOrder	Positive Integer	Required	The order in which the item should appear when the call center is edited in Salesforce. For example, an item with sortOrder="1" comes just before an item with sortOrder="2".
			The values for sortOrder must be non-negative integers, and no numbers can be skipped within a single call center definition. For example, if there are three item elements in a call center definition file, one <item> element must have sortOrder="0", one <item> element must have sortOrder="1", and one <item> element must have sortOrder="2".</item></item></item>
name	String	Required	The internal name of the item as defined in the Salesforce database. You can use this value to refer to the item when writing custom adapter or SoftPhone code.
			Names must be composed of only alphanumeric characters with no white space or other punctuation. They are limited to 40 characters each.

Name	Туре	Required?	Description	
			Names beginning with req are reserved for required Salesforce sections only (see Required Call Center Elements and Attributes). Other reserved words that cannot be used for the name attribute include label, sortOrder, internalNameLabel, and displayNameLabel.	
label	String	Optional	The name of the item when viewed in Salesforce. Labels can be composed of any string of UTF-8 characters. They are limited to 1,000 characters each.	

Required Call Center Elements and Attributes

There must be one <section> that includes <item> elements with the following names in every call center definition file:

<item> Name</item>	Description	
reqInternalName	Represents the unique identifier for the call center in the database. It must have a sortOrder value of 0, and its value must be specified in the call center definition (see Specifying Values for <item> Elements). A value for reqInternalName must be composed of no more than 40 alphanumeric characters with no white space or other punctuation. It must start with an alphabetic character and must be unique from the reqInternalName of all other call centers defined in your organization.</item>	
reqDisplayName	Represents the name of the call center as displayed in Salesforce. It must have a sortOrder value of 1. A value for reqDisplayName has a maximum length of 1,000 UTF-8 characters.	
reqDescription	Represents a description of the call center. A value for ${\tt reqDescription}$ has a maximum length of 1,000 UTF-8 characters.	
reqProgld	Represents the Program ID (progld) of the CTI adapter that should be used for this call center. This value is specified in the default call center definition file that comes bundled with every CTI adapter installer, or in the base COM class of a custom CTI adapter.	
reqVersion	Represents the version of the CTI Toolkit with which the adapter was built. This elemer is available for call centers built with CTI Toolkit versions 3.0 or higher.	
reqAdapterUrl	Represents the location of where the CTI adapter is hosted. For example, http://localhost:11000. This element is available for call centers built with CTI Toolkit versions 4.0 or higher.	

Specifying Values for <item> Elements

With the exception of the reqInternalName <item>, whose value must always be specified in a call center definition file, you can specify <item> values either in the call center definition file or in Salesforce once the definition file has been imported.

To specify a value for an <item> element in a call center definition file, place the value between the opening and closing tags of the <item>. For example:

```
<item sortOrder="0" name="reqInternalName" label="Call Center Internal
Label">MyCallCenter</item>
```

sets the value of the reqInternalName <item> to MyCallCenter. Note that any <item> value other than the value for reqInternalName can be edited in Salesforce after the call center definition is imported.

Sample Call Center Definition File

The following XML code makes up a sample call center definition file:

```
<!--
     All sections and items whose name value begins with "req" are
     required in a valid call center definition file. The sortOrder
     and label attributes can be changed for all required sections
     and items except regGeneralInfo, regInternalName, and
     reqDisplayName, in which only the label attribute can be altered.
  Note that the value for the reqInternalName item is limited to
     40 alphanumeric characters and must start with an alphabetic
     character. reqInternalName must be unique for all call centers
     that you define.
<callCenter>
<section sortOrder="0" name="reqGeneralInfo" label="General Info">
   <item sortOrder="0" name="reqInternalName"</pre>
         label="Internal Name">callCenter001</item>
   <item sortOrder="1" name="reqDisplayName"</pre>
         label="Display Name">My Call Center</item>
   <item sortOrder="2" name="reqDescription"</pre>
         label="Description">Located in San Francisco, CA</item>
 <item sortOrder="3" name="reqProgId"</pre>
         label="CTI Connector ProgId">MyAdapter.MyAdapter.1</item>
 <item sortOrder="4" name="reqVersion"</pre>
         label="Version">4.0</item>
<item sortOrder="5" name="reqAdapterUrl"</pre>
         label="CTI Adapter URL">http://localhost:11000</item>
</section>
<section sortOrder="1" name="ServerInfo" label="CTI Server Info">
   <item sortOrder="0" name="HostA"</pre>
         label="Host A">Host A</item>
   <item sortOrder="1" name="PortA"</pre>
         label="Port A">Port A</item>
   <item sortOrder="2" name="HostB"</pre>
         label="Host B">Host B</item>
   <item sortOrder="3" name="PortB"</pre>
         label="Port B">Port B</item>
   <item sortOrder="4" name="PeripheralID"</pre>
         label="Peripheral ID">1000</item>
```

CHAPTER 5 Packaging and Publishing a CTI Adapter

Once you have finished updating your CTI connector code, you can package it into a fully functional CTI adapter and publish it on Force.com AppExchange.

Packaging a CTI Adapter

To package your CTI connector code into a CTI adapter:

- 1. Update the GetAdapterName () and GetAdapterVersion () methods in your COM base class as appropriate. See Steps 6 and 7 in Adding a COM Base Class to a CTI Adapter Project on page 22 for details.
- 2. Verify that you have specified a valid client key in CTIAppExchange.h. For more information, see Specifying a Valid CTI Client Key on page 17.
- 3. Compile the CTI connector into a .dll and test it with the Salesforce CRM Call Center SoftPhone connector.
- **4.** Bundle the required files to create a complete CTI adapter package. While the contents of a CTI adapter code package will vary, all CTI adapter code packages include the following:

The SoftPhone connector executable (SalesforceCTI.exe) and the adapter Web interface ConnctorPage.html

The SoftPhone connector portion of a SoftPhone CTI adapter. These files come as a pre-compiled executable with the 🚼 logo.

Your CTI connector .dll (<your_cti_system_adapter_name>.dll

The .dll component that makes up your customized CTI connector

Any dynamically-linked libraries that are required for the CTI system

Most CTI adapter implementations require additional CTI-system-specific .dll files to enable communication.

The Salesforce Office Toolkit Library (SF MSApi4.dll)

The .dll component that is required to access the Salesforce Force.com API.

Microsoft XML Library 6 (msxml6.dll and msxml6r.dll)

The .dll files that enable the SoftPhone connector to translate the SoftPhone user interface XML into HTML.

A default call center definition file (<your cti system adapter name>.xml)

The default call center definition file for a call center that uses your custom adapter. For more information, see Call Center Definition Files on page 48.

5. Create an installer for the CTI adapter package. To rapidly deploy the adapter to all machines in a call center at once, generate an .msi file that can be used with your preferred Software Management System.

Selling a CTI Adapter on the AppExchange

You can make your customized CTI adapter publicly available on the AppExchange to let other Salesforce users purchase it for their own organizations.

For detailed instructions on creating and posting your listing, see the AppExchange Publishing Guide.

CHAPTER 6 Configuring HTTPS for a CTI Adapter

Once you have finished updating your CTI connector code, packaged it into a fully functional CTI adapter, and created a call center in your Salesforce organization, you can configure your CTI adapter to use the secure hypertext transfer protocol (HTTPS). HTTPS provides added security for your call center and also helps prevent the Mixed Content warnings that can appear in your browser if your Salesforce organization uses the HTTPS protocol but your call center does not.

To configure HTTPS for a CTI adapter:

- 1. Add an HTTPS URL to your Salesforce organization's call center, as described in Adding HTTPS to a Call Center on page 54.
- 2. Add the same HTTPS URL to your CTI adapters, as described in Adding HTTPS to CTI Adapters on page 54.
- 3. Create and install a security certificate for Windows[®], as described in Creating and Installing a Security Certificate on page 55.
- Tip: After you configure HTTPS for a CTI adapter, you might need to add a security exception to users' browsers before they can log in to a call center.
- Note: HTTPS configuration is only available for CTI adapters and call centers using Windows XP and Windows 7.

Adding HTTPS to a Call Center

Once you create a call center in your Salesforce organization, you can add HTTPS to it:

- 1. From Setup, enter Call Centers in the Quick Find box, then select Call Centers.
- **2.** Click the name of a call center.
- 3. Click Edit.
- 4. Type the secure URL for your adapter in CTI Adapter URL. For example, https://localhost:11000.
- 5. Click Save.

For more information on creating a call center, see Creating a Call Center.

Adding HTTPS to CTI Adapters

Once you add HTTPS to a call center in your Salesforce organization, you can add HTTPS to CTI adpaters:

- 1. Use a text or XML editor to open SalesforceCTI.exe.config, which is located at the root directory of the drive where the CTI adapter is installed. For example, C:\Program Files\salesforce.com\Demo Adpater 4.00\SalesforceCTI.exe.config.
- 2. Change the value of the AdapterUrl setting to match the HTTPS URL you added to your call center. For example, https://localhost:11000.
- 3. Click Save.

Creating and Installing a Security Certificate

Once you add HTTPS to a call center in your Salesforce organization, and add HTTPS to CTI adpaters, you can use MakeCert to create a certificate authority. A Certificate authority validates certificates to ensure that encrypted data can only be accessed by designated users or organizations.

- (1) Important: MakeCert is a third-party product not provided or supported by Salesforce. Therefore, you might find it necessary to change the steps below to complete the specified tasks. MakeCert is included as part of the Windows SDK, which you can download from Microsoft.
 - For information about MakeCert, see http://msdn.microsoft.com/en-us/library/aa386968%28v=vs.85%29.aspx.
 - For information on using MakeCert, see http://msdn.microsoft.com/en-us/library/aa388165%28v=VS.85%29.aspx.
 - For information on configuring a port with an SSL certificate, see http://msdn.microsoft.com/en-us/library/ms733791%28v=VS.90%29.aspx.

To create and install a security certificate with MakeCert:

- 1. With administrator privileges, open a command prompt window by clicking **Start** > **All Programs** > **Accessories** > **Command Prompt**. Alternatively, you can click **Start** > **Run**, enter *cmd* in the **Open** field, and click **OK**.
- 2. In the command window, enter cd \ to navigate to the root directory of the drive where Windows SDK is installed.
- **3.** Navigate to the Windows SDK \bin directory by entering the following command (be sure to replace the file path with the path from your system):

```
cd C:\Program Files\Microsoft SDKs\Windows\v7.1\bin>.
```

- 4. Enter makecert.
- **5.** Create the certificate authority certificate. For example, enter:

```
makecert - n "CN=XXX Authority" -cy authority -a sha1 -sv "XXX_authority.pvk" -r "XXX authority.cer" -sr localmachine -ss ROOT. Note that XXX = your company name.
```

6. Create a certificate for your CTI adapter based on the certificate authority certificate created in the previous step. For example, enter:

makecert -n "CN=localhost" -ic "XXX_authority.cer" -iv "XXX_authority.pvk" -a shall -sky exchange -pe -sr localmachine -ss MY "cti adapter.cer".

7. Bind the CTI adapter certificate to the correct port number. For example (Windows XP), enter:

```
httpcfg set ssl -i 0.0.0.0:<port number> -h <certificate thumbprint>.

For example (Windows 7), enter: netsh http add sslcert ipport=0.0.0.0:<port number> certhash=<certificate thumbprint> appid=<application GUID>.
```

Mote:

- <port number> is the port number specified in SalesforceCTI.exe.config.
- < certificate thumbprint > is the thumbprint of the CTI adapter certificate created in the previous step.
- <application GUID> is any valid GUID, such as {9D06D291-3882-4D7B-A6E5-5DD6EEE2486F}.
- **8.** Check that the binding was created successfully. For example (Windows XP), enter: httpcfg query ssl. For example (Windows 7), enter: netsh http show sslcert.

CHAPTER 7 CTI Adapter Log Files

Every CTI adapter generates two log files in the installation directory of the machine where the adapter is installed:

- cti adapter.log includes information about the CTI connector component.
- browser_connector.log includes information about the SoftPhone connector component.

In default installations, the installation directory is C:\Program Files\salesforce.com\<adapterName>\, where <adapterName> is the name of the CTI adapter that was installed (for example, C:\Program Files\salesforce.com\Cisco IPCC Enterprise Adapter).

The log files document all CTI adapter activity and can be configured to report at three levels of verbosity:

Log Level	Description	
Errors Only	Error messages only	
Medium	Error messages, warnings, and information about significant actions that occur, such as setting a URL or authenticating a new user	
	Note: A warning is any irregular condition that is not fatal. For example, if a call arrives and there is no available phone line, the code issues a warning and creates a virtual line just for that call.	
High	Error messages, warnings, information about significant actions that occur, and the complete text of all method calls that send data into and out of the specified component	

By default, CTI adapters are installed with the **Errors Only** log level enabled. If you want to change the log level, right-click the adapter icon () in the system tray of the call center machine and choose the desired setting.

CHAPTER 8 Salesforce CRM Call Center API Reference

The API provides access to information about computer—telephony integration (CTI) call centers with the describeSoftphoneLayout() call. You must have the CTI feature enabled for your organization. Contact your account representative for assistance.

The API supports limited access to call center-related objects, including being able to create call centers, and create or modify additional numbers for the call center.

Topic	Description	
Call Center object description, including fields and usage.		
AdditionalNumber	Configuration settings that allow you to add an additional number if it cannot easily be categorized as a user, contact, lead, account, or any other object. Examples include phone queues or conference rooms.	

In addition, several fields have been added to existing objects to support call centers. The following fields provide configuration settings for operation of a call center.

Object Name	Field Name	Field Type	Field Properties	Description
OpenActivity ActivityHistory Task	CallDisposition	string	Create (Task only) Filter Nillable Update (Task only)	Represents the result of a given call, for example, "we'll call back," or "call unsuccessful." Limit is 255 characters. For the Task object, corresponds to the Salesforce user interface label Call Result . You can also create and update values for this field in Task.
OpenActivity ActivityHistory Task	CallDurationIn Seconds	int	Create (Task only) Filter Nillable Update (Task only)	Duration of the call in seconds. For Task, you can also create and update values for this field.
OpenActivity ActivityHistory	CallObject	string	Filter Nillable	Name of a call center. Limit is 255 characters. For Task, you can also create and update values for this field.

Object Name	Field Name	Field Type	Field Properties	Description
Task			Update (Task only)	
OpenActivity ActivityHistory Task	CallType	picklist	Create (Task only) Filter Nillable Restricted picklist Update	The type of call being answered: Inbound, Internal, or Outbound. For Task, you can also create and update values for this field.
User	CallCenterId	reference	Create Filter Nillable Update	The unique identifier for the call center associated with this user.
User	UserPermissionsCall CenterAutoLogin	boolean	Create Update	Indicates whether a user will be automatically logged in to a call center when logging in to the Salesforce application (true) or not (false).

describeSoftphoneLayout()

Retrieves layout information for a Salesforce CRM Call Center Softphone.

Syntax

DescribeSoftphoneLayoutResult[] = connection.describeSoftphoneLayout();

Usage

Use this call to obtain information about the layout of a Softphone. Use only in the context of Salesforce CRM Call Center; do not call directly from client programs.

Arguments

This call does not take any objects.

Response

The response is a DescribeSoftphoneLayoutResult object:

Name	Туре	Description
callTypes	DescribeSoftphoneLayoutCallType[]	A set of attributes associated with each allowed call type. A call type may be Inbound, Outbound, or Internal.
id	ID	ID of layout. Note that layout objects are not exposed via the API.
name	string	Name of the call type: Inbound, Outbound, or Internal.

Describe Soft phone Layout Call Type

Each DescribeSoftphoneLayoutResult object contains one or more call types:

Name	Туре	Description
infoFields	DescribeSoftphoneLayoutInfoField[]	A set of information field in the softphone layout.
name	string	Name of the layout.
screenPopOptions	DescribeSoftphoneScreenPopOption[]	Settings in the softphone layout that specify how to display screen pops when the details of calls match or don't match existing records.
		This setting only displays for softphone layouts associated with CTI 2.0 adapters or higher. See "Salesforce CTI Toolkit" in the Salesforce online help. This field is available in API version 18.0 and later.
screenPopsOpenWithin	string	Setting in the softphone layout that specify whether to display screen pops in a new browser window or tab when the details of calls match or don't match existing records.
		This setting only displays for softphone layouts associated with CTI 2.0 adapters or higher. See "Salesforce CTI Toolkit" in the Salesforce online help. This field is available in API version 18.0 and later.
sections	DescribeSoftphoneLayoutSection[]	A set of object names and the corresponding item name in the softphone layout. There is one section for each object in a call type.

DescribeSoftphoneLayoutInfoField

An information field in the softphone layout.

name string The name of an information field in the softphone layou correspond to a Salesforce object. For example, caller ID an information field. Information fields hold static inform type.	may be specified in

DescribeSoftphoneLayoutSection

Each call type returned in a DescribeSoftphoneLayoutResult object contains one section for each call type. Each section contains object-item pairs:

Name	Туре	Description
entityApiName	string	The name of an object in the Salesforce application that corresponds to an item displayed in the softphone layout, for example, a set of accounts or cases.
items	DescribeSoftphoneLayoutItem[]	A set of softphone layout items.

DescribeSoftphoneLayoutItem

Each layout item corresponds to a record in Salesforce:

Name	Type	Description
itemApiName	string	The name of a record in the Salesforce application that corresponds to an item displayed in the softphone layout, for example, the Acme account.

DescribeSoftphoneScreenPopOption

Each call type returned in a DescribeSoftphoneLayoutResult object contains one screenPopOptions field for each call type. Each screenPopOptions field contains details about screen pop settings:

Name	Type	Description
matchType	string	Setting on a softphone layout to pop a screen for call details that match a single record, multiple records, or no records.
screenPopData	string	Setting on a softphone layout for a specific object or page to pop for a call's matchType. For example, pop a specified Visualforce page when the details of a call match a record.
screenPopType	picklist	Setting that specifies how to pop a screen for a call's matchType. For example, pop a detail page or don't pop any page when the details of a call match a record.

Sample Code—Java

This sample describes the soft phone layout and writes its properties to the console. It then gets the allowed call types. For each call type, it gets its information fields, layout sections, and the layout items in the layout sections. It writes these values to the console.

```
public void describeSoftphoneLayout() {
   try {
     DescribeSoftphoneLayoutResult result =
        connection.describeSoftphoneLayout();
   System.out.println("ID of retrieved Softphone layout: " +
        result.getId());
   System.out.println("Name of retrieved Softphone layout: " +
```

```
result.getName());
   System.out.println("\nContains following " +
        "Call Type Layouts\n");
    for (DescribeSoftphoneLayoutCallType type :
         result.getCallTypes()) {
     System.out.println("Layout for " + type.getName() +
          " calls");
     System.out.println("\tCall-related fields:");
      for (DescribeSoftphoneLayoutInfoField field:
            type.getInfoFields()) {
       System.out.println("\t\t{" + field.getName());
     System.out.println("\tDisplayed Objects:");
     for (DescribeSoftphoneLayoutSection section :
           type.getSections()) {
        System.out.println("\t\tFor entity " +
            section.getEntityApiName() +
            " following records are displayed:"
       );
        for (DescribeSoftphoneLayoutItem item :
              section.getItems()) {
         System.out.println("\t\t\" + item.getItemApiName());
        }
     }
 } catch (ConnectionException ce) {
   ce.printStackTrace();
}
```

Sample Code—C#

This sample describes the soft phone layout and writes its properties to the console. It then gets the allowed call types. For each call type, it gets its information fields, layout sections, and the layout items in the layout sections. It writes these values to the console.

```
// Display the call-related fields contained in the call type
           Console.WriteLine("\tCall-related fields:");
           foreach (DescribeSoftphoneLayoutInfoField dsplInfoField
               in dsplCallType.infoFields)
            Console.WriteLine("\t\t{0}", dsplInfoField.name);
           // Display the objects that are included in the layout
           Console.WriteLine("\tDisplayed Objects:");
           foreach (DescribeSoftphoneLayoutSection dsplSection
               in dsplCallType.sections)
               Console.WriteLine("\t\tFor entity {0} following records are displayed:",
                   dsplSection.entityApiName);
               foreach (DescribeSoftphoneLayoutItem dsplItem in dsplSection.items)
                   Console.WriteLine("\t\t\t{0}\", dsplItem.itemApiName);
            }
  }
  catch (SoapException e)
      Console.WriteLine(e.Message);
      Console.WriteLine(e.StackTrace);
      Console.WriteLine(e.InnerException);
}
```

CallCenter

Represents a call center, which is a logical representation of a single computer-telephony integration (CTI) system instance in an organization.

Supported Calls

create(), describeSObjects(), getDeleted(), getUpdated(), query(), retrieve()

Special Access Rules

Customer Portal users can't access this object.

Fields

Field	Details
AdapterURL	Type string

Field	Details
	Properties
	Create, Filter, Group, Nillable, Sort
	Description
	An optional field that specifies the location of where the CTI adapter is hosted. For example, http://localhost:11000.
	This field is available for call centers using CTI Toolkit version 4.0 and API version 23.0 or later.
CustomSettings	Туре
	string
	Properties Create, Filter, Group, Nillable, Sort
	Description
	Specifies settings in the call center definition file, such as whether the call center uses the Open CTI, and SoftPhone properties, such as height in pixels.
	This field is available for Open CTI and in API version 25.0 or later.
Id	Туре
	ID
	Properties Defaulted on create, Filter
	Description System field that uniquely identifies this call center. Label is Call Center ID . This ID is created automatically when the call center is created.
InternalName	Type string
	Properties
	Create, Filter, Group, Sort
	Description
	The internal name of the call center.
	Limit is 80 characters.
Name	Туре
	string
	Properties Create, Filter, Group, Sort
	Description
	The name of the call center.
	The name of the can center.

Field	Details
Version	Type double
	Properties Create, Filter, Nillable, Sort
	Description The version of the CTI Developer's Toolkit used to create the call center (for versions 2.0 and later). This field is available in API version 18.0 and later.
	This field is available in API version 18.0 and later.

Usage

Create a call center or query an existing call center.

AdditionalNumber

Represents an optional additional number for a call center. This additional number is visible in the call center's phone directory.

Supported Calls

create(), delete(), describeSObjects(), getDeleted(), getUpdated(), query(), retrieve(),
undelete(), update(), upsert()

Special Access Rules

Customer Portal users can't access this object.

Fields

Field	Details
CallCenterId	Type reference
	Properties Create, Filter, Group, Nillable, Sort, Update
	Description System field that contains the ID of the user who created the call center associated with this additional number. If value is null, this additional number is displayed in every call center's phone directory.
Description	Type string

Field	Details
	Properties Create, Filter, Group, Nillable, Sort, Update
	Description Description of the additional number, such as Conference Room B.
	Limit: 255 characters.
Name	Type string
	Properties Create, Filter, Group, Sort, Update
	Description The name of the additional number.
	Limit: 80 characters.
Phone	Type phone
	Properties Create, Filter, Nillable, Group, Sort, Update
	Description The phone number that corresponds to this additional number.

Usage

Create an additional number for a call center directory. Use this object if the number is not easily categorized as a User, Contact, Lead, Account, or the other object. Examples include phone queues or conference rooms.

FREQUENTLY ASKED QUESTIONS

Review the following frequently asked questions about the CTI Toolkit.

Q: What is the difference between Salesforce CRM Call Center and the CTI Toolkit?

Salesforce CRM Call Center is an "edition" of Salesforce in the mold of Force.com Connect for Microsoft Outlook or Connect Offline. Salesforce does not charge for this edition, and it can be enabled in Developer Edition, Group Edition, Professional Edition, Enterprise Edition, Unlimited Edition, and Performance Edition.

The CTI Toolkit is a body of C++ code that allows partners to integrate telephony systems with Salesforce using Salesforce CRM Call Center. It creates a client-side application that resides in the Windows system tray and abstracts access to the SOAP API.

Salesforce is not selling any CTI adapters for any phone systems. Instead, Salesforce is providing the CTI Toolkit to its partners so that they can develop Salesforce CRM Call Center integrations with a consistent user interface.

Q: Why does Salesforce CRM Call Center involve a client-side application? Isn't Salesforce the "No Software" company?

Salesforce continues to be the "No Software" company, except in those cases when it is necessary to integrate with a service that sits behind a firewall. In such situations, Salesforce provides small pieces of software to perform that integration. Other examples of Salesforce software include Connect for Office and Connect for Outlook.

Q: Does the CTI adapter support multiple telephone lines?

Yes. The Salesforce CTI adapter supports multiple fixed lines and multiple calls for the same line, for which it creates "virtual" lines that disappear after the call has ended. In total, the adapter supports up to five fixed or virtual lines at once.

Q: Is it possible to have multiple CTI adapters working in parallel against the same switch?

This is possible for the vast majority of switches, though some switches restrict the same agent from logging in to more than one CTI adapter at the same time.

Q: Does a CTI adapter require any investment or changes to a switch?

This depends on your organization's needs.

There are two types of CTI adapters: those that integrate with a public-branch exchange (PBX), and those that integrate with an automatic call distributer (ACD).

- Those that integrate with a PBX do not have a notion of agent presence, and do not receive any data attached to a call. This means that a call center user cannot specify whether he or she is ready to receive calls. Instead, the adapter will simply display a new call whenever the phone rings. Additionally, PBX-level integrations cannot generate screen pops based on interactive voice response (IVR) data, such as when a caller enters an account number before being connected to a user. PBX-level integrations typically can only generate screen pops based on the incoming caller ID (the automatic number identification, or ANI).
- Those that integrate with an ACD allow call center users to specify whether they are ready to take calls and can receive data attached to calls, such as an account number.

If your organization has a PBX system but wants ACD integration, you must buy an ACD system.

Additionally, some Salesforce partners provide a middleware server that sits between the telephony system and the call center user and normalized the messages passing between the two. If your organization chooses a partner such as this, you must install the partner's middleware.

Q: Does Salesforce CRM Call Center require VoIP?

No, Salesforce CRM Call Center can work with regular telephone service switches, as long as they have a CTI server attached to them.

Q: How long does it usually take for a partner to write a custom CTI adapter?

Salesforce generally estimates that it should take three to six weeks of development time and three to six weeks of testing time to write a new CTI adapter.

Q: Is there a list of the telephony platforms that are currently covered?

Yes. Please contact your Salesforce representative to learn about the commitments that have already been made by other partners.

Q: How can I demo a CTI adapter?

Salesforce has provided a "demo adapter" that does not require any phone system behind it, but that can simulate incoming and outgoing calls. See The Demo Adapter on page 11 for information.

Q: I'm seeing 'L' prefixes in front of all the string and character literals in the CTI Toolkit code. Why are these present?

The 'L' prefix indicates that the string will be stored as an array of "wide" (2-byte) characters, which are necessary to support Unicode character encoding. Salesforce enforces the use of this prefix on all string values to support localization. For more information, see The 'L' Literal String and Character Prefix on page 16.

Q: My SoftPhone is not behaving the way that I expect it to. How can I troubleshoot it?

If you are experiencing difficulties with a SoftPhone:

- Verify that you are running Internet Explorer version 7, 8, or 9 or Firefox version 3.5, 3.6, or 4 (Safari and Chrome are not supported).
- Try logging out of Salesforce and then logging back in.
- Try stopping and restarting your CTI adapter:
 - 1. Right-click the CTI adapter system tray icon (😭) in the lower-right corner of your computer screen and select **Exit**.
 - 2. From your machine's **Start** menu, choose **Programs** > **Salesforce** > **<Your_CTI_Adapter_Name>** > **Salesforce CTI Adapter**.

If you are still experiencing difficulties, examine the CTI adapter log files for more information. For information, see CTI Adapter Log Files on page 56.

GLOSSARY

Agent

A Salesforce CRM Call Center user who handles inbound or outbound calls. An agent is usually identified by a four-digit number that serves as the agent's ID within the associated computer telephony integration (CTI) system.

Automatic Number Identification (ANI)

The number from which a caller is dialing in Salesforce CRM Call Center.

Call

Any inbound, outbound, consult, or internal voice connection via telephone.

Salesforce CRM Call Center

A Salesforce feature that seamlessly integrates Salesforce with third-party computer-telephony integration (CTI) systems. For more information, see "Salesforce Call Center" in the Salesforce online help.

Computer-Telephony Integration (CTI)

The linkage between a telephone system and a computer that facilitates incoming- and outgoing-call handling and control.

Consult Call

A call that results from a Salesforce CRM Call Center user initiating a conference or transfer.

CTI Adapter

A lightweight software program that controls the appearance and behavior of a Salesforce softphone. The adapter acts as an intermediary between a third-party computer telephony integration (CTI) system, Salesforce, and a Salesforce CRM Call Center user. It must be installed on any machine that needs access to Salesforce CRM Call Center functionality..

CTI Connector

A component of a computer telephony integration (CTI) adapter that maintains an in-memory representation of a Salesforce CRM Call Center user's softphone, including the phone numbers, records, and status associated with a call. The CTI connector is the component of a CTI adapter that can be customized by an organization.

CTI System

The hardware and software that implements computer-telephony integration (CTI) for a particular call center.

Dial-Tone Multi-Frequency (DTMF)

The system that informs a switch of the number that is being pressed by a caller in Salesforce CRM Call Center.

Dialed Number Identification Service (DNIS)

The number a caller dialed in Salesforce CRM Call Center.

Directory Number (DN)

Any internal number that is configured on a public branch exchange. You can define additional directory numbers through the Salesforce CRM Call Center setup within Salesforce.

Event

A message broadcast from a CTI system that alerts any registered listeners that an action has taken place in the phone system. For example, when a user's telephone rings, a CTI system broadcasts a "RINGING" event. A CTI adapter, the Salesforce CRM Call Center component that acts as a listener, receives this event and updates the SoftPhone as appropriate.

Event Sink

An object in a computer telephony integration (CTI) connector that receives CTI system events and routes them to other methods for processing.

Inbound Call

A call that originates from another party in Salesforce CRM Call Center.

Interactive Voice Response (IVR)

The hardware and software that prompts a Salesforce CRM Call Center caller to enter specific digits, such as a menu selection, or an account number. IVR is also known as a Voice Response Unit (VRU).

Internal Call

A call between users in the same call center in Salesforce CRM Call Center.

Outbound Call

Any call that originates from a user to a number outside of a call center in Salesforce CRM Call Center.

Private Branch Exchange (PBX)

A telephony switch that is used exclusively by a single call center to route calls in Salesforce CRM Call Center.

Queue

A mechanism for storing one or more inbound calls that cannot be immediately answered by a Salesforce CRM Call Center user. Some CTI systems use multiple queues to differentiate between different types of calls.

Routing Point

A mechanism that determines the Salesforce CRM Call Center queue that should control an incoming call.

SoftPhone

The telephone interface that a Salesforce CRM Call Center user sees in either the sidebar of pages in Salesforce Classic or the footer of pages in Lightning Experience and the Salesforce console.

SoftPhone Connector

A component of a CTI adapter that converts SoftPhone XML into HTML and distributes it to a call center user's browser.

Voice Response Unit (VRU)

See Interactive Voice Response (IVR).

INDEX

'L' prefix on string literals[L prefix on string literals] 16	CCTIUserInterface
	event handlers 6
A	method names 17
Adapters	SetInfoFieldLabel() method 47
See CTI adapters 2	SoftPhone command routing 6
AdditionalNumber object 64	CCTIUserInterface class
Agent states	adding a call information field 43
adding 42	adding a custom logo 43
displaying 41	adding buttons 37
mapping 40	adding call center states 42
AgentStateToString method 40	button layout 36
ANI search 45	CallInitiateConference method 35
API 57	CallInitiateTransfer method 35
AppExchange	CreateLoginForm method 32
publishing CTI adapters 53	CTIChangeAgentState() method 42
Attached data search 45–46	CTIIsOccupiedAgentState() method 42
_	CTILogin method 32, 34
В	customizing 29
Blind transfers and conferences 35	customizing call logs 46
browser_connector.log 56	displaying call center states 41
Buttons, SoftPhone	displaying call information 43
enabling 35	displaying call-related records 44
_	enabling buttons 35
C	enabling reason codes 38–39
Call center definition files	implementing an event sink 26
default 10	implementing call center user commands 30
Call center states	Initialize method 32
adding 42	Initialize() method 42
displaying 41	mapping call center states 40
mapping 40	non-required subclass commands 31
Call centers	OnCallDialing event handler 27
adding HTTPS 54	OnCallRinging event handler 27
associating with an adapter 66	required subclass commands 30
ProgID 66	SetOneStepConferenceEnabled method 35
Call centers and the API 57	SetOneStepTransferEnabled method 35
Call information field labels 47	subclassing 24
Call information fields 43	UIAddAgentState() method 42
Call information, displaying 43	UIHandleMessage method 30
Call logs	UIParseIncomingXMLMessage method 30
customizing 46	Virtual keyword 29
CallCenter object 62	Client keys 17
CallInitiateConference method 35	COM base classes 22
CallInitiateTransfer method 35	Command messages
Calls	implementing 30
describeSoftphoneLayout() 58	Command routing 6
CCTISearch 12	Conferences 35

Index

Constants 16	Custom SoftPhone labels 47
CreateLoginForm method, CCTIUserInterface 32	D
CTI adapters	D
about 5	Database
adding HTTPS 54	Salesforce 7
associating with a call center 66	Demo adapter
building a CTI connector solution 20	customizing 12
building new CTI connectors 20	setting up 11
code package contents 10	using 11
configuring HTTPS 54	describeSoftphoneLayout() call 58
CTI connectors 5	Documentation
customizing 19	about 2
demo adapter 11–12	E
displaying call information 43	E
log files 56	Event sinks
packaging 53	OnCallRinging event handler 27
ProgID 66	required event handlers 26
registering with CTI systems 7	Event sinks, CTI connector 12
selling on the AppExchange 53	_
setting up a new CTI connector solution 20	F
SoftPhone connectors 6	Frequently asked questions 66
system requirements 2	
CTI client keys 17	G
CTI connector	GetAdapterName() method 22
source code 10	GetAdapterVersion() method 22
CTI connectors	Glossary 69
adding a COM base class 22	1.1
building a solution 20	Н
building new 20	HTTPS
CCTIUserInterface commands 30–31	CTI adapters 54
classes 12	installing a security certificate 55
Commands, call center user 30	
customizing 19	
event sinks 26–27	Initialize method, CCTIUserInterface 32
processing CTI system events 6	Initialize() method 42
ProgID 22	
setting up a new project 20	L
SoftPhone command routing 6	Layout, SoftPhone
subclassing CCTIUserInterface 24	buttons 36
CTI systems	Localizing SoftPhones 47
registering adapters 7	Log files 56
CTI toolkit 9	Logos, custom SoftPhone 43
CTI Toolkit 4.0 Developer's Guide	Logout reason codes, enabling 38
about 1	Logs, call
cti_adapter.log 56	customizing 46
CTIChangeAgentState() method 42	A A
CTIConstants.h 16	M
CTIIsOccupiedAgentState() method 42	Messages
CTILogin method, CCTIUserInterface 32, 34	call center user command 30

Index

Mute transfers and conferences 35	SoftPhone connectors
N	executable 10
	processing CTI system events 6
Not-ready reason codes, enabling 38	SoftPhone command routing 6
0	SoftPhone layouts
	buttons 36
Objects	SoftPhones
AdditionalNumber 64	adding a custom logo 43
CallCenter 62	custom labels 47
Office Toolkit library 10	customizing 29
OnAgentStateChange method 40	translating labels 47
OnAgentStateEnablementChange() method 41	troubleshooting 66
OnButtonEnablementChange method 35	SOQL 7
OnCallDialing event handler, CCTIUserInterface 27	SOSL 7
OnCallRinging event handler, CCTIUserInterface 27, 43, 45, 47	String literals and the 'L' prefix 16
One-step transfers and conferences 35	System architecture 4
D	System events 6
P	System requirements 2
PARAM_MAPs 27	т
ProgID 22, 66	1
D	Transfers 35
R	Translating SoftPhone labels 47
Reason codes, enabling 38–39	Troubleshooting tips 66
S	U
Salesforce database 7	UIAction() method 6, 22
Search, modifying 44–46	UIAddAgentState() method 42
SendUlRefreshEvent() method 22	UIAddButtonToAllLines method 37
SetAgentStateEnablement() method 41	UIHandleMessage method 30
SetInfoFieldLabel() method 47	UIHandleMessage() method 6
SetLogoutReasonCodes method 38	UIParseIncomingXMLMessage method 30
SetLogoutReasonRequired method 38	UIRefresh event 5–6
SetNotReadyReasonCodes method 38	User states
SetNotReadyReasonRequired method 38	adding 42
SetOneStepConferenceEnabled method 35	displaying 41
SetOneStepTransferEnabled method 35	mapping 40
SetWrapUpReasonCodes method 38	
Single-step transfers and conferences 35	V
SoftPhone	Virtual keyword 29
adding buttons 37	,
displaying call-related records 44	W
enabling buttons 35	Wrap-up reason codes, enabling 38