Oracle® Telephony Adapter SDK

Developers Reference Guide

Release 11*i*

Part No. A97207-03

April 2003

ORACLE®

Oracle Telephony Adapter SDK Developer's Reference Guide, Release 11*i*

Part No. A97207-03

Contents

6 Implementing Multi-Site Functionality for the Telephony Adapter

7 Testing the Telephony Adapter

E Diagnostics and Troubleshooting

Send Us Your Comments

Oracle Telephony Adapter SDK Developers Reference Guide, Release 11*i*

Part No. A97207-03

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us at the following address:

Oracle Corporation
Oracle Telephony Adapter SDK Documentation
1900 Oracle Way
Reston, Virginia 20190
USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

Audience for This Guide

Welcome to Release 11*i* of the Oracle Telephony Adapter SDK Developers Reference Guide.

This guide assumes you have a working knowledge of the following:

Telephony Developers

- Java and C programming languages
- TCP/IP sockets
- Threads, such as multithreading, synchronization and wait-notify
- Real-time programming, such as event driven and synchronous versus asynchronous modes

Telephony Analysts

- Call center concepts, such as ACD queue, route points, agent work modes, ANI and DNIS
- Telephony platform (PBX and CTI middleware) call model, such as transfer, conference and call ID transitions
- Vendor-specific CTI APIs and test tools, such as Intel NetMerge Call Processing Software and Cisco ICM

See "Other Information Sources" in this preface for Oracle Applications product information.

How To Use This Guide

This guide contains the information you need to understand and use Oracle Telephony Adapter SDK, and contains the following chapters and appendixes.

- Chapter 1, "Introduction" provides an overview of the features and functions of Oracle Telephony Adapter SDK with Oracle Advanced Inbound.
- Chapter 2, "Concepts" explains concepts and gives details, such as keys and values, of Oracle Telephony Adapter SDK.
- Chapter 3, "Development Environment" lists minimum software and hardware requirements, typical configurations and scalability and performance guidelines.
- Chapter 4, "Oracle Telephony Adapter SDK Components" explains the contents and installation of the SDK, and describes the APIs for the C and Java programming languages.
- Chapter 5, "Developing the Telephony Adapter" provides step-by-step instructions on implementing Oracle Telephony Adapter SDK.
- Chapter 6, "Implementing Multi-Site Functionality for the Telephony Adapter" describes the procedures for Multi-Site Telephony Adapter implementations.
- Chapter 7, "Testing the Telephony Adapter" explains how to verify the adapter implementation.
- Chapter 8, "Deploying the Telephony Adapter" explains the steps involved in packaging and uploading Oracle Telephony Adapter to deploy the implementation in a live interaction center environment.
- Chapter 9, "Basic Integration" describes the Basic Integration of Telephony Adapter SDK.
- Chapter 10, "Oracle Advanced Outbound Extension" explains the Oracle Telephony Adapter SDK extension for Oracle Advanced Outbound.
- Appendix A, "SDK Qualification Worksheet" is a list of questions to use at a customer site in determining if the integration of Oracle Telephony Adapter SDK is necessary and what the SDK requirements are.
- Appendix B, "SDK Scope Analysis" is a list of questions to use at a customer site in refining the SDK requirements.
- Appendix C, "Telephony Adapter Test Cases" lists Oracle Telephony Manager events.

- Appendix D, "Sample Code" contains samples of relevant code.
- Appendix E, "Diagnostics and Troubleshooting" describes how to troubleshoot Oracle Advanced Inbound implementations using Oracle Telephony Adapter SDK.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Other Information Sources

You can choose from many sources of information, including online documentation, training, and support services, to increase your knowledge and understanding of Oracle Telephony Adapter SDK.

If this guide refers you to other Oracle Applications documentation, use only the Release 11*i* versions of those guides.

Online Documentation

All Oracle Applications documentation is available online (HTML or PDF). Online help patches are available on MetaLink.

Related Documentation

Oracle Telephony Adapter SDK shares business and setup information with other Oracle Applications products. Therefore, you may want to refer to other product documentation when you set up and use Oracle Telephony Adapter SDK.

You can read the documents online by choosing Library from the expandable menu on your HTML help window, by reading from the Oracle Applications Document Library CD included in your media pack, or by using a Web browser with a URL that your system administrator provides.

If you require printed guides, you can purchase them from the Oracle Store at http://oraclestore.oracle.com.

Documents Related to All Products

Oracle Applications User's Guide

This guide explains how to enter data, query, run reports, and navigate using the graphical user interface (GUI) available with this release of Oracle Telephony Adapter SDK (and any other Oracle Applications products). This guide also includes information on setting user profiles, as well as running and reviewing reports and concurrent processes.

You can access this user's guide online by choosing "Getting Started with Oracle Applications" from any Oracle Applications help file.

Documents Related to This Product

Oracle Advanced Inbound Implementation Guide

This guide contains the post-installation implementation procedures for configuring Oracle Advanced Inbound.

Oracle Interaction Center Server Manager Implementation Guide

This guide contains the installation and implementation information for Oracle Interaction Center Server Manager.

Installation and System Administration

Oracle Applications Concepts

This guide provides an introduction to the concepts, features, technology stack, architecture, and terminology for Oracle Applications Release 11*i*. It provides a useful first book to read before an installation of Oracle Applications. This guide also introduces the concepts behind Applications-wide features such as Business Intelligence (BIS), languages and character sets, and Self-Service Web Applications.

Installing Oracle Applications

This guide provides instructions for managing the installation of Oracle Applications products. In Release 11*i*, much of the installation process is handled using Oracle Rapid Install, which minimizes the time to install Oracle Applications, the Oracle8 or Oracle9technology stack, and the Oracle8*i* or Oracle9*i* Server technology stack by automating many of the required steps. This guide contains instructions for using Oracle Rapid Install and lists the tasks you need to perform to finish your installation. You should use this guide in conjunction with individual product user's guides and implementation guides.

Oracle Applications Supplemental CRM Installation Steps

This guide contains specific steps needed to complete installation of a few of the CRM products. The steps should be done immediately following that tasks given in the Installing Oracle Applications guide.

Upgrading Oracle Applications

Refer to this guide if you are upgrading your Oracle Applications Release 10.7 or Release 11.0 products to Release 11*i*. This guide describes the upgrade process and lists database and product-specific upgrade tasks. You must be either at Release 10.7 (NCA, SmartClient, or character mode) or Release 11.0, to upgrade to Release 11*i*. You cannot upgrade to Release 11*i* directly from releases prior to 10.7.

Maintaining Oracle Applications

Use this guide to help you run the various AD utilities, such as AutoUpgrade, AutoPatch, AD Administration, AD Controller, AD Relink, License Manager, and others. It contains how-to steps, screen shots, and other information that you need to run the AD utilities. This guide also provides information on maintaining the Oracle applications file system and database.

Oracle Applications System Administrator's Guide

This guide provides planning and reference information for the Oracle Applications System Administrator. It contains information on how to define security, customize menus and online help, and manage concurrent processing.

Oracle Alert User's Guide

This guide explains how to define periodic and event alerts to monitor the status of your Oracle Applications data.

Oracle Applications Developer's Guide

This guide contains the coding standards followed by the Oracle Applications development staff. It describes the Oracle Application Object Library components needed to implement the Oracle Applications user interface described in the *Oracle Applications User Interface Standards for Forms-Based Products*. It also provides information to help you build your custom Oracle Forms Developer 6*i* forms so that they integrate with Oracle Applications.

Oracle Applications User Interface Standards for Forms-Based Products

This guide contains the user interface (UI) standards followed by the Oracle Applications development staff. It describes the UI for the Oracle Applications products and how to apply this UI to the design of an application built by using Oracle Forms.

Other Implementation Documentation

Multiple Reporting Currencies in Oracle Applications

If you use the Multiple Reporting Currencies feature to record transactions in more than one currency, use this manual before implementing Oracle Telephony Adapter SDK. This manual details additional steps and setup considerations for implementing Oracle Telephony Adapter SDK with this feature.

Multiple Organizations in Oracle Applications

This guide describes how to set up and use Oracle Telephony Adapter SDK with Oracle Applications' Multiple Organization support feature, so you can define and support different organization structures when running a single installation of Oracle Telephony Adapter SDK.

Oracle Workflow Guide

This guide explains how to define new workflow business processes as well as customize existing Oracle Applications-embedded workflow processes.You also use this guide to complete the setup steps necessary for any Oracle Applications product that includes workflow-enabled processes.

Oracle Applications Flexfields Guide

This guide provides flexfields planning, setup and reference information for the Oracle Telephony Adapter SDK implementation team, as well as for users responsible for the ongoing maintenance of Oracle Applications product data. This manual also provides information on creating custom reports on flexfields data.

Oracle eTechnical Reference Manuals

Each eTechnical Reference Manual (eTRM) contains database diagrams and a detailed description of database tables, forms, reports, and programs for a specific Oracle Applications product. This information helps you convert data from your existing applications, integrate Oracle Applications data with non-Oracle applications, and write custom reports for Oracle Applications products. Oracle eTRM is available on MetaLink

Oracle Manufacturing APIs and Open Interfaces Manual

This manual contains up-to-date information about integrating with other Oracle Manufacturing applications and with your other systems. This documentation includes APIs and open interfaces found in Oracle Manufacturing.

Oracle Order Management Suite APIs and Open Interfaces Manual

This manual contains up-to-date information about integrating with other Oracle Manufacturing applications and with your other systems. This documentation includes APIs and open interfaces found in Oracle Order Management Suite.

Oracle Applications Message Reference Manual

This manual describes Oracle Applications messages. This manual is available in HTML format on the documentation CD-ROM for Release 11*i*.

Oracle CRM Application Foundation Implementation Guide

Many CRM products use components from CRM Application Foundation. Use this guide to correctly implement CRM Application Foundation.

Training and Support

Training

Oracle offers training courses to help you and your staff master Oracle Advanced Inbound and reach full productivity quickly. You have a choice of educational environments. You can attend courses offered by Oracle University at any one of our many Education Centers, you can arrange for our trainers to teach at your facility, or you can use Oracle Learning Network (OLN), Oracle University's online education utility. In addition, Oracle training professionals can tailor standard courses or develop custom courses to meet your needs. For example, you may want to use your organization structure, terminology, and data as examples in a customized training session delivered at your own facility.

Support

From on-site support to central support, our team of experienced professionals provides the help and information you need to keep Oracle Telephony Adapter SDK working for you. This team includes your Technical Representative, Account Manager, and Oracle's large staff of consultants and support specialists with expertise in your business area, managing an Oracle8*i* server, and your hardware and software environment.

The Oracle Telephony Adapter SDK includes a set of public and published APIs. The Oracle Telephony Adapter SDK APIs and the API documentation are supported directly by Oracle through the Oracle support team. The products developed by field teams using the SDK, such as a custom telephony adapter, are typically not supported directly by Oracle unless specific support arrangements have been made.

Oracle*MetaLink*

Oracle*MetaLink* is your self-service support connection with Web, telephone menu, and e-mail alternatives. Oracle supplies these technologies for your convenience, available 24 hours a day, 7 days a week. With Oracle*MetaLink*, you can obtain information and advice from technical libraries and forums, download patches, download the latest documentation, look at bug details, and create or update TARs. To use MetaLink, register at (http://metalink.oracle.com).

Alerts: You should check Oracle*MetaLink* alerts before you begin to install or upgrade any of your Oracle eApplications. Navigate to the Alerts page as follows: Technical Libraries/ERP Applications/Applications Installation and Upgrade/Alerts.

Self-Service Toolkit: You may also find information by navigating to the Self-Service Toolkit page as follows: Technical Libraries/ERP Applications/Applications Installation and Upgrade.

Do Not Use Database Tools to Modify Oracle Applications Data

Oracle STRONGLY RECOMMENDS that you never use SQL*Plus, Oracle Data Browser, database triggers, or any other tool to modify Oracle Applications data unless otherwise instructed.

Oracle provides powerful tools you can use to create, store, change, retrieve, and maintain information in an Oracle database. But if you use Oracle tools such as SQL*Plus to modify Oracle Applications data, you risk destroying the integrity of your data and you lose the ability to audit changes to your data.

Because Oracle Applications tables are interrelated, any change you make using Oracle Applications can update many tables at once. But when you modify Oracle Applications data using anything other than Oracle Applications, you may change a row in one table without making corresponding changes in related tables. If your tables get out of synchronization with each other, you risk retrieving erroneous information and you risk unpredictable results throughout Oracle Applications.

When you use Oracle Applications to modify your data, Oracle Applications automatically checks that your changes are valid. Oracle Applications also keeps track of who changes information. If you enter information into database tables using database tools, you may store invalid information. You also lose the ability to track who has changed your information because SQL*Plus and other database tools do not keep a record of changes.

About Oracle

Oracle Corporation develops and markets an integrated line of software products for database management, applications development, decision support, and office automation, as well as Oracle Applications, an integrated suite of more than 160 software modules for financial management, supply chain management, manufacturing, project systems, human resources and customer relationship management.

Oracle products are available for mainframes, minicomputers, personal computers, network computers and personal digital assistants, allowing organizations to integrate different computers, different operating systems, different networks, and even different database management systems, into a single, unified computing and information resource.

Oracle is the world's leading supplier of software for information management, and the world's second largest software company. Oracle offers its database, tools, and applications products, along with related consulting, education, and support services, in over 145 countries around the world.

1 Introduction

This section contains the following topics:

1.1 Oracle Interaction Center and Oracle Advanced Inbound

Oracle Interaction Center is a family of products that serve as the telephony-enabling foundation of Oracle's eBusiness Suite of applications. Oracle Advanced Inbound is an Oracle Interaction Center product specifically for inbound interaction centers, and which has the following primary functions:

- Agent softphone
- Agent screen pop
- Virtual queuing
- Call routing
- Call and data transfer

Oracle Advanced Inbound consists of a group of server processes, including:

- Inbound Telephony Server, which monitors incoming calls arriving at ACD queues and route points
- Interaction Queuing and Distribution, which maintains a virtual queue of media items for each agent

- Routing Server, which routes and classifies calls
- Oracle Telephony Adapter Server, which normalizes telephony platform-specific messages and events, and provides a single interface between the Oracle Interaction Center and the underlying CTI platform
- Oracle Telephony Manager, which monitors and controls agent extensions
- Oracle Universal Work Queue server, which maintains a virtual work queue for each agent, and provides a single interface between the Oracle Interaction Center and the Oracle E-Business Suite applications

1.2 Overview of Oracle Telephony Adapter SDK

Oracle Advanced Inbound features out-of-the-box screen pops of customer data in the Oracle eBusiness Suite, call routing and classifications based on IVR and business data, a feature-rich soft phone and a software development kit (SDK). With the SDK, consultants can make custom integrations to virtually any PBX or CTI middleware that support CTI interfaces or that have the capability to pass data between their telephony client using XML and HTTP.

By way of its PBX/ACD and CTI Middleware Integration Program, Oracle provides Oracle-certified telephony adapters, enabling out-of-the-box integration to a selection of leading PBX/CTI middleware combinations. Integrations using the SDK can be either server based or client based. Server-based integrations offer the richest set of CTI functionality, whereas client-based integrations offer basic login, dialing and application screen pops.

Oracle Telephony Adapter SDK provides a method for consultants and partners to integrate telephony platforms that are not supported directly by Oracle Advanced Inbound. Field teams can use Oracle Telephony Adapter SDK to develop an adapter that plugs into the Telephony Adapter Server and supports CTI integration for the customer's telephony platform.

The Oracle Telephony Adapter SDK consists of the following components:

- Oracle Telephony Adapter API, the interfaces that the adapter must implement and are called by Oracle Advanced Inbound, and the interfaces that Oracle Advanced Inbound implements and are called by the adapter.
- Oracle Telephony Adapter SDK Programmers Reference Guide, (this document) which explains how to use the API to implement and deploy the adapter, test the adapter implementation, and so on.
- Test Utility for testing and validating the adapter implementation.

Oracle Telephony Adapter SDK has three levels of integration, which are described in the following sections:

- Section 1.2.1, "Advanced Integration"
- Section 1.2.2, "Standard Integration"
- Section 1.2.3, "Basic Integration"

1.2.1 Advanced Integration

Advanced Integration is the full implementation of all the functions and features required by Oracle Advanced Inbound and Oracle Advanced Outbound. Advanced Integration requires implementing all the methods and interfaces specified by the Telephony Adapter SDK. The runtime system requires configuring and running all Interaction Center server processes: Oracle Universal Work Queue, Interaction Queuing and Distribution, Oracle Telephony Manager, Inbound Telephony Server, Oracle Telephony Adapter Server and Oracle Routing Server.

The following table lists the Advanced Integration features.

Table 1–1 Advanced Integration Features

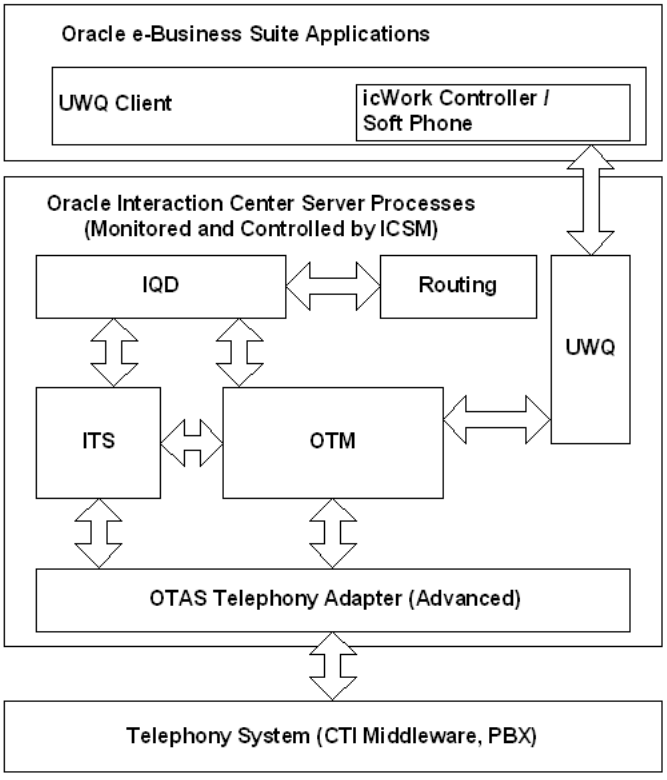
Feature	Advanced Integration
Summary	■ Screen pop ■ Active/Enhanced Passive/Passive ■ Full Softphone ■ IVR Integration ■ Advanced Outbound
Screen Pop	Screen pop based on data from: ■ ANI ■ DNIS ■ PBX application data ■ IVR data
CTI Functionality	■ Answer Call Release Call Hold / Retrieve ■ Forward ■ Send DTMF Digits ■ Route Point Monitoring and Control

Table 1–1 Advanced Integration Features (Cont.)

Feature	Advanced Integration
Softphone	Full Softphone: ■ Three Line buttons ■ Hold button ■ Release button ■ Forward button
Call Routing	Oracle Advanced Inbound does the routing (Active Mode)
Call Classification	Classification based on: ■ DNIS ■ IVR Data ■ PBX application data ■ IVR data
Outbound Dialing Modes	Choice of Advanced Outbound dialing modes: ■ Predictive ■ Progressive ■ Preview
Interaction History	Full Interaction History: ■ Life cycle segments (IVR, in queue, with agent) ■ Abandoned calls
Web Callback	Fully integrated

As the following figure illustrates, in Advanced Integration, the Telephony Adapter Server is the intermediary between the telephony system and Oracle Advanced Inbound. Oracle Universal Work Queue is the intermediary between Oracle Advanced Inbound and the agent's client system, including the icWork Controller and Advanced Integration softphone.

Figure 1–1 Advanced Integration Architecture

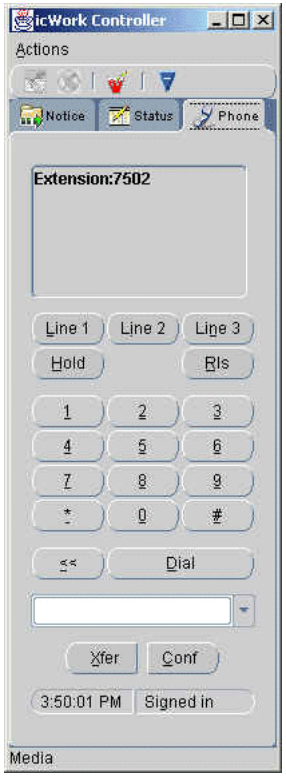


ITS = Oracle Inbound Telephony Server
IQD = Interaction Queuing and Distribution
OTM = Oracle Telephony Manager
UWQ = Oracle Universal Work Queue
ICSM = Interaction Center Server Manager

Advanced Integration Softphone

The following illustration shows the softphone with full features, displaying buttons for three lines, hold, release, dial, transfer, conference and numbered dialing buttons.

Figure 1–2 Advanced Integration Softphone



See Also

- Chapter 2, "Concepts"
- Chapter 3, "Development Environment"
- Chapter 4, "Oracle Telephony Adapter SDK Components"
- Chapter 5, "Developing the Telephony Adapter"

- Chapter 7, "Testing the Telephony Adapter"
- Chapter 8, "Deploying the Telephony Adapter"

1.2.2 Standard Integration

Standard Integration requires implementing a subset of the methods and interfaces that are specified by the Telephony Adapter SDK. The runtime system requires configuring and running all Interaction Center server processes required for Oracle Advanced Inbound *except* for Inbound Telephony Server and Interaction Queuing and Distribution. Oracle Routing Server does classifications.

The following table lists and describes the features of Standard Integration.

Table 1–2 Standard Integration Features

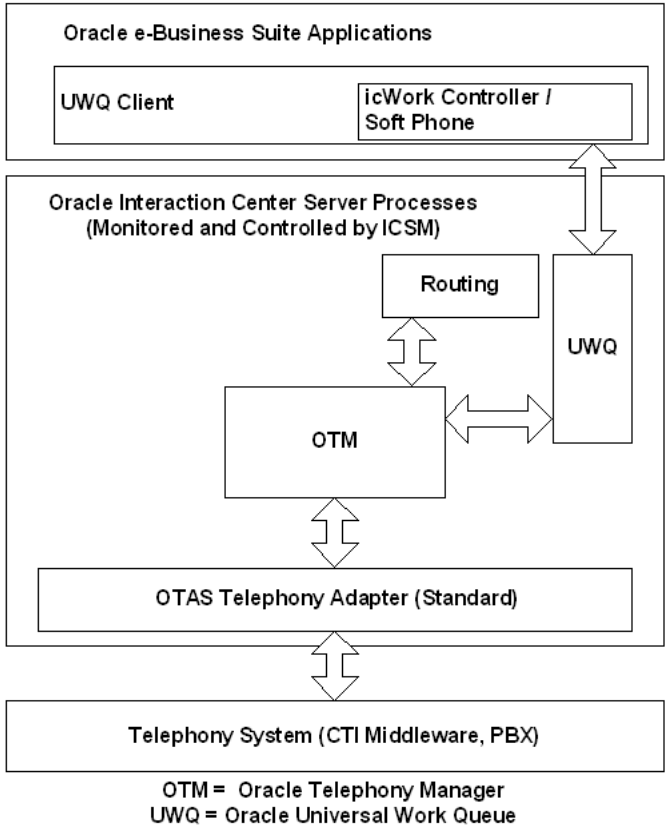
Feature	Standard Integration
Summary	■ Screen pop ■ Softphone Lite ■ Passive (ACD routing)
Screen pop	Screen pop based on: ■ PBX Application Data ■ ANI ■ DNIS
CTI Functionality	■ Transfer ■ Conference Use the physical telephone to do all other telephony functions.
Softphone	Softphone Lite: ■ Transfer button ■ Conference button
Call Routing	ACD does the routing (Passive Mode)
Call Classification	Classification based on: ■ ANI ■ DNIS ■ PBX Application Data

Table 1–2 Standard Integration Features (Cont.)

Feature	Standard Integration
Outbound Dialing Modes	■ Progressive ■ Preview
Interaction History	Moderate Interaction History: Media LCS for Trn/Conf
Web Callback	Optional

As the following figure illustrates, in Standard Integration the Telephony Adapter Server is the intermediary between the telephony system and Oracle Advanced Inbound. Oracle Universal Work Queue is the intermediary between Oracle Advanced Inbound and the agent's client system, including the icWork Controller and the softphone.

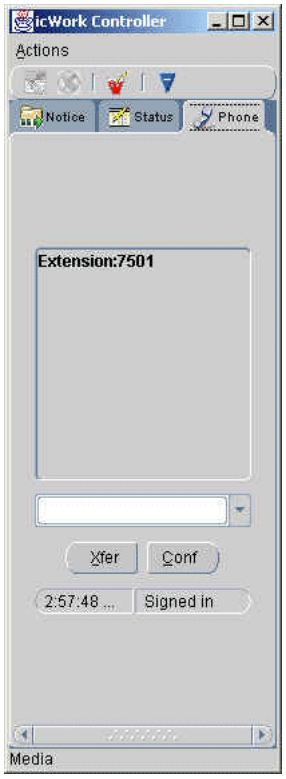
Figure 1–3 Standard Integration Architecture



Standard Integration Softphone

The following illustration shows the icWorkController softphone with lightweight functions, displaying the extension number and buttons for transfer and conference.

Figure 1–4 Standard Integration Softphone



See Also

- Chapter 2, "Concepts"
- Chapter 3, "Development Environment"
- Chapter 4, "Oracle Telephony Adapter SDK Components"
- Chapter 5, "Developing the Telephony Adapter"
- Chapter 7, "Testing the Telephony Adapter"

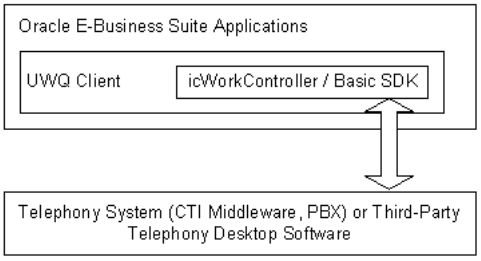
- Chapter 8, "Deploying the Telephony Adapter"

1.2.3 Basic Integration

Basic Integration is the lightest level of integration to Oracle Advanced Inbound. The API and the underlying infrastructure for Basic Integration are different from Standard Integration and Advanced Integration.

As the previous figure illustrates, unlike Standard and Advanced Integrations, Basic Integration does not require any Advanced Inbound or Universal Work Queue server processes. Oracle e-Business Applications and the Telephony System communicate through an Oracle Universal Work Queue Client using HTTP and XML.

Figure 1–5 Basic Integration Architecture



The following table lists and describes the features of Basic Integration.

Table 1–3 Basic Integration Features

Features	Basic Integration
Screen pop	Screen pops based on: ■ ANI ■ DNIS ■ Other interaction keys. See Section 9.4.4, "Additional Interaction Keys".

Table 1–3 Basic Integration Features (Cont.)

Features	Basic Integration
CTI Functionality	■ Login / Logout ■ Ready / Not Ready ■ Wrap Up ■ Make Call Use the physical telephone to do all other telephony functions.
Softphone	Basic Panel
Call Routing	ACD does the routing (Passive Mode)
Call Classification	(Optional) By way of third-party field passed to Basic client
Outbound Dialing Modes	■ Manual. ■ Preview mode is supported in implementations in which the option for Advanced Outbound in preview mode is elected. Support is by way of the Oracle Universal Work Queue server and the Interaction Center Server Manager. With Basic Integration, Advanced Outbound in preview mode is the only dialing mode available. Starting the Universal Work Queue server for preview mode does not result in progressive or predictive mode.
Interaction History	Limited Interaction History: ■ Number of calls per site ■ Number of calls per agent ■ Talk time
Web Callback	Not Available

Basic Integration Softphone

Basic Integration softphone is Basic Panel, a tab on icWork Controller that agents can use to make calls, view the current telephony event, and optionally configure to display detailed events and logs. Basic Panel is shown in the following illustration.

Figure 1–6 Basic Integration Softphone

See Also

Chapter 9, "Basic Integration"

1.3 New in this Release

This release of Oracle Telephony Adapter SDK includes the following new feature:

Multi-Site Telephony Adapter Implementation.

1.4 Approaching an SDK Project

Use the following outline in planning an SDK project.

1. Qualify a project for SDK implementation. See Appendix A, "SDK Qualification Worksheet".
2. Scope the extent of the project and brief the programmer. See Appendix B, "SDK Scope Analysis".
3. Develop the Telephony Adapter. See Chapter 5, "Developing the Telephony Adapter" for the C and Javadoc APIs.
4. Unit test the adapter and perform end-to-end integration testing. See Appendix C, "Telephony Adapter Test Cases" and Chapter 7, "Testing the Telephony Adapter".
5. Analyst review of the deliverables and sign off.
6. Deploy the Telephony Adapter, see Chapter 8, "Deploying the Telephony Adapter".

2

Concepts

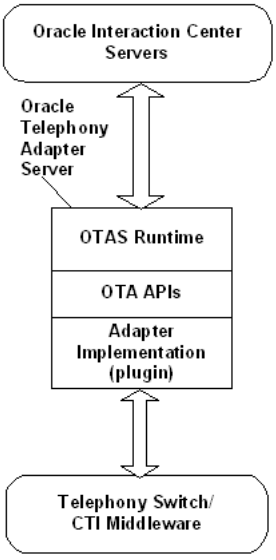
This section explains key concepts of Oracle Telephony Adapter SDK.

This section includes the following topics:

2.1 Architecture

The Oracle Telephony Adapter Server consists of a communications core (Oracle Telephony Adapter Server Runtime), the Oracle Telephony Adapter APIs, and the adapter code that implements and writes to the Oracle Telephony Adapter APIs.

Figure 2–1 Oracle Telephony Adapter Server Architecture



As the previous figure illustrates, the Oracle Telephony Adapter Server consists of Oracle Telephony Adapter Server Runtime, Oracle Telephony Adapter Server APIs, and the adapter implementation (plug-in). The Oracle Telephony Adapter Server connects the telephony switch and CTI middleware with Oracle Interaction Center.

2.2 Adapter Implementation

The adapter implementation is platform-specific custom code that maps telephony platform (PBX/ACD and CTI middleware) APIs to Oracle Telephony Adapter APIs.

2.3 Oracle Telephony Adapter API Objects

Oracle Telephony Adapter APIs specify three objects that the adapter code must implement and through which it must communicate. The objects are TeleDeviceFactory, TelesetDevice and RoutePointDevice.

2.3.1 TeleDeviceFactory

Oracle uses the TeleDeviceFactory object to allocate and deallocate switch resources. In the current release, only the Teleset and Route Point resources can be allocated.

2.3.2 TelesetDevice

A teleset device represents a physical telephone (currently only digital telesets, not analog telesets) that is used by an agent. Agents log into their telesets and use this object to perform various call control functions, such as placing a call, putting a call on hold and hanging up a call. The teleset device also relays switch events signifying state changes and call progression from the telephony platform to Oracle Interaction Center.

For Advanced Integration, the teleset device has the following characteristics:

- Each teleset is displayed as three lines on the Oracle softphone.
- Each line can hold one call.
- Depending on the telephony platform, one key on the physical teleset maps to one or more lines on the Oracle softphone. For example, for Avaya Definity one physical teleset is programmed with three keys that share the same "station number." In this case, on the Oracle softphone one line represents one key of the Avaya physical teleset. For Nortel Meridian, one physical teleset is programmed with two keys, Personal DN and ACD DN. In this case, on the softphone, line one represents the Personal DN key, line two represents the ACD DN key, and line three represents a consultation call placed from either key.
- Most events that are fired to the Oracle Interaction Center must specify the line on which the event occurs (line indexes 0, 1 or 2).
- Most call control commands are performed on the line that is specified by the API call, such as holdCall. The API calls makeCall and consultationCall can return events on a different line.
- Only one consultation call can be placed at a time. Therefore, only one transfer or conference can be pending.

2.3.3 RoutePointDevice

A route point device represents a switch queue that can be either a controlled queue, in which Oracle Advanced Inbound actively routes calls, or a monitored queue, in which Oracle Advanced Inbound only keeps track of the queued calls while the switch does the actual call routing.

The route point device has the following characteristics:

- Each route point has a queue of pending calls.
- Each call in the queue is identified by a call identifier, usually the call ID that is assigned by the telephony platform.
- The call identifier is used by call control commands. Any events that are fired must contain the call identifier.
- If the call ID changes due to a blind transfer, then the corresponding call identifier should not change.

2.4 Call Data

Call data is any data that is associated with a call, specifically the ANI, DNIS and any IVR data. This data is used by Oracle Interaction Center to properly classify and route the call and is also the basis for a screen pop. The adapter expects call data from the telephony platform at the start of a call or soon after the start.

2.5 Media Item ID

Oracle Interaction Center keeps track of logical interactions that are identified by a Media Item Identifier (Media Item ID). Although a switch-specific call ID may change as the call is transferred, the Media Item ID must remain the same. When the Media Item ID is assigned by Oracle, it is the responsibility of the implementer to ensure that it is propagated as the call moves from agent to agent. If the call ID on a line changes due to a transfer or conference completion, then the corresponding Media Item ID does not change.

2.5.1 Media Item ID Model

The Media Item ID model has the following characteristics.

Note: The following examples do not show all events.

Table 2–1 Oracle Interaction Center Actions

Oracle Interaction Center Actions	Adapter Server Actions/States	Oracle Interaction Center States
A: makeCall (MI: 1, Dest: B)[4]	A: dial (Dest: B) map resultant call id 1 to MI 1[1]	
	A: beginCallEvent (MI: 1, line: 0)[2]	A: line 0 has MI 1[1]
	B: beginCallEvent (MI: 1, line: 0)[2]	B: line 0 has MI 1[1]
A: releaseCall (line: 0)	A: hangup (call id 1)	
	A: callReleasedEvent (line: 0)	A: line 0 is clear
	B: callReleasedEvent (line: 0)	B: line 0 is clear

Table 2–2 Customer Actions

Customer Actions	Adapter Server Actions/States	Oracle Interaction Center States
X: calls A	A: beginCallEvent (MI: n/a, line:0) Oracle Telephony Adapter Server does not know MI[1]	A: line 0 has MI 2[4]: OIC assigns Media Item ID to call
X: hangup call	A: callReleasedEvent (line:0)	A: line 0 is clear

Table 2–3 Customer/ Oracle Interaction Center Actions

Customer/OIC Actions	Adapter Server Actions/States	Oracle Interaction Center States
X: calls A	A: beginCallEvent (MI: n/a, line:0) // call id 2, Oracle Telephony Adapter Server does not know MI[1]	A: line 0 has MI 3[3]: OIC assigns Media Item ID to call
A: consult (MI[1]: 3, line: 0, Dest: B)[4]	A: consult (call id: 2, Dest: B) // map call id 2 to MI[1] 3 // map consult call id 3 to MI 3	
	A: beginCallEvent (MI: 3, line: 1)[2]	A: line 1 has MI 3
	B: beginCallEvent (MI: 3, line: 0)[2]	B: line 0 has MI 3
A: transfer (line: 1, old line: 0)	A: transfer (call ID 3, call ID 2)	
	A: callReleasedEvent (line: 0)	A: line 0 is clear

Table 2–3 Customer/ Oracle Interaction Center Actions (Cont.)

Customer/OIC Actions	Adapter Server Actions/States	Oracle Interaction Center States
	A: callReleasedEvent (line: 1)	A: line 1 is clear
	B call ID changes (ABA model)[5]	No event to OIC

Footnotes

1. Media Item ID
2. If an incoming call already has a Media Item ID, then it must be passed to the receiving device.
3. If an incoming call does not have a Media Item ID, then Oracle Interaction Center assigns a new Media Item ID and uses it with any future consultation calls.
4. New calls are assigned a new Media Item ID when the dial command is made.
5. There is a one-to-many relationship between Media Item ID and switch-specific call IDs. If the call ID changes while the actual call is transferred or conferenced, then the Media Item ID should remain the same.

3

Development Environment

Oracle Telephony Adapter SDK has the following requirements.

3.1 Minimum Hardware Requirements

Oracle Telephony Adapter SDK has the following hardware requirements.

Development Machine

- CPU: 500MHz
- Memory (RAM): 256MB
- Disk space: 10G
- Operating Systems: Windows NT 4.0 SP6 or Windows NT 2000 when using the C API, and Linux and UNIX when using the Java API
- Plus any required hardware for telephony connectivity

Telephony Hardware

- PBX
- Telesets

3.2 Minimum Software Requirements

Oracle Telephony Adapter SDK has the following software requirements.

General

- Java Development Kit version 1.1.8, 1.2 or 1.3

- (Optional) An Integrated Development Environment (IDE) for Java such as Oracle JDeveloper

SDK for Java

Any third-party CTI Java libraries

SDK for C

- Any third-party CTI software for Windows NT and Windows 2000
- C compiler and make utility
- (Optional) An Integrated Development Environment (IDE) for Windows such as Visual C^{++}

4

Oracle Telephony Adapter SDK Components

Oracle Telephony Adapter SDK provides APIs for the C and Java programming languages.

Oracle Telephony Adapter Java APIs provide an open interface for consultants and switch and CTI middleware vendors to implement an integration to Oracle Interaction Center (OIC) for any telephony platforms using the Java programming language.

Oracle Telephony Adapter C APIs enable programmers to integrate switches and CTI middlewares using the C programming language.

The Oracle Telephony Adapter C API consists of the following APIs and functions:

- Header files for TeleDeviceFactory (TeleDeviceFactoryAPI.h), TelesetDevice (TelesetDeviceAPI.h) and RoutePointDevice (RoutePointDeviceAPI.h).
- EventListener API (TelesetEventListenerAPI.h and RouteEventListenerAPI.h)
- Common utilities functions , typedefs and constants (occt_pub.h , occt_md.h)
- OHashtable functions (Hashtable.h)

The C adapter implementer implements all methods that are defined in the Header files.

This section contains the following topics:

4.1 Installing Oracle Telephony Adapter SDK

Use the following instructions to install the Oracle Telephony Adapter SDK.

Prerequisite

Java Development Kit or Java Runtime Environment must be installed before you install Oracle Telephony Adapter SDK. Oracle recommends that you use JDK version 1.3.

Downloading Oracle Telephony Adapter SDK

The "Patch for Telephony Adapter Software Development Kit (SDK)" is available in Oracle*MetaLink* as Patch 2621791.

Use the following steps to download Oracle*MetaLink* Patch 2621791.

1. In the side navigation bar, select **Patches**.

   The Select a Patch Search Area page appears.

2. Click **New MetaLink Patch Search**.

   The Simple Search page appears.

3. Enter `2621791` in the blank text field.

4. Click **Go**.

5. The Patch 2621791 page appears.

6. Click **Download**.

   The download process begins.

Note: Before you extract the downloaded Zip file, in the WinZip Extract window make sure to check "Use folder names" (or a similar option) so that the files extract into the specified directory structure. See the following section "Unpacking."

Unpacking

Use a file extraction program to unzip p2621791_11i_WINNT.zip to a directory path that does not contain white spaces. After you unzip p2621791_11i_WINNT.zip, edit the file oracle/apps/cct/bin/sdkenv.cmd and modify the value of the SDK_JRE_HOME environment variable to the location where JDK is installed. The directory path where the JDK is installed should not contain blank spaces.

4.2 Package Contents

Oracle Telephony Adapter SDK package p2621791_11i_WINNT.zip contains the following files.

Table 4–1 Contents of Zip File

Directory	Description
bin	Scripts for starting Adapter Server and Test Tools
lib	Java and C runtime libraries supplied by Oracle
docs	Documentation
c	SDK for C
c/include	SDK for C: C include files
c/lib	SDK for C: C libraries supplied by Oracle for C adapter development
c/impl	SDK for C: Directory for storing C adapter implementation
c/sample	SDK for C: Sample C code
java	SDK for Java
java/javadoc	SDK for Java: Javadoc
java/classes	SDK for Java: Directory for storing Java adapter implementation class files
java/sample	SDK for Java: Sample code
3rdParty	Storage for Java and C libraries provided by switch and CTI middleware vendor
c/ao/include	AO SDK Extension: C include files
c/ao/lib	AO SDK Extension: C libraries supplied by Oracle for AO SDK Extension
c/ao/sample	AO SDK Extension: Sample AO All C Implementation
java/sample/advanced	SDK for Java: Sample Code for Advanced Integration
java/sample/ao	AO SDK Extension: Sample AO All Java Implementation
java/sample/basic	SDK for Java: Sample Code for Basic Integration
java/sample/standard	SDK for Java: Sample Code for Standard Integration

4.2.1 SDK for Java

Developers can use the Oracle Telephony Adapter SDK for Java to implement the adapter by using the Java programming language. The SDK for Java should be used when the Java API is available from the target switch and CTI middleware to be integrated, such as CT Connect and JTAPI.

The adapter implemented using SDK for Java is packaged as JAR files.

4.2.2 SDK for C

Developers can use the Oracle Telephony Adapter SDK for the C programming language to implement the adapter on only the Windows NT and Windows 2000 platforms. The SDK for C should be used when the C API is the only available option from the switch and middleware to be integrated, such as Cisco ICM and TAPI. SDK for Windows NT and Windows 2000 is a set of C functions implementing integrations to the switch and CTI middleware. These functions are collectively packaged as an adapter.

The adapter implemented using SDK for Windows NT and Windows 2000 is packaged as a Windows DLL (Dynamically Loaded Library) file.

4.2.3 Oracle Telephony SDK Integrated Test Utility

Oracle Telephony SDK Integrated Test Utility is a user interface for running, configuring and testing telephony adapters.Developers can use this utility to test their adapter implementations in offline mode, which does not require an Oracle Application Database or Oracle Interaction Center servers. However, the Test Utility does require that the telephony adapter be connected to the telephony platform. The Test Utility is primarily used in the first phase of adapter development where developers can work on their own development workstations without connecting to an Oracle Application Database. The integrated test utility includes three components: Oracle Telephony Adapter Server, TelesetDevice Test Utility and Telephony Adapter Verification Tool.

4.2.3.1 Oracle Telephony Adapter Server

The Test Utility contains the Oracle Telephony Adapter Server. Developers can use Oracle Telephony Adapter Server to run and test adapter implementations.

4.2.3.2 TeleDevice Test Utility

TeleDevice Test Utility provides a user interface for testing TelesetDevice and RoutePointDevice objects. Developers can use this utility to launch the softphone and to drive test cases.

4.2.3.3 Telephony Adapter Verification Tool

Telephony Adapter Verification Tool provides developers with a means to verify their adapter implementation against a predefined set of telephony test cases.

5

Developing the Telephony Adapter

The section explains the development process of Standard and Advanced Integration of Oracle Telephony Adapter SDK and includes the following topics:

5.1 Advanced Integration

Advanced Integration requires implementing all the methods and interfaces that are specified by the Oracle Telephony Adapter SDK. The runtime system requires that all Oracle Interaction Center server processes are configured and running.

For a list of functions that Advanced Integration supports, see Section 1.2.1, "Advanced Integration".

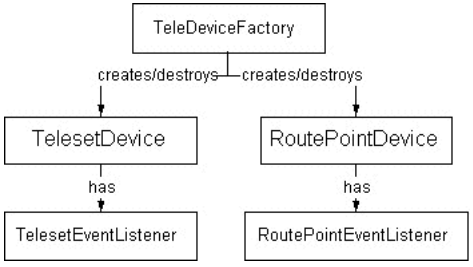
5.1.1 Advanced Integration Major Interfaces

The Advanced Integration of Oracle Telephony Adapter SDK has the following major interfaces:

- TeleDeviceFactory
- TelesetDevice
- RoutePointDevice
- TelesetEventListener
- RoutePointEventListener

The following diagram shows that in the major interfaces of Oracle Telephony Adapter SDK, TeleDeviceFactory creates and destroys TelesetDevice, which in turn has TelesetEventListener. TeleDeviceFactory also creates and destroys RoutePointDevice, which has RoutePointEventListener.

Figure 5–1 Major Interfaces of Advanced Integration



The Advanced Integration of Oracle Telephony Adapter SDK has the following major interfaces.

SDK for Java

- Create classes that extend Java abstract classes AbstractTelesetDevice and AbstractRoutePointDevice, and create classes that implement Java interface TeleDeviceFactory.
- Package all classes as a JAR file.

SDK for C

- Create implementation for C functions defined in TeleDeviceFactoryAPI.h, TelesetDeviceAPI.h and RoutePointDeviceAPI.h.
- Package all functions in a DLL file.

5.2 Standard Integration

The Standard Integration of Oracle Telephony Adapter includes a subset of functions of Advanced Integration. Standard Integration requires a subset of TelesetDevice methods and the full TelesetEventListener model that is to be implemented. Standard Integration does not require implementation of RoutePointDevice.

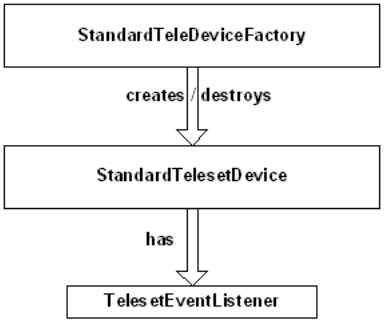
For a list of functions that Standard Integration supports, see Section 1.2.2, "Standard Integration".

This section contains the following topics:

5.2.1 Standard Integration Major Interfaces

The following diagram shows that in the major interfaces of Standard Integration, StandardTeleDeviceFactory creates and destroys StandardTelesetDevice, which has TelesetEventListener.

Figure 5–2 Major Interfaces of Standard Integration



The Standard Integration of Oracle Telephony Adapter SDK has the following major interfaces.

SDK for Java

- Develop Java classes that extend StandardTelesetDevice and StandardTeleDeviceFactory.
- Same as for Advanced Integration, package all classes in a JAR file.

SDK for C

- Create implementation for C functions defined in TeleDeviceFactoryAPI.h and TelesetDeviceAPI.h marked as required by Standard Integration.
- Same as for Advanced Integration, package all functions as a DLL file.

5.2.2 Implementing StandardTeleDeviceFactory

StandardTeleDeviceFactory is an abstract class that defines a subset of the methods of TeleDeviceFactory. Adapter developers must extend StandardTeleDeviceFactory to create their own factory class.

5.2.2.1 Methods to Implement

- init
- createTelesetDevice
- destroyTeleDevice

See Also

- Section 1.2.2, "Standard Integration"
- Oracle Telephony Adapter SDK Javadoc

5.2.3 Implementing StandardTelesetDevice

StandardTelesetDevice is an abstract class that defines a subset of the methods of TelesetDevice. Adapter developers must extend StandardTelesetDevice to create their own TelesetDevice class.

5.2.3.1 Methods to Implement

- loginAgent
- logoutAgent
- agentReady
- agentNotReady
- makeCall
- consultationCall
- completeTransfer
- completeConference

> **Note:** In addition to the above defined methods, adapter developers must implement all call events that are defined in TelesetEventListener. See Section 5.1, "Advanced Integration".

See Also

- Section 1.2.2, "Standard Integration"
- Oracle Telephony Adapter SDK Javadoc

5.3 Overview of the Development Process

The following list of procedures describes a high-level overview of developing Oracle Telephony Adapter SDK. Steps 2 and 7 are iterative.

1. Install Oracle Telephony Adapter SDK.
2. Develop.
3. Build.
4. Run Oracle Telephony Adapter Server.
5. Test and redevelop.
6. Deploy the adapter implementation to the interaction center.
7. Integrate the test with the interaction center and redevelop.
8. Produce.

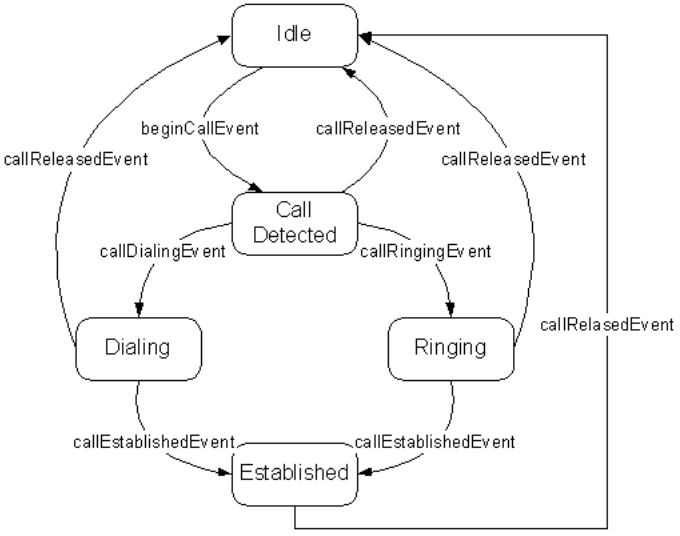
5.4 Call Scenarios

The following call scenarios describe typical call flow events for API calls on the TelesetEventListener interface.

5.4.1 Basic Call Flow

The following diagram illustrates the basic call event flow for a single teleset line. The initial state is idle. When a call comes in and the corresponding API event listener is called, the event initiates a transition that causes the line to be in a new state. Each transition is initiated by an API call to the TelesetEventListener interface, such as beginCallEvent, CallReleasedEvent, callDialingEvent, and callEstablishedEvent. The callDialingEvent initiates outbound calls, and the callRingingEvent initiates incoming calls.

Figure 5–3 Basic Call Flow

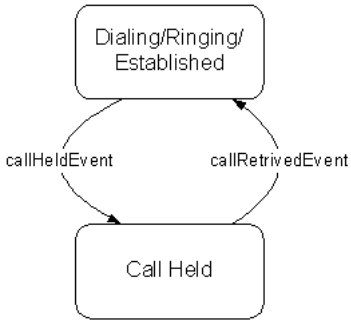


Basic Call Event Flow for a Line

5.4.2 Call Held Event Flow

The following diagram illustrates a call in which either party can put the other party on hold and then retrieve the call. The call can be between customer and agent or agent to agent. When a call is in a dialing, ringing or established state, the call can be placed on hold. The callHeldEvent is fired to the initiator. When the call is retrieved from hold, the callRetrievedEvent is fired to the initiator.

Figure 5–4 Call Held Event Flow



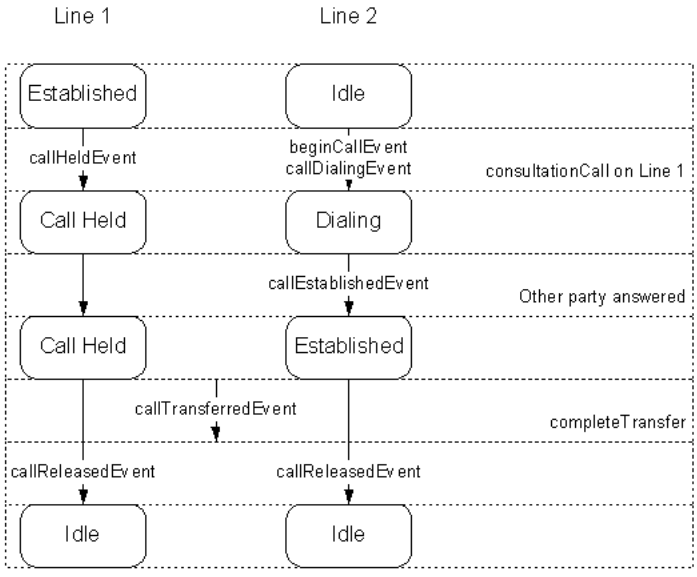
Call Held Event Flow

5.4.3 Two-Step Transfer Event Flow

The following diagram illustrates that in a consult transfer, the controlling agent can transfer the call to another agent and can consult with the other agent before completing the transfer.The controlling agent receives a callHeldEvent on Line 1, and the beginCallEvent and callDialingEvent on Line 2. When the other agent answers, the controlling agent receives a callEstablishedEvent on Line 2. The controlling agent then completes the transfer of the call and receives a callTransferredEvent.After the transfer is complete, the controlling agent receives a callReleasedEvent on Line 1 and another callReleasedEvent on Line 2.

> **Note:** With some switch and CTI middleware combinations, the initial and final call IDs are the same. For example, in the sequence "ABA" the initial call ID is A, the call ID of the consultation call is B, and the final call ID of the transferred call is again A. With some other switch and CTI middleware combinations, the transferred call retains the call ID of the consultation call: "ABB."

Figure 5–5 Two-Step Transfer Event Flows

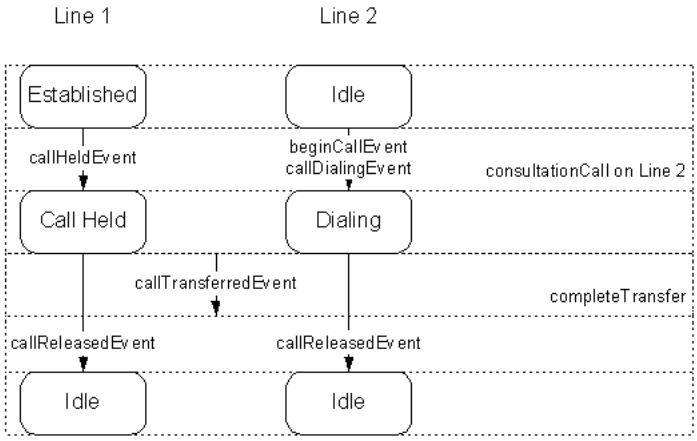


Two-Step Call Transfer Event Flow

5.4.4 Blind Transfer Event Flow

The following diagram illustrates that in the blind transfer scenario, an agent transfers a call on Line 1 to another agent without consulting with that other agent. The controlling agent receives callHeldEvent on Line 1 and the beginCallEvent and callDialingEvent on Line 2. The controlling agent then completes the transfer of the call and receives a callTransferredEvent. After the transfer is complete, the controlling agent receives a callReleasedEvent on Line 1 and another callReleasedEvent on Line 2.

Figure 5–6 Blind Transfer Event Flow

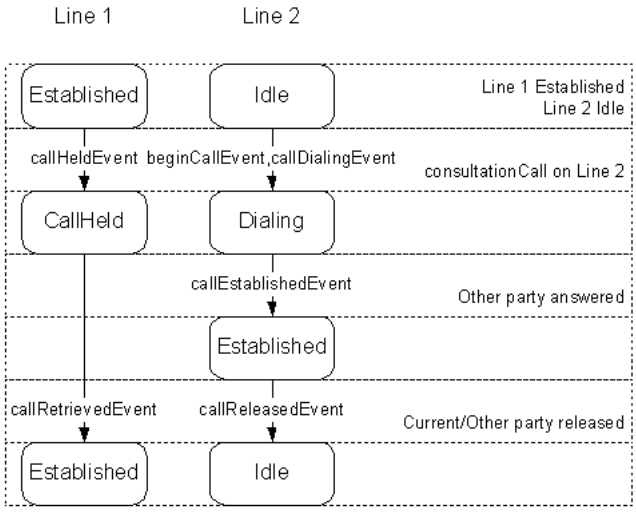


Two-Step Blind Transfer Event Flow

5.4.5 Consultation Cancelled Event Flow

The following diagram illustrates that in a consult transfer, the controlling agent can transfer the call to another agent and can consult with the other agent before completing the transfer.The controlling agent receives a callHeldEvent on Line 1, and the beginCallEvent and callDialingEvent on Line 2. When the other agent answers, the controlling agent receives a callEstablishedEvent on Line 2. The controlling agent then cancels the transfer and receives a callReleasedEvent on Line 2 and a callRetrievedEvent on Line 1. At this point, the agent returns to the original caller.

Figure 5–7 Consultation Cancelled (Transfer or Conference Call)



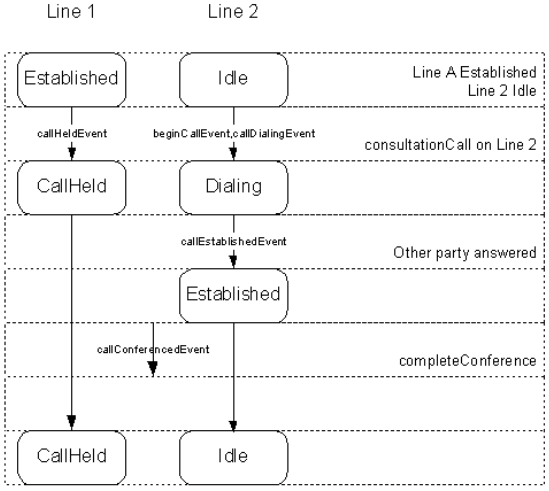
Consultation Cancelled (Transfer or Conference)

5.4.6 Two-Step Conference Event Flow with Inactive Line

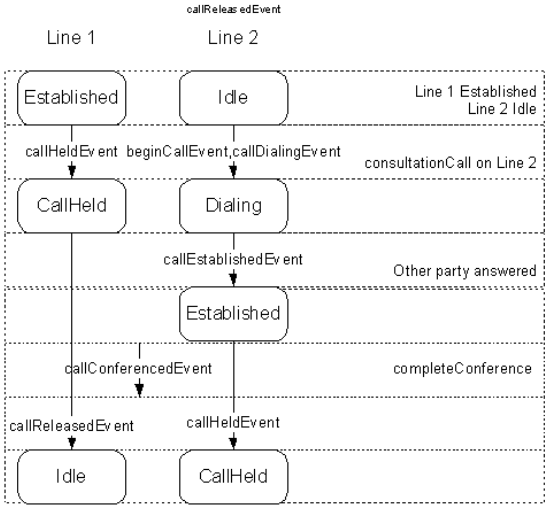
The following diagram illustrates that in a consult conference, the controlling agent can conference another agent to the call and can consult with the other agent before completing the conference. The controlling agent receives a callHeldEvent on Line 1, and the beginCallEvent and callDialingEvent on Line 2. When the other agent answers, the controlling agent receives a callEstablishedEvent on Line 2. The controlling agent then completes the conference and receives a

callConferencedEvent.After the conference is complete, the controlling agent receives a callReleasedEvent on Line 1 or Line 2.

Figure 5–8 Call Conference Event Flows with Conference Line Inactive



Call Conference Event Flow (ABA, Conference Line Inactive)

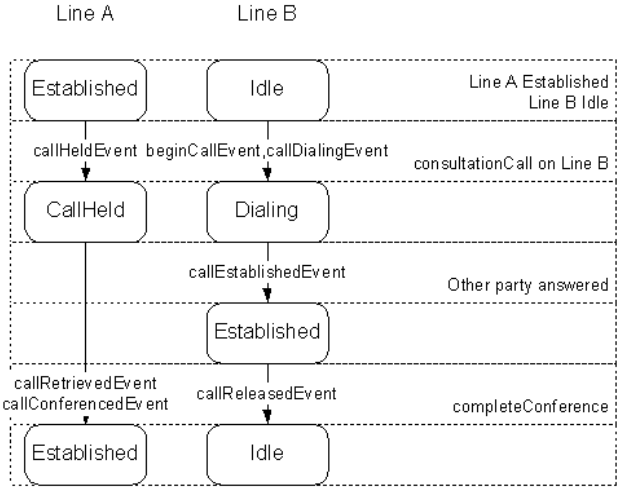


Call Conference Event Flow (ABB, Conference Line Inactive)

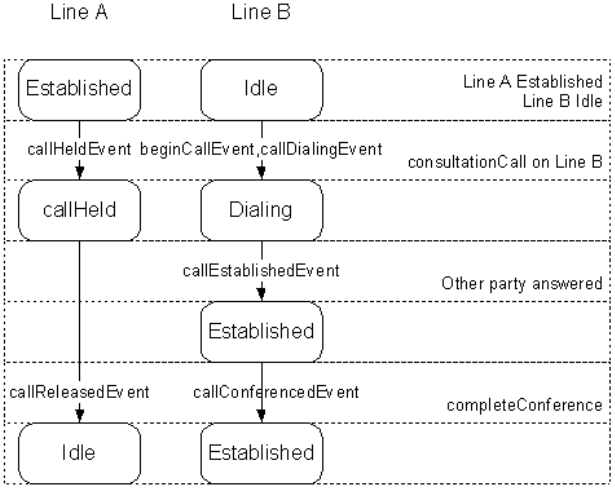
5.4.7 Call Conference Event Flow

The following diagram illustrates how the Call Conference Event Flow is similar to the Two-Step Conference Event Flow with Inactive Line. In a standard Call Conference Event Flow, the controlling agent places a consultation call to another party and can converse with the other party before completing the conference. After placing the consultation call, the controlling agent receives a callHeldEvent on Line 1 (the original call), and beginCallEvent and callDialingEvents on Line 2 (the consultation call). When the other party answers, the controller receives a callEstablishedEvent on Line 2. At this point, if the controller completes the conference, a callReleasedEvent is received on one line and a callConferencedEvent on the other line. If the call is on hold, the callConferencedEvent implies that the call is retrieved.

Figure 5–9 Call Conference Event Flows with Conference Active Line



Call Conference Event Flow (ABA, Conference Line Active)



Call Conference Event Flow (ABB, Conference Line Active)

5.5 Oracle Telephony Adapter Life Cycle

This section describes the creation and destruction life cycle SDKs for Java and C for Oracle Telephony Adapter, and contains the following topics:

5.5.1 Oracle Telephony Adapter Server Startup

SDK for Java

- Oracle Telephony Adapter Server process starts.
- Oracle Telephony Adapter Server loads Adapter Configurations.
- Oracle Telephony Adapter Server locates the TeleDeviceFactory class name from configuration.
- Oracle Telephony Adapter Server loads the TeleDeviceFactory class and creates a new instance.
- Oracle Telephony Adapter Server calls TeleDeviceFactory.init() and passes all configuration data as a Hashtable.
- TeleDeviceFactory initializes itself appropriately with the configuration data.

SDK for C

- Oracle Telephony Adapter Server process starts.
- Oracle Telephony Adapter Server loads Adapter Configurations.
- Oracle Telephony Adapter Server loads the adapter DLL.
- Oracle Telephony Adapter Server calls factInit() function.
- Configuration data is available by way of C function occtProviderConfigGet.
- C adapter initializes itself appropriately with the configuration data.

5.5.2 TelesetDevice Life Cycle

5.5.2.1 Creation

SDK for Java

- Oracle Telephony Adapter Server creates a new TelesetDevice by calling the method TeleDeviceFactory.createTelesetDevice().
- TeleDeviceFactory returns a new instance of TelesetDevice.
- Oracle Telephony Adapter Server registers TelesetEventListener with the newly created TelesetDevice by calling the method TelesetDevice.addTelesetEventListener().
- TelesetDevice starts receiving API calls from Oracle Telephony Adapter Server and starts calling event methods on TelesetEventListener.

SDK for C

- Oracle Telephony Adapter Server creates a new OcctTelesetDevice internal structure.
- Oracle Telephony Adapter Server calls tsInit() function with the OcctTelesetDevice pointer.
- Adapter performs any initialization in tsInit() function.
- Adapter starts receiving API function calls (tsXXX()) and starts calling event functions (tsXXXEvent()).

5.5.2.2 Destruction

SDK for Java

- Oracle Telephony Adapter Server calls TeleDeviceFactory.destroyTeleDevice().
- Oracle Telephony Adapter Server unregisters TelesetEventListener with the TelesetDevice by calling the method TelesetDevice.removeTelesetEventListener().
- TeleDeviceFactory performs any cleanup and frees resources allocated to TelesetDevice.
- TeleDevice stops receiving API calls from Oracle Telephony Adapter Server and stops calling event methods on TelesetEventListener.

- TeleDevice object can then be garbage collected by Java Virtual Machine.

SDK for C

- Oracle Telephony Adapter Server calls tsDestroy() function.
- Adapter performs any cleanup and frees resources allocated to the TelesetDevice in tsDestroy().
- Adapter stops receiving API function calls (tsXXX()) and stops calling event functions (tsXXXEvent()).
- OcctTelesetDevice structure is freed and the pointer to OcctTelesetDevice can no longer be used.

5.5.3 RoutePointDevice Life Cycle

5.5.3.1 Creation

SDK for Java

- Oracle Telephony Adapter Server creates new a RoutePointDevice by calling the method TeleDeviceFactory.createRoutePointDevice().
- TeleDeviceFactory returns a new instance of RoutePointDevice.
- Oracle Telephony Adapter Server register RoutePointEventListener with the newly created RoutePointDevice by calling the method RoutePointDevice.addRoutePointEventListener().
- RoutePointDevice starts receiving API calls from Oracle Telephony Adapter Server and starts calling event methods on RoutePointEventListener.

SDK for C

- Oracle Telephony Adapter Server creates a new OcctRoutePointDevice internal structure.
- Oracle Telephony Adapter Server calls rpInit() function with the RoutePointDevice pointer.
- Adapter performs any initialization in rpInit() function.
- Adapter starts receiving API function calls (rpXXX()) and starts calling event functions (rpXXXEvent()).

5.5.3.2 Destruction

SDK for Java

- Oracle Telephony Adapter Server calls TeleDeviceFactory.destroyTeleDevice().
- Oracle Telephony Adapter Server unregister RoutePointEventListener with the RoutePointDevice by calling the method RoutePointDevice.removeRoutePointEventListener().
- TeleDeviceFactory performs any cleanup and freeing of resources allocated to the RoutePointDevice.
- RoutePointDevice stops receiving API commands from Oracle Telephony Adapter Server and stops calling event methods on RoutePointEventListener.
- RoutePointDevice object can then be garbage collected by Java Virtual Machine.

SDK for C

- Oracle Telephony Adapter Server calls rpDestroy() function.
- Adapter performs any cleanup and frees resources allocated to the TelesetDevice in rpDestory().
- Adapter stops receiving API function calls (rpXXX()) and stops calling event functions (rpXXXEvent()).
- OcctRoutePointDevice structure is freed and the pointer to OcctRoutePointDevice can no longer be used.

5.5.4 Oracle Telephony Adapter Server Shutdown

- Oracle Telephony Adapter Server calls TeleDeviceFactory.destroyTeleDevice(), one for each TeleDevice that is created, to clean up any existing TeleDevice objects.
- Oracle Telephony Adapter Server process shuts down.

5.6 Implementing TeleDeviceFactory

This section describes the methods available to implement TeleDeviceFactory, an interface implemented by third-party consultants or switch and CTI middleware providers. This implementation controls access to the creation and destruction of the TeleDevice and RoutePointDevice objects. The TeleDevice is the base class for both TeleDevice and RoutePointDevice objects.

5.6.1 Methods

Use the following methods in implementing TeleDeviceFactory.

5.6.1.1 init

Definition

- Initialize the provider with the provider-specific information it needs to start and connect to the switch and CTI middleware.
- Prerequisite: No TeleDevice objects instantiated.
- Expected: Can now instantiate TeleDevice objects.

Parameters

`data`: key-value pairs with the provider data.

SDK for Java

```
public abstract void init (Hashtable data) throws TeleDeviceException.
```

SDK for C

```
OEXPORT OcctCodes factInit ();
IMPORT_EXPORT char** occtProviderConfigGetKeys (int *len);
IMPORT_EXPORT char* occtProviderConfigGet (char* key);
```

5.6.1.2 createTelesetDevice

Definition

Create an object that can accept TelesetDevice commands and sends TelesetEventListener events to Oracle Interaction Center. This method is called once per teleset when this method returns the TelesetDevice object or an Exception.

- Prereq: TelesetDevice not already instantiated.
- Expected: TelesetDevice instantiated.

Parameters

- `extension1`: DN (Directory Number) of the first line of a teleset.
- `extension2`: DN of the second line of a teleset, can be a duplicate of extension1.

- `extension3`: DN of the third line of a teleset, can be a duplicate of extension1 or extension2. It can also be "*", which indicates a spillover line for both extension1 and extension2.

SDK for Java

```
public abstract TelesetDevice createTelesetDevice (String extension1, String
extension2, String extension3) throws TeleDeviceException.
```

SDK for C

```
OEXPORT OcctCodes tsInit (OcctTelesetDevice* ptr, const char* extension1, const
char* extension2, const char* extension3);
```

5.6.1.3 createRoutePointDevice

Definition

Create an object that can accept RoutePointDevice commands and that sends RoutePointEventListener events to Oracle Interaction Center. This method is called once per route point when this method returns the RoutePointDevice object or an Exception.

- Prereq: RoutePointDevice is not already instantiated.
- Expected: RoutePointDevice is instantiated.

Parameters

- `routePointExtension`: DN of the route point.
- `getRouteRequest`:
  - True if the provider should forward route requests to Oracle Telephony Manager.
  - False indicates that a route is being monitored passively.
- `data`: Provider-specific init data for the route point.

SDK for Java

```
public abstract RoutePointDevice createRoutePointDevice (String
routePointExtension, boolean getRouteRequest, Hashtable data) throws
TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes rpInit (OcctRoutePointDevice* ptr, const char*
routePointExtension, oboolean getRouteRequest, OHashtable* data);
```

5.6.1.4 destroyTeleDevice

Definition

Notification that Oracle Interaction Center is finished with the given TeleDevice. Provider can then reclaim resources as required. When this method returns, Provider may *not* send TeleEventListener events pertaining to the defunct TeleDevice until the TeleDevice is recreated.

- Prereq: CreateTeleDevice.
- Expected: Any resource allocated to teleDevice is freed.

Parameters

`teleDevice`: TeleDevice to be destroyed.

SDK for Java

```
public abstract void destroyTeleDevice (TeleDevice teleDevice) throws
TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes rpDestroy (OcctRoutePointDevice* ptr);
OEXPORT OcctCodes tsDestroy (OcctTelesetDevice* ptr);
```

5.7 Implementing TelesetDevice

This section contains the following topics:

5.7.1 TelesetDevice Interface

The TelesetDevice interface is used by Oracle Interaction Center to access a telephony resource of the switch or CTI middleware. The implementation of this interface is provided by the third-party consultant or switch or CTI middleware vendor. Each TelesetDevice is expected to have three lines. Each line is assumed to have only one call at any given time. Lines are 0-based indexed, for example, 0, 1, 2.

5.7.1.1 Methods

Most of the methods defined in TelesetDevice Interface use the line index to specify on which line the operation is targeted.

Use the following methods in implementing TelesetDevice.

answerCall

Description

Answer the given ringing call.

Parameters

`lineIndex`: The line on which the call is ringing (0-based).

Precondition

The specified line must be valid, and a valid call must be ringing on the specified line.

Outcome Success

send callEstablishedEvent

Outcome Failure

- Send errorEvent or throw TeleDeviceException
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_ANSWER_CALL

- Possible Error Code: ErrorCodes.FAILED , FUNCTION_NOT_SUPPORTED , NO_CURRENT_CALL , LINE_INDEX_OUT_OF_RANGE , LINE_ALREADY_IN_USE

Throws

TeleDeviceException -

SDK for Java

```
public void answerCall (int lineIndex)
        throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsAnswerCall (OcctTelesetDevice*, oint);
```

makeCall

Description

Place a call from the given teleset.

Precondition

There must be no call on the specified line, and the specified line must be valid.

Outcome Success

Send callDialingEvent.

Outcome Failure

- Send errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_RELEASE_CALL.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED, DESTINATION_INVALID, DESTINATION_BUSY, LINE_INDEX_OUT_OF_RANGE, LINE_ALREADY_IN_USE.

Parameters

- `mediaItemId`: New call appearances resulting from this command should return the given media item ID.
- `lineIndex`: Location from which to make the call (0-based).

- `destinationNumber`: The number to dial.

Throws

TeleDeviceException -

SDK for Java

```
public void makeCall (java.lang.String mediaItemId,
int lineIndex,
        java.lang.String destinationNumber)
        throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsMakeCall (OcctTelesetDevice*, const char*, oint, const
char*);
```

releaseCall

Description

Release (hang up) the given established call.

Precondition

There must be a valid call on the specified line.

Outcome Success

Send callReleasedEvent.

Outcome Failure

- Send errorEvent or throw TeleDeviceException
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_RELEASE_CALL.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED, LINE_INDEX_OUT_OF_RANGE, NO_CURRENT_CALL.

Parameters

`lineIndex`: Where call is (0-based).

Throws

TeleDeviceException -

SDK for Java

```
public void releaseCall (int lineIndex)
        throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsReleaseCall (OcctTelesetDevice*, oint);
```

holdCall

Description

Hold the given established call.

Precondition

There must be a valid call on the specified line.

Outcome Success

Send callHeldEvent.

Outcome Failure

- Send errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_HOLD_CALL.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED, LINE_INDEX_OUT_OF_RANGE, NO_CURRENT_CALL.

Parameters

`lineIndex`: Where the call is (0-based).

Throws

TeleDeviceException -

SDK for Java

```
public void holdCall (int lineIndex)
        throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsHoldCall (OcctTelesetDevice*, oint);
```

retrieveCall

Description
Retrieve the given held call.

Precondition
A valid call must be held on the specified line.

Outcome Success
Send callRetrievedEvent.

Outcome Failure
- Send errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_RETRIEVE_CALL.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED, LINE_INDEX_OUT_OF_RANGE, NO_CURRENT_CALL.

Parameters
`lineIndex`: Where call is (0-based).

Throws
TeleDeviceException -

SDK for Java
```
public void retrieveCall (int lineIndex)
       throws TeleDeviceException
```

SDK for C
```
OEXPORT OcctCodes tsRetrieveCall (OcctTelesetDevice*, oint);
```

consultationCall

Description
Place a consultation call.

Precondition
There must be a valid call on the specified line.

Outcome Success

Send callDialingEvent.

Outcome Failure

- Send errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_CONSULTATION_CALL.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED, DESTINATION_INVALID, DESTINATION_BUSY, LINE_INDEX_OUT_OF_RANGE, NO_CURRENT_CALL.

Parameters

- `mediaItemId`: New call appearances resulting from this command should return the given media item id.
- `lineIndex`: Where the base call is (0-based).
- `destinationNumber`: The number to dial.
- `consultationType`: See oracle.apps.cct.sdk.constants.ConsultationCallTypes.

Throws

TeleDeviceException -

See Also

ConsultCallTypes.

SDK for Java

```
public void consultationCall (java.lang.String mediaItemId,
            int lineIndex,
            java.lang.String destinationNumber,
            ConsultCallTypes consultationType)
            throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsConsultationCall (OcctTelesetDevice*, const char*, oint,
const char*, oint);
```

completeTransfer

Description

- Complete a call transfer.
- Precondition consultationCall is called previously.

Outcome Success

Send callTransferredEvent for held line, callReleasedEvent for active line.

Outcome Failure

- Send errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_COMPLETE_TRANSFER.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED, LINE_INDEX_OUT_OF_RANGE, NO_CURRENT_CALL.

Parameters

- `activeLineIndex`: Line index of active call (0-based).
- `heldLineIndex`: Line index of held call (0-based).

Throws

TeleDeviceException -

SDK for Java

```
public void completeTransfer (int activeLineIndex,
            int heldLineIndex)
        throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsCompleteTransfer (OcctTelesetDevice*, oint, oint);
```

completeConference

Description

Complete a call conference.

Precondition

consultationCall is called previously.

Outcome Success

Send callConferencedEvent, callReleasedEvent.

Outcome Failure

- Send errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_COMPLETE_CONFERENCE.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED, LINE_INDEX_OUT_OF_RANGE, NO_CURRENT_CALL.

Parameters

- `activeLineIndex`: Line index of active call (0-based).
- `heldLineIndex`: Line index of held call (0-based).

Throws

TeleDeviceException -

SDK for Java

```
public void completeConference (int activeLineIndex,
            int heldLineIndex)
        throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsCompleteTransfer (OcctTelesetDevice*, oint, oint);
```

blindTransfer

Description

Blind transfer the call to the destination number.

Precondition

There must be a valid call on the specified line.

Outcome Success

Send callTransferredEvent, callReleasedEvent.

Outcome Failure

- Send errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_BLIND_TRANSFER.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED, DESTINATION_INVALID, DESTINATION_BUSY, LINE_INDEX_OUT_OF_RANGE, NO_CURRENT_CALL.

Parameters

- `mediaItemId`: New call appearances resulting from this command should return the given media item id.
- `lineIndex`: Base call line index (0-based).
- `destinationNumber`: The number to dial.

Throws

TeleDeviceException -

SDK for Java

```
public void blindTransfer (java.lang.String mediaItemId,
            int lineIndex,
            java.lang.String destinationNumber)
        throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsBlindTransfer (OcctTelesetDevice*, const char*, oint, const
char*);
```

swapWithHeld

Description

Hold the active call, retrieve the held call.

Precondition

There must be valid calls on the lines specified.

Outcome Success

Send callHeldEvent and callRetrievedEvent for the corresponding calls.

Outcome Failure

- Send errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_SWAP_WITH_HELD.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED, LINE_INDEX_OUT_OF_RANGE, NO_CURRENT_CALL.

Parameters

- `activeLineIndex`: Where the active call is (0-based).
- `heldLineIndex`: Where the held call is (0-based).

Throws

TeleDeviceException -

SDK for Java

```
public void swapWithHeld (int activeLineIndex,
            int heldLineIndex)
        throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsSwapWithHeld (OcctTelesetDevice*, oint, oint);
```

sendDtmf

Description

Send DTMF (Dual Tone Multi-Frequency) dial tones to the call.

Precondition

There must be a valid call on the specified line.

Outcome Success

DTMF is sent to middleware.

Outcome Failure

- Send errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_SEND_DTMF.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED, LINE_INDEX_OUT_OF_RANGE, NO_CURRENT_CALL.

Parameters

- `lineIndex`: Call line index (0-based).
- `dtmfDigits`: The DTMF dial tones to send.

Throws

TeleDeviceException -

SDK for Java

```
public void sendDtmf (int lineIndex,
           java.lang.String dtmfDigits)
       throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsSendDtmf (OcctTelesetDevice*, oint, const char*);
```

loginAgent

Description

Login the agent to the teleset.

Precondition

Agent is not logged on.

Outcome Success

Send agentLogintEvent.

Outcome Failure

- Send errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_LOGIN_AGENT.

- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED, AGENT_INVALID, AGENT_PASSWORD_INVALID, TELESET_IN_USE.

Parameters

- `acdAgentId`: Switch-specific agent id.
- `acdAgentPassword`: Switch-specific agent password.
- `acdQueue`: Switch-specific agent acd queue.

Throws

TeleDeviceException -

SDK for Java

```
public void loginAgent (java.lang.String acdAgentId,
            java.lang.String acdAgentPassword,
            java.lang.String acdQueue)
        throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsLoginAgent (OcctTelesetDevice*, const char*, const char*,
const char*);
```

logoutAgent

Description

Logout the agent from the teleset.

Precondition

Agent is logged on.

Outcome Success

Send agentLogoutEvent.

Outcome Failure

- Send errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_LOGOUT_AGENT.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED, AGENT_PASSWORD_INVALID.

Parameters

- `acdAgentId`: Switch-specific agent id.
- `acdAgentPassword`: Switch-specific agent password.
- `acdQueue`: Switch-specific agent acd queue.

Throws

TeleDeviceException -

SDK for Java

```
public void logoutAgent (java.lang.String acdAgentId,
           java.lang.String acdAgentPassword,
           java.lang.String acdQueue)
       throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsLogoutAgent (OcctTelesetDevice*, const char*, const char*,
const char*);
```

addTelesetEventListener

Description

Add a TelesetEventListener to the listener list.

Parameters

`eventListener`: The TelesetEventListener interested in this TelesetDevice's events.

SDK for Java

```
public void addTelesetEventListener (TelesetEventListener eventListener)
```

SDK for C

Not applicable.

removeTelesetEventListener

Description

Remove the specified listener from the listener list.

Parameters

`eventListener`: The TelesetEventListener to remove.

SDK for Java

```
public void removeTelesetEventListener (TelesetEventListener eventListener)
```

SDK for C

Not applicable.

agentNotReady Method

Definition

- Set the agent state to "Do not route phone calls to me."
- Prereq: agentLoginEvent.
- Expected: agentNotReadyEvent.

Parameters

Not available.

SDK for Java

```
public abstract void agentNotReady () throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsAgentNotReady (OcctTelesetDevice* ptr);
```

agentReady Method

Definition

Set the agent state to "route phone calls to me."

Precondition

Agent is logged on.

Outcome Success

Send agentReadyOnEvent.

Outcome Failure

- Send errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_TELESET_AGENT_READY.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED.

Throws

TeleDeviceException -

SDK for Java

```
public void agentReady ()
        throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes tsAgentReady (OcctTelesetDevice*);
```

5.7.2 TelesetEventListener Interface

Third-party consultants or switch or CTI middleware providers use the TelesetEventListener interface to notify Oracle Interaction Center of teleset events. Oracle Telephony Manager gives the object that implements this interface to the TelesetDevice object for event reporting.

5.7.2.1 Methods

Use the following methods to implement TelesetEventListener.

beginCallEvent

Definition

This must be the first event for a new call for a particular TelesetDevice. No teleset state is associated with this event.

Parameters

- `mediaItemId`: Media item associated with call.
- `mediaItemIdComplete`: True if adapter has valid mediaItemId in this event , false if adapter might call updateMediaItemId
- `lineIndex`: Location of new call

- `ani`:
- `dnis`:
- `data`: IVR and other data associated with call.
- `dataComplete`: True if the adapter has all the data associated with this call, false if the adapter might call updateCallData.

SDK for Java

```
public void beginCallEvent (java.lang.String mediaItemId,
boolean mediaItemIdComplete,
           int lineIndex,
           java.lang.String ani,
           java.lang.String dnis,
           java.util.Hashtable data,
           boolean dataComplete)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsBeginCallEvent (OcctTelesetDevice*, const char*,
oboolean, oint, const char*, const char*, OHashtable*, oboolean);
```

callDialingEvent

Definition

- Call is dialing.
- Previous event expected: beginCallEvent.
- Next event expected: callEstablishedEvent or callReleasedEvent.

Parameters

`lineIndex`: Location of the call

SDK for Java

```
public void callDialingEvent (int lineIndex)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsCallDialingEvent (OcctTelesetDevice*, oint);
```

callRingingEvent

Definition

- Call is ringing.
- Previous event expected: beginCallEvent.
- Next event expected: callEstablishedEvent or callReleasedEvent.

Parameters

`lineIndex`: Location of the call.

SDK for Java

```
public void callRingingEvent (int lineIndex)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsCallRingingEvent (OcctTelesetDevice*, oint);
```

callEstablishedEvent

Definition

- Call is established. The involved parties are connected.
- Previous event expected: beginCallEvent, callDialingEvent, callRingingEvent.
- Next event expected: callReleasedEvent, callHeldEvent.

Parameters

`lineIndex`: Location of the call.

SDK for Java

```
public void callEstablishedEvent (int lineIndex)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsCallEstablishedEvent (OcctTelesetDevice*, oint);
```

callHeldEvent

Definition

- The call has been held by TelesetDevice. The call does not have to be established to be held.
- Previous event expected: callDialingEvent or callEstablishedEvent.
- Next event expected: callRetrievedEvent.

Parameters

`lineIndex`: Location of the call.

SDK for Java

```
public void callHeldEvent (int lineIndex)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsCallHeldEvent(OcctTelesetDevice*, oint);
```

callRetrievedEvent

Definition

- The call held has been retrieved by TelesetDevice.
- Previous event expected: callHeldEvent.
- Next event expected: callReleasedEvent.

Parameters

lineIndex: Location of the call.

SDK for Java

```
public void callRetrievedEvent (int lineIndex)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsCallRetrievedEvent (OcctTelesetDevice*, oint);
```

callReleasedEvent

Definition

- Call has been released/disconnected. This is the last event for a call on a particular line.
- Previous event expected: callDialingEvent or callRingingEvent or callEstablishedEvent.
- Next event expected: None.

Parameters

`lineIndex`: Location of the call.

SDK for Java

```
public void callReleasedEvent (int lineIndex)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsCallReleasedEvent (OcctTelesetDevice*, oint);
```

callTransferredEvent

Definition

- Call has been transferred. This event is only sent to the transferrer, who has now dropped out of the call. This event MUST occur before the transferrer receives callReleasedEvent.
- Previous event expected: callEstablishedEvent.
- Next event expected: callReleasedEvent.

SDK for Java

```
public void callTransferredEvent ()
```

SDK for C

```
IMPORT_EXPORT void OCALL tsCallTransferredEvent (OcctTelesetDevice*);
```

callConferencedEvent

Definition

- The call has been conferenced. This event is only sent to the conference controller. If a line index is given, then it means that the call on the line is now active (implied retrieve if it was held). If two calls become active, two callConferencedEvents should be sent, one for each line (the switch does not combine base and consultation call into one call). This event MUST be sent to the conference controller before any callReleasedEvent is sent.
- Previous event expected: callEstablishedEvent.
- Next event expected: callReleasedEvent.

Parameters

`activeLineIndex`: Line index of call to retrieve at the same time.

SDK for Java

```
public void callConferencedEvent (int activeLineIndex)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsCallConferencedEvent (OcctTelesetDevice*, oint);
```

callConferencedEvent

Definition

- The call has been conferenced, and no line is activated. This event is for switches in which the conference controller can complete a call with both calls on hold and still have the resulting call(s) on hold. This event MUST be sent to the conference controller before any releasedEvent is sent.
- Previous event expected: callEstablishedEvent.
- Next event expected: callRetrivedEvent or callReleasedEvent.

SDK for Java

```
public void callConferencedEvent ()
```

SDK for C

```
IMPORT_EXPORT void OCALL tsCallConferencedEvent2 (OcctTelesetDevice*);
```

agentLoginEvent

Definition
Agent logged in.

SDK for Java
```
public void agentLoginEvent ()
```

SDK for C
```
IMPORT_EXPORT void OCALL tsAgentLoginEvent (OcctTelesetDevice*);
```

agentLogoutEvent

Definition
Agent logged out.

SDK for Java
```
public void agentLogoutEvent ()
```

SDK for C
```
IMPORT_EXPORT void OCALL tsAgentLogoutEvent (OcctTelesetDevice*);
```

agentReadyEvent

Definition
The agent is in the "ready to accept calls" state.

SDK for Java
```
public void agentReadyEvent ()
```

SDK for C
```
IMPORT_EXPORT void OCALL tsAgentReadyEvent (OcctTelesetDevice*);
```

agentNotReadyEvent

Definition
The agent is in the "not ready to accept calls" state.

SDK for Java

```
public void agentNotReadyEvent ()
```

SDK for C

```
IMPORT_EXPORT void OCALL tsAgentNotReadyEvent (OcctTelesetDevice*);
```

updateMediaItemIdEvent

Definition

- Update the media item ID for a call. This indicates the call on the specified line is associated with a media item ID. This event can only be sent when beginCallEvent for the specified call is sent with mediaItemIdComplete = false. In addition, this event can only be sent within seven seconds after beginCallEvent is sent.
- Previous event expected: beginCallEvent.

Parameters

- `mediaItemId`: The new ID.
- `lineIndex`: Which call?

SDK for Java

```
public void updateMediaItemIdEvent (int lineIndex,
```

SDK for C

```
IMPORT_EXPORT void OCALL tsUpdateMediaItemIdEvent (OcctTelesetDevice*, oint,
const char*);
```

updateCallDataEvent

Definition

- Update call data for a call. This indicates the call on the specified line is associated with the call data provided. This event can only be sent when beginCallEvent for the call specified is sent with dataComplete = false. This event can be sent anytime after beginCallEvent is sent.
- Previous event expected: beginCallEvent.

Parameters

- `data`: The new data.
- `lineIndex`: Which call?

SDK for Java

```
public void updateCallDataEvent (int lineIndex,
           java.util.Hashtable data)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsUpdateCallDataEvent (OcctTelesetDevice*, oint,
OHashtable*);
```

updateLineIndexEvent

Definition

- The call has changed line indexes. This event can be sent anytime after beginCallEvent.
- Previous event expected: beginCallEvent.

Parameters

- `oldLineIndex`: (0+)
- `newLineIndex`: (0+)

SDK for Java

```
public void updateLineIndexEvent (int oldLineIndex,
           int newLineIndex)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsUpdateLineIndexEvent (OcctTelesetDevice*, oint,
oint);
```

updateSoftphoneDisplayEvent

Definition

- Update the softphone display. This event can be sent anytime after beginCallEvent.
- Previous event expected: beginCallEvent.

Parameters

`softphoneDisplayData`: The data that is displayed on the information screen.

SDK for Java

```
public void updateSoftphoneDisplayEvent (java.util.Hashtable
softphoneDisplayData)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsUpdateSoftphoneDisplayEvent (OcctTelesetDevice*,
OHashtable*);
```

updateOtherPartyNumberEvent

Definition

- Update the other party number. This event can be sent anytime after beginCallEvent.
- Previous event expected: beginCallEvent.

Parameters

`lineIndex`: A call that is changing the other party number.

`otherPartyNumber`: A new other party number.

SDK for Java

```
public void updateOtherPartyNumberEvent (int lineIndex,
            java.lang.String otherPartyNumber)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsUpdateOtherPartyNumberEvent (OcctTelesetDevice*,
oint, const char*);
```

errorEvent

Definition

Generic error.

Parameters

- `lineIndex`: The location of the call, -1 if no line is associated with error

- `errorType`: The command or category of error
- `errorCode`: Specific error
- `errorMessage`: "other error"

See Also

ErrorTypes, ErrorCodes

SDK for Java

```
public void errorEvent (int lineIndex,
            ErrorTypes errorType,
            ErrorCodes errorCode,
            java.lang.String errorMessage)
```

SDK for C

```
IMPORT_EXPORT void OCALL tsErrorEvent (OcctTelesetDevice*, oint, oint, oint,
const char*);
```

5.8 Implementing RoutePointDevice

The following interfaces apply to RoutePointDevice.

- Section 5.8.1, "RoutePointDevice Interface"
- Section 5.8.2, "RoutePointEventListener Interface"

5.8.1 RoutePointDevice Interface

Oracle Telephony Manager uses the RoutePointDevice interface to access a route point resource of the switch or CTI middleware. Third-party consultants or switch or CTI middleware vendor provide the implementation of this interface.

assignMediaItemId

Description

Assign a media item ID to a call ID. The adapter must then pass the media item ID (through the TelesetEventListener interface) to the destination when the call is routed. If the call has already been routed, the adapter must send an updateMediaItemIdEvent (through the TelesetEventListener interface) to the destination. This method is only called in passive mode.

Precondition

callQueuedEvent specified call ID must be received before this call.

Outcome Success

Associate media item ID to call ID. The adapter maintains an internal mapping.

Outcome Failure

- Sent errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_ROUTE_POINT_ASSIGN_MEDIA_ITEM_ID.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED.

Parameters

- `callId`: The callId to be associated with the mediaItemId.
- `mediaItemId`: New call appearances resulting from this command should return the given media item id.

Throws

TeleDeviceException -

SDK for Java

```
public void assignMediaItemId (java.lang.String callId,java.lang.String
mediaItemId)
        throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes rpAssignMediaItemId (OcctRoutePointDevice*, const char*, const
char*);
```

routeCall

Definition

Route Call to Given Destination.

Precondition

routeRequestEvent for the specified call Id must be received before this call.

Outcome Success

Call routed to destination and send callDivertedEvent.

Outcome Failure

- Sent errorEvent or throw TeleDeviceException.
- ErrorType: ErrorTypes.ERROR_TYPE_ROUTE_POINT_ROUTE_CALL.
- Possible Error Code: ErrorCodes.FAILED, FUNCTION_NOT_SUPPORTED.

Parameters

- `callId`: The route request to divert.
- `destinationNumber`: The number to which calls should be routed.
- `mediaItemId`: New call appearances resulting from this command should return the given media item id.

Throws

TeleDeviceException -

SDK for Java

```
public void routeCall (java.lang.String callId,
            java.lang.String destinationNumber,
            java.lang.String mediaItemId)
        throws TeleDeviceException
```

SDK for C

```
OEXPORT OcctCodes rpRouteCall (OcctRoutePointDevice*, const char*, const char*,
const char*);
```

addRoutePointEventListener

Definition

Add a RoutePointEventListener to the listener list.

Parameters

`eventListener`: RoutePointEventListener interested in the events of this RoutePointDevice.

SDK for Java

```
public void addRoutePointEventListener (RoutePointEventListener eventListener)
```

SDK for C

Not applicable.

removeRoutePointEventListener

Definition

Remove the specified listener from the listener list.

Parameters

`eventListener`: The RoutePointEventListener to remove.

SDK for Java

```
public void removeRoutePointEventListener (RoutePointEventListener
eventListener)
```

SDK for C

Not applicable.

5.8.2 RoutePointEventListener Interface

Third-party consultants or switch and CTI middleware providers use the RoutePointEventListener interface to notify Oracle Interaction Center of route point events. Oracle Telephony Manager gives the object that implements this interface to the RoutePointDevice object for event reporting.

beginCallEvent

Definition

- This must be the first event for a new call for a particular RoutePointDevice.
- Next event expected: callQueuedEvent, routeRequestedEvent (active mode).

Parameters

- `mediaItemId`: The media item associated with call.

- `mediaItemIdComplete`: True if all media item IDs are available at this call, false if the adapter might call updateMediaItemId.
- `callId`: Adapter-specific call ID.
- `ani`:
- `dnis`:
- `data`: IVR and other data associated with the call.
- `dataComplete`: True if all call data is available at this call. False if the adapter might call updateCallData.

SDK for Java

```
public void beginCallEvent (java.lang.String mediaItemId,
            boolean mediaItemIdComplete,
            java.lang.String callId,
            java.lang.String ani,
            java.lang.String dnis,
            java.util.Hashtable data,
            boolean dataComplete)
```

SDK for C

```
IMPORT_EXPORT void OCALL rpBeginCallEvent (OcctRoutePointDevice*, const char*,
oboolean, const char*, const char*, const char*, OHashtable*, oboolean);
```

callQueuedEvent

Definition

- The call has been queued at the route point.
- Previous event expected: beginCallEvent.
- Next event expected: routeRequestedEvent (active mode) or callDivertedEvent.

Parameters

`callId`: Adapter-specific call ID.

SDK for Java

```
public void callQueuedEvent (java.lang.String callId)
```

SDK for C

```
IMPORT_EXPORT void OCALL rpCallQueuedEvent (OcctRoutePointDevice*, const char*);
```

callAbandonedEvent

Definition

- The call was released or disconnected before it was connected to an agent.
- Previous event expected: callQueuedEvent.

Parameters

`callId`: Adapter-specific call ID.

SDK for Java

```
public void callAbandonedEvent (java.lang.String callId)
```

SDK for C

```
IMPORT_EXPORT void OCALL rpCallAbandonedEvent (OcctRoutePointDevice*, const
char*);
```

callDivertedEvent

Definition

- The call has been diverted (a blind transfer without answering the call).
- Previous event expected: callQueuedEvent.

Parameters

`callId`: Adapter-specific call ID.

SDK for Java

```
public void callDivertedEvent (java.lang.String callId).
```

SDK for C

```
IMPORT_EXPORT void OCALL rpCallDivertedEvent (OcctRoutePointDevice*, const
char*);
```

routeRequestedEvent

Definition

- The route point has submitted a call for routing.
- Previous event expected: beginCallEvent.

Parameters

`callId`: Adapter-specific call ID.

SDK for Java

```
public void routeRequestedEvent (java.lang.String callId).
```

SDK for C

```
IMPORT_EXPORT void OCALL rpRouteRequestedEvent (OcctRoutePointDevice*, const
char*);
```

updateMediaItemIdEvent

Definition

- Update media item ID for a call. This indicates the call ID specified is associated with a media item ID. This event can only be sent when beginCallEvent for the corresponding call is sent with mediaItemIdComplete = false. In addition, this event can only be sent within **five seconds** after beginCallEvent is sent.
- Previous event expected: beginCallEvent (mediaItemIdComplete = false).

Parameters

- `mediaItemId`: The new ID.
- `callId`: The specific call.

SDK for Java

```
public void updateMediaItemIdEvent (java.lang.String callId,java.lang.String
mediaItemId)
```

SDK for C

```
IMPORT_EXPORT void OCALL rpUpdateMediaItemIdEvent (OcctRoutePointDevice*, const
char*, const char*);
```

updateCallDataEvent

Definition

- Update call data for a call. This indicates the call ID that is specified is associated with the call data that is provided. This event can only be sent when beginCallEvent is sent with dataComplete = false. It can be sent anytime after beginCallEvent.
- Previous event expected: beginCallEvent (dataComplete = false).

Parameters

- `data`: The new data.
- `callId`: Which call?

SDK for Java

```
public void updateCallDataEvent (java.lang.String callId,
           java.util.Hashtable data)
```

SDK for C

```
IMPORT_EXPORT void OCALL rpUpdateCallDataEvent (OcctRoutePointDevice*, const
char*, OHashtable*);
```

errorEvent

Definition

Generic error.

Parameters

- `callId`: Which call? null if no call is associated with error.
- `errorType`: Command or category of error.
- `errorCode`: Specific error.
- `errorMessage`: "Other error."

SDK for Java

```
public void errorEvent (java.lang.String callId,
           ErrorTypes errorType,
           ErrorCodes errorCode,
           java.lang.String errorMessage)
```

SDK for C

```
IMPORT_EXPORT void OCALL rpErrorEvent (OcctRoutePointDevice*, const char*, oint,
oint, const char*);
```

See Also

Oracle Telephony Adapter SDK Javadoc

5.9 Oracle Telephony Adapter Server Messaging Service

Oracle Telephony Adapter Server Messaging Service is a simple utility to send point-to-point messages to any TeleDevice that is identified by the extension. Developers can send a message to a TeleDevice, regardless of whether the receiving TeleDevice is local or remote.

For scalability and fault tolerance purposes, more than one Oracle Telephony Adapter Server can be run in the same server group. Interaction Center servers connect to all available Oracle Telephony Adapter Server in the server group by using a round robin algorithm that is based on Oracle Telephony Adapter Server loads.

When developing the telephony adapter, each TeleDevice (both TelesetDevice and RoutePointDevice) must remain independent of each other.

Caution:

- Do not make cross references between TelesetDevice and RoutePointDevice. Proper operation and event generation of a TeleDevice should not rely on accessing other TeleDevices.
- Do not make a global table of TeleDevices in any implementation, as the locality of a particular TeleDevice cannot be assumed, and the targeted TeleDevice necessary to access could be located in a separate Oracle Telephony Adapter Server process.

Some call scenarios for certain switch and CTI middleware combinations require that some event or state information is communicated across TeleDevices. Use the Oracle Telephony Adapter Server Messaging Service to address this issue.

5.9.1 Oracle Telephony Adapter Server Messaging Service API

A message is defined as a String in Java and char* in C. Oracle Telephony Adapter Server Messaging Service is capable of delivering the message to the specified

destination. Adapter developers can decide how they want to format and parse the string message.

5.9.1.1 Sending Messages

Use the following method and functions to send messages in Java and C.

Java

To send a message, use the Messenger utility class method:

```
public static boolean sendMessage (String destnExtnNumber, String message)
```

C

To send a message, use the occtSendMessage function:

```
oint occtSendMessage (char* destnExtnNumber, char* message);
```

- `destExtnNumber` is the extension number of the target TeleDevice. For TelesetDevice, it can be any of the 3 line extensions. For RoutePointDevice, it is the extension number.
- `message` is the message to send.

5.9.1.2 Receiving Messages

Use the following method and functions to receive messages in Java and C.

Java

To receive messages, implement the following method:

```
public void receiveMessage (String destnExtnNumber, String message);
```

for TelesetDevice and RoutePointDevice.

C

To receive messages, implement the following functions:

```
void tsReceiveMessage (const char* destnExtnNumber, const char* message);
void rpReceiveMessage (const char* destnExtnNumber, const char* message);
```

5.9.1.3 Messaging Service Sample Scenarios

The following sample scenarios describe using Oracle Telephony Adapter Server Messaging Service in faking TeleDevice events and propagating media items IDs.

Faking TeleDeviceEvents

Sometimes it is impossible or very difficult to implement logic in the adapter to generate a required TeleDeviceEvent for a particular TeleDevice based solely on the CTI middleware events received by that specific TeleDevice. For example, when a call is routed by the ACD from the route point to an agent teleset, if CT Connect does not send any NULL/Diverted event to the RoutePointDevice, then it is impossible for the adapter to generate a callDivertedEvent, unless the adapter logic leverages the RECEIVE/INBOUND_CALL received by the target agent's TelesetDevice to fake the callDivertedEvent. Oracle Telephony Adapter Server Messaging Service can be used for cross-referencing between TeleDevices to fake events.

The following scenario describes faking a TeleDevice event.

1. Customer X calls route point RP. RP's RoutePointDevice gets a QUEUED/INBOUND_CALL event from CT Connect => RP's RoutePointDevice sends a beginCallEvent.
2. ACD routes the call from RP to agent A. RP's RoutePointDevice does not receive any NULL/DIVERTED event from CT Connect, but A's TelesetDevice receives a RECEIVE/INBOUND_CALL event (DNIS=RP) => A's TelesetDevice calls sendMessage(RP, "EVT:callDiverted;CALLID:12345") to send a message to RP's RoutePointDevice by way of Oracle Telephony Adapter Server Messaging Service.
3. When RP's RoutePointDevice receives the Oracle Telephony Adapter Server message, RP's RoutePointDevice fakes a callDivertedEvent so that the inbound call queue count is decremented properly.

Propagating Media Item IDs

Oracle Advanced Inbound requires that the same media item ID is associated and propagated to all physical calls that are related to the same initial call, even though the call ID assigned by the switch and CTI middleware may change when the call is transferred or conferenced. One way to meet this requirement is to leverage the CTI middleware's ability to associate data to a call. For example, the adapter may store the media item ID in CT Connect's application data field or in Cisco ICM's call variables. Oracle Telephony Adapter Server Messaging Service offers another alternative: the adapter may pass the media item ID to the destination TeleDevice by way of an Oracle Telephony Adapter Server message.

The following scenario describes propagating media item IDs.

1. Agent A is already on a call with customer X on line 1.

2. Agent A initiates a consultation call to Agent B.
3. Oracle Telephony Manager invokes A's TelesetDevice.consultationCall(Media Item Id=555, line=1, destination=B, consult type=transfer).
4. A's TelesetDevice invokes sendMessage(B,"MIID:555;ANI:A") to let B's TelesetDevice know ahead of time that the next call arriving at B with ANI=A should be associated with media item ID 555.
5. A's TelesetDevice invokes consultCall(callid, destination) on the switch or middleware.
6. B's TelesetDevice receives message and stores the information.
7. B's TelesetDevice receives RECEIVE/INBOUND_CALL event (ANI=A) from CT Connect. B's TelesetDevice generates a beginCallEvent(MIID=555).

5.10 Java Additional APIs

This section describes additional Java classes available in Oracle Telephony Adapter SDK for Java that adapter implementers can use, and contains the following topics:

- Section 5.10.1, "Class Logger"
- Section 5.10.2, "Class ErrorCodes"
- Section 5.10.3, "Class ConsultCallTypes"
- Section 5.10.4, "Class TeleDeviceEventMulticaster"

5.10.1 Class Logger

Class logger is the log utility that uses the Oracle Interaction Center standard logging infrastructure. The current log level is automatically initialized at boot time by Oracle Telephony Adapter Server.

The Logger API consists of two static methods and should be invoked as follows:

```
if (Logger.logStatus (Logger.LOG_LEVEL_VERBOSE))
{
    // Do any processing required to assemble message.
    String msg = "Message.";
    Logger.log (Logger.LOG_LEVEL_VERBOSE,msg);
}
```

5.10.1.1 Static Constants

Five levels of logging severity are available: FATAL, ERROR, WARNING, INFO and VERBOSE. Follow the guidelines in the table below for log messages.

Table 5–1 Log Levels

Log Level	Description
LOG_LEVEL_FATAL	Fatal Log Level, for any fatal, unrecoverable errors
LOG_LEVEL_ERROR	Error Log Level, for any severe errors (default)
LOG_LEVEL_WARNING	Warning Log Level, for any recoverable errors
LOG_LEVEL_INFO	Informational Log Level, for informational messages only
LOG_LEVEL_VERBOSE	Verbose Log level, for debug messages only

public static boolean logStatus(int loglevel)

Returns true if the current log level set by Oracle Telephony Adapter Server is higher than the loglevel argument.

Parameter

`loglevel`: The target log level of the log message to be written

public static void log(int loglevel, String message)

Write the given message to the log file. The adapter should log a message only if logStatus method returns true.

Parameters

- `loglevel`: The target log level of the log message to be written
- `message`: The message to be written

5.10.2 Class ErrorCodes

ErrorCodes objects represent a unique error condition that is used in conjunction with TeleDeviceException and errorEvent method.

ErrorCodes holds all static constants of ErrorCodes object representing differrent ErrorCode values.

Static Constants

Table 5–2 Class ErrorCodes Static Fields

Static Field	Description
MISC_ERROR	Miscellaneous error. Provide explanation in error message.
MISC_INVALID_STATE	Switch object is in an invalid state for operation.
USER_ALREADY_ LOGGED_IN	Login failed because the user is already logged in at another teleset.

5.10.3 Class ConsultCallTypes

The ConsultCallTypes object represents unique consultation call types that are used by various API methods as argument.

ConsultCallTypes hold all static constants of the ConsultCallTypes object representing different ConsultCallType values.

Static Constants

Table 5–3 Class ConsultCallTypes Static Fields

Static Field	Description
TRANSFER	Consultation call for a transfer
CONFERENCE	Consultation call for a conference

5.10.4 Class TeleDeviceEventMulticaster

Class TeleDeviceEventMulticaster is a utility class for managing a list of TelesetEventListener and RoutePointEventListener. This class should be used by the implementation of TelesetDevice and RoutePointDevice interface, as in the following example.

```
public class TelesetDeviceImpl implements TelesetDevice
{
    TelesetEventListener    m_listener ;
    public void addTelesetEventListener (TelesetEventListener eventListener)
    {
     m_listener = TeleEventMulticaster.add (m_listener,eventListener);
    }
    public void removeTelesetEventListener (TelesetEventListener
    eventListener)
```

```
    {
         m_listener = TeleEventMulticaster.remove (m_
        listener,eventListener);
    }
    public void fireCallRingingEvent(int lineIndex)
    {
        if(m_listener != null)
        {
        m_listener.callRingingEvent(lineIndex);
        }
    }
    .... implemation of other methods ....
}
```

public static RoutePointEventListener add(RoutePointEventListener listener, RoutePointEventListener toBeAdded)

Definition

Add a new RoutePointEventListener.

Parameters

- `listenerList`: Original listener (list)
- `toBeAdded`: Listener to be added

public static RoutePointEventListener remove(RoutePointEventListener listener, RoutePointEventListener toBeRemoved)

Definition

Remove a RoutePointEventListener.

Parameters

- `listenerList`: Original listener (list)
- `toBeRemoved`: Listener to be removed

public static TelesetEventListener add(TelesetEventListener listener, TelesetEventListener toBeAdded)

Definition

Add a new TelesetEventListener.

Parameters

- `listenerList`: Original listener (list)
- `toBeAdded`: Listener to be added

public static TelesetEventListener remove(TelesetEventListener listener, TelesetEventListener toBeRemoved)

Definition

Remove a TelesetEventListener.

Parameters

- `listenerList`: Original listener (list)
- `toBeRemoved`: Listener to be removed

5.11 Windows NT and Windows 2000 Additional APIs

This section describes additional C functions available in SDK for the Windows NT and Windows 2000 operating systems. These functions are defined in the header file occt_pub.h and Hashtable.h. This section contains the following topics:

5.11.1 Constants

Constants are defined in occt_pub.h for log levels, consultation types and error codes.

Table 5–4 Log Levels and Corresponding ConsultCall Types

Log Levels	ConsultCall Types
VERBOSE_LEVEL	CODE_CONFERENCE
INFO_LEVEL	CODE_TRANSFER

Table 5–4 Log Levels and Corresponding ConsultCall Types (Cont.)

Log Levels	ConsultCall Types
WARNING_LEVEL	
ERROR_LEVEL	
FATAL_LEVEL	

Table 5–5 Error Types and Error Codes

Error Types	Error Codes
TYPE_UNKNOWN	CODE_SUCCESS
TYPE_TELESET_CREATE_TELESET_DEVICE	CODE_FAILURE
TYPE_TELESET_ANSWER_CALL	CODE_FUNCTION_NOT_SUPPPORTED
TYPE_TELESET_RELEASE_CALL	CODE_LINE_INDEX_OUT_OF_RANGE
TYPE_TELESET_HOLD_CALL	CODE_NO_CURRENT_CALL
TYPE_TELESET_RETRIEVE_CALL	CODE_DESTINATION_BUSY
TYPE_TELESET_CONSULTATION_CALL	CODE_DESTINATION_INVALID
TYPE_TELESET_COMPLETE_TRANSFER	CODE_AGENT_INVALID
TYPE_TELESET_COMPLETE_CONFERENCE	CODE_AGENT_PASSWORD_INVALID
TYPE_TELESET_BLIND_TRANSFER	CODE_TELESET_IN_USE
TYPE_TELESET_SWAP_WITH_HELD	CODE_TELESET_INVALID
TYPE_TELESET_SEND_DTMF	CODE_SWITCH_CONNECTION_LOST
TYPE_TELESET_LOGIN_AGENT	CODE_SWITCH_RECONNECT_SUCCESS
TYPE_TELESET_LOGOUT_AGENT	
TYPE_TELESET_AGENT_READY	
TYPE_TELESET_AGENT_NOT_READY	
TYPE_ROUTE_POINT_CREATE_ROUTE_ POINT_DEVICE	
TYPE_ROUTE_POINT_ASSIGN_MEDIA_ ITEM_ID	
TYPE_ROUTE_POINT_ROUTE_CALL	
TYPE_NETWORK	

5.11.2 occtLogPrintf Function

Function Prototype

```
void occtLogPrintf (LogLevelCodes level,const char* tmpl,...);
```

Definition

The occtLogPrintf function is similar to the Logger class in SDK for Java. This function accesses the logging service of Oracle Telephony Adapter Server runtime and writes log messages to the standard Oracle Telephony Adapter Server log file. This function works like a normal standard C printf function in that it writes formatted operands to the Oracle Telephony Adapter Server log file. The argument operands are formatted under control of the tmpl operand.

Parameters

- `level`: Log level of this message
- `tmpl`: C printf style template, for example, "string = %s, int = %d\n"
- `...`: A variable list of arguments corresponding to the template tokens

5.11.3 OHashtable Functions

Oracle Telephony Adapter SDK for C provides a simple implementation of Hashtable. Developers can use these functions to create and manipulate data in hashtables.

5.11.3.1 occtHashtableNew

Function Prototype

```
OHashtable* occtHashtableNew ();
```

Definition

Create a new OHashtable

5.11.3.2 occtHashtableGet

Function Prototype

```
void* occtHashtableGet (OHashtable* ht, char* key);
```

Definition
Get the value of a key.

Parameters
- `ht`: The pointer to OHashtable
- `key`: The key

5.11.3.3 occtHashtablePut

Function Prototype
```
void* occtHashtablePut (OHashtable* ht, char* key, void* value);
```

Definition
Put a key-value pair into the hashtable.

Parameters
- `ht`: The pointer to OHashtable
- `key`: The key
- `value`: The value

5.11.3.4 occtHashtableDestroy

Function Prototype
```
void occtHashtableDestroy (OHashtable* ht);
```

Definition
Destroy an OHashtable and free all resources that are allocated to this structure.

Parameter
`ht`: The pointer to OHashtable

5.11.3.5 occtHashtableSize

Function Prototype
```
int occtHashtableSize (OHashtable* ht);
```

Definition

Get the size of the hashtable.

Parameters

- `ht`: The pointer to OHashtable
- `key`: The key

5.11.3.6 occtHashtableGetKeys

Function Prototype

```
char** occtHashtableGetKeys (OHashtable* ht, int *len);
```

Definition

Get all the keys from the hashtable.

Parameters

- `ht`: Pointer to OHashtable
- `len`: Pointer to an int variable , the total number of keys will be set to this variable returns an array of keys (two-dimensional char array) occtHashtableRemoveKey

Function Prototype

```
void* occtHashtableRemoveKey (OHashtable* ht, char* key);
```

Definition

Remove a key-value pair from hashtable.

Parameters

- `ht`: The pointer to OHashtable
- `key`: The key

5.11.4 occtTelesetDevice Functions

Each OcctTelesetDevice pointer is internally associated with an OHashtable. Developers can access the data stored in the Hashtable by using the following functions.

5.11.4.1 occtTelesetDeviceGet

Function Prototype

```
void* occtTelesetDeviceGet (OcctTelesetDevice* device, char* key);
```

Definition

Get the value of a key.

Parameters

- `device`: Pointer to the OcctTelesetDevice structure
- `key`: The key to retrieve the value

5.11.4.2 occtTelesetDevicePut

Function Prototype

```
void* occtTelesetDevicePut (OcctTelesetDevice* device, char* key,void* value);
```

Definition

Put a key-value pair into the hashtable.

Parameters

- `device`: Pointer to the occtTelesetDevice structure
- `key`: The key
- `value`: The value

5.11.4.3 occtTelesetDeviceRemoveKey

Function Prototype

```
void* occtTelesetDeviceRemoveKey (OcctTelesetDevice* device, char* key);
```

Definition

Remove a key-value pair from the hashtable.

Parameters

- `device`: Pointer to the occtTelesetDevice structure
- `key`: The key

5.11.4.4 occtTelesetDeviceGetKeys

Function Prototype

```
char** occtTelesetDeviceGetKeys (OcctTelesetDevice* device, int *len);
```

Definition

Get all keys from the hashtable.

Parameters

- `device`: Pointer to the OcctTelesetDevice structure
- `len`: Pointer to an int variable, the total number of keys set to this variable returns an array of keys (two- dimensional char array)

5.11.5 occtRoutePointDevice Functions

Each OcctRoutePointDevice pointer is internally associated with an OHashtable. Developers can access the data stored in the Hashtable by using the following functions.

5.11.5.1 occtRoutePointDeviceGet

Function Prototype

```
void* occtRoutePointDeviceGet (OcctRoutePointDevice* device, char* key);
```

Definition

Get the value of a key.

Parameters

- `device`: Pointer to the OcctRoutePointDevice structure
- `key`: The key to retrieve the value

5.11.5.2 occtRoutePointDevicePut

Function Prototype

```
void* occtRoutePointDevicePut (OcctRoutePointDevice* device, char* key,void*
value);
```

Definition

Put a key-value pair into the hashtable.

Parameters

- `device`: Pointer to the OcctRoutePointDevice structure
- `key`: The key
- `value`: The value

5.11.5.3 occtRoutePointDeviceRemoveKey

Function Prototype

```
void* occtRoutePointDeviceRemoveKey (OcctRoutePointDevice* device, char* key);
```

Definition

Remove a key-value pair from the hashtable.

Parameters

- `device`: Pointer to the OcctRoutePointDevice structure
- `key`: The key

5.11.5.4 occtRoutePointDeviceGetKeys

Function Prototype

```
char** occtRoutePointDeviceGetKeys (OcctRoutePointDevice* device, int *len);
```

Definition

Get all keys from the hashtable.

Parameters

- `device`: Pointer to the OcctRoutePointDevice structure.
- `len`: Pointer to an int variable, the total number of keys set to this variable returns an array of keys (a two-dimensional char array).

6

Implementing Multi-Site Functionality for the Telephony Adapter

This section explains the procedures and guidelines to use when an SDK computer telephony integration project requires multi-site functionality. With multi-site functionality, a call spans multiple interaction centers, and the destination agent who receives the call has access to the same call data that is included in the call at the source.

This chapter includes the following multi-site topics:

6.1 Multi-Site Environment Call Flows

Multi-site call scenarios are of two categories: enterprise routing and enterprise call and data transfer. Enterprise routing involves routing a call from a route point at one interaction center to a destination, either a route point or an agent, at another interaction center. Enterprise call and data transfer involves one agent in an interaction center consulting, and subsequently transferring or conferencing a call, to an agent at another interaction center. The call spans two interaction centers, and therefore two different Oracle Telephony Adapter Servers. The Oracle Telephony Adapter Servers do not require the same implementation, because different interaction centers in an enterprise may have different telephony switches.

Any Oracle Telephony Adapter Server that implements the Oracle Telephony Adapter SDK can send and receive multi-site calls from any other Oracle Telephony Adapter Server. Multi-site communication does not require coordinating protocols with other Oracle Telephony Adapter Server developers. Multi-site support does

not depend on underlying CTI middleware or switch support other than the basic support that is already present.

When a call is actively moved by routeCall, makeCall, consultationCall or blindTransfer to another target, a multi-site message is first sent to the target. This message contains the media item ID for the call and several possible ANIs that the target can expect to receive. When the call arrives at the target, the target can match the incoming call ANI to one of the expected ANIs, and then reattach the media item ID to the call.

6.1.1 Enterprise Routing Flow

The following call flow scenario applies to enterprise routing.

1. A call arrives at a route point, which is typically a monitored ACD or control queue. The Oracle Inbound Telephony Server creates a media item and its accompanying media item ID. The media item contains IVR data and other call data such as ANI or DNIS.
2. An eligible agent at another interaction center requests the call.
3. The Oracle Telephony Manager of the remote interaction center knows the canonical phone number of the target agent or teleset and the originating middleware configuration. Oracle Telephony Manager determines the proper dial string (by using Oracle Advanced Inbound's Local Call Data functionality) to send to the originating Inbound Telephony Server.
4. Before routing the call, the Oracle Telephony Adapter Server of the originating interaction center uses the Oracle Telephony Adapter Server Messaging Service to send a message to the destination. The message contains the media item ID and the call ANI that the destination can expect to receive.
5. Oracle Telephony Adapter Server Messaging Service uses the destination dial string to determine the destination middleware, and then sends the message. The destination Oracle Telephony Adapter Server Messaging Service determines the exact teleset and delivers the message. The destination saves the message.
6. The originating Oracle Telephony Adapter Server distributes the call by using the Oracle Telephony Adapter Server Messaging Service.
7. The destination, using the Oracle Telephony Adapter Server Messaging Service, receives the expected call (by matching the incoming call ANI to the expected ANI in the previously received message) and reattaches the media item ID to the call.

8. The destination agent receives a screen pop that includes the call data that was gathered at the initial route point, including IVR data and the initial route point that received the call.

6.1.2 Enterprise Call and Data Transfer Flow

The following call flow scenario applies to enterprise call and data transfer.

1. An agent has an active call and places a consultation call to another agent. A media item already exists for the original (customer) call.
2. Before placing the consultation call, the Oracle Telephony Adapter Server of the originating interaction center sends a message to the destination by way of the Oracle Telephony Adapter Server Messaging Service. This message contains the media item ID and several call ANIs that the destination can expect to receive.
3. Oracle Telephony Adapter Server Messaging Service uses the destination dial string to determine the destination middleware, and then sends the message. The destination Oracle Telephony Adapter Server Messaging Service determines the exact teleset and delivers the message, which the destination saves. If the target is not logged in or is not an interaction center agent, then the dial string would not match any teleset.
4. The originating Oracle Telephony Adapter Server places the consultation call.
5. The destination Oracle Telephony Adapter Server receives the expected call, matches the incoming call ANI to one of the expected ANIs in the previously received message, and reattaches the media item ID to the call.
6. The destination agent (a transfer or conference recipient) receives the same screen pop that the originating agent received, but which includes information that was updated by the originating agent.

6.2 Implementing Multi-Site Functionality

The procedures for implementing multi-site functionality for a telephony adapter include the following topics:

- Section 6.2.1, "Implementing a Teleset Device for a Multi-Site"
- Section 6.2.2, "Implementing Route Points for a Multi-Site"
- Section 6.2.3, "Developing Message Class for a Multi-Site"
- Section 6.2.4, "Developing MessengerUtility Class for a Multi-Site"

- Section 6.2.5, "Telephony Adapter Multi-Site Implementation Templates"

6.2.1 Implementing a Teleset Device for a Multi-Site

Use the following procedures to implement a teleset device for a telephony adapter in a multi-site environment.

6.2.1.1 Initialization and Registration

Complete the following procedure before returning from loginAgent().

Log in

Not Applicable

Responsibility

Not Applicable

Prerequisites

Implement Advanced SDK.

Steps

1. Save the internal ANI. This is the number that agents see on their telesets when they receive calls from the originating teleset. Depending upon the PBX/ACD (switch) platform, this number can be one of three things:
   - The line extension, given in createTelesetDevice(ext1, ext2, ext3).
   - The acd agent ID, given during loginAgent(acdAgentId, acdAgentPassword, acdQueue).
   - The teleset hardware number, which administrators can retrieve by using the following syntax:

     ```
     MessengerUtility.getTelesetHardwareNumber(acdAgentId)
     ```

2. Save the external ANI, which is the number that agents at other interaction centers see on their teleset when agents at this interaction center call them. The values are set up in the database. Retrieve the values by calling the following syntax:

   ```
   String[] externalAnis = MessengerUtility.getTelesetAnis(acdAgentId);
   ```

Optionally, use the following syntax to retrieve the external ANIs one at a time for each extension:

```
String externalAni1 = MessengerUtility.getTelesetAni(acdAgentId,
extension);
```

3. Use the following syntax to register internal targets, which are all the internal numbers that other agents in this interaction center can use to call this teleset device. Generally, these numbers are the same as the internal ANIs. Internal targets are switch specific:

```
MessengerUtility.addDeviceTargetNumbers(acdAgentId,
numbersAsStringArray);
MessengerUtility.addDeviceTarget(acdAgentId, number);
```

4. Use the following syntax to instantiate a MessengerUtility object that will keep track of multi-site messages for and match incoming calls to the appropriate multi-site message:

```
MessengerUtility m_msgUtil = new MessengerUtility();
```

6.2.1.2 Keeping Track of Call ANIs

It is essential to keep track of the call ANIs, which are usually the customer telephone numbers received from the switch. For outbound calls, this may be the external ANI, which is typically the telephone number associated with the switch's outbound trunks. To identify the ANI, make an outbound call, blind transfer to a route point, then route the call to an agent and see which ANI appears at the destination teleset.

- On initialization, define a variable to keep track of current call ANIs. Because only one call can be on a line, define a simple array:

```
    MAX_NUMBER_OF_TELESET_LINES = 3;
    private String m_callAnis[] = new String[
MAX_NUMBER_OF_TELESET_LINES];
```

- In each instance of TelesetDevice object, create a String array m_callAnis for storing call ANIs. In each instance of RoutePointDevice object, create a Hashtable m_callAnis for storing call ANIs.
- On makeCall() method:

```
m_callAnis[lineIndex] = m_externalANI[lineIndex];
```

- When receiving a begin call event:

```
m_callAnis[lineIndex] = <other party number from event>;
```

- When receiving a call transferred or conferenced event:

```
m_callAnis[lineIndex] = <third party number/other party number from event>;
```

> **Note:** It is not necessary to keep track of call ANIs for calls that are transferred or conferenced multiple times. For example, in a situation in which agent A consults agent B who consults agent C who in turn consults agent D, the transfer can be completed in any order without agent D knowing the call ANI.

6.2.1.3 Initiating Multi-Site Calls

Before initiating a call (makeCall, consultationCall, blindTransfer), send a multi-site message to the destination. Always send this message, even for internal calls. In each instance of TelesetDevice object, create a String array m_callAnis for storing call ANIs. In each instance of RoutePointDevice object, create a Hashtable m_callAnis for storing call ANIs:

```
m_msgUtil.sendMultisiteMessage
    (dest, mediaItemId, m_callAnis[lineIndex],
    m_internalAnis[lineIndex], m_externalAnis[lineIndex]);
```

6.2.1.4 Receiving Multi-Site Calls

Save received messages with the MessengerUtility. When receiving a call, check to see if a multi-site message is associated with the call, and then extract the media item ID from the message:

- On receiveMessage() method:

```
m_msgUtil.putMessage(message);
```

- Before calling the SDK beginCallEvent() API:

```
Message m = m_msgUtil.removeMessage(callAni);
if (m != null)  mediaItemId = m.get(Message.OCCT_OTAS_KEY_MEDIA_ITEM_
ID);
```

6.2.2 Implementing Route Points for a Multi-Site

Use the following procedures when implementing route points for a telephony adapter in a multi-site environment.

Prerequisites

Implement Advanced SDK.

6.2.2.1 Initialization and Registration

Route points have neither internal nor external ANIs. Because they only have a route point number, that number is automatically registered. Instantiate a MessengerUtility object that will keep track of multi-site messages and match incoming calls to the appropriate multi-site message. (Route points usually require a bigger message queue than telesets because they receive more calls.)

```
MessengerUtility m_msgUtil = new MessengerUtility(20, 60);
```

6.2.2.2 Keeping Track of Call ANIs

It is necessary to keep track of the call ANIs, which are usually the customer telephone numbers received from the switch. This number may be harder to determine for consultations through the route point. Most switches only send the agent internal ANI and possibly the base call (customer) ANI. In these cases, it is easier to save the multi-site message that comes with the call and reuse it, because it includes all the various ANIs. In each instance of TelesetDevice object, create a String array m_callAnis for storing call ANIs. In each instance of RoutePointDevice object, create a Hashtable m_callAnis for storing call ANIs.

- On initialization, define a variable to keep track of current call ANIs. Because most implementations can only differentiate calls by their call ID, it is best to define some sort of hash table:

  ```
  // key = call id, value = call ani
  private Hashtable m_callAnis = new Hashtable();
  ```

- On receiving a call event, use:

  ```
  m_callAnis.put(callId, ani);
  ```

See Also

Section 6.2.2.3, "Receiving Multi-Site Calls"

6.2.2.3 Receiving Multi-Site Calls

Save received messages with MessengerUtility. When receiving a call, check to see if there is a multi-site message that is associated with the call, and then extract the media item ID from the message. At this point, save the message in permanent storage and use it when routing the call. It is not possible to save a message on receiveMessage because it does not have a callId.

- On receiveMessage() method:

```
m_msgUtil.putMessage(message);
```

- Before calling the SDK beginCallEvent() API:

```
Message m = m_msgUtil.removeMessage(callAni);
if (m != null)
{
    mediaItemId = m.get(Message.OCCT_OTAS_KEY_MEDIA_ITEM_ID);
    m_msgUtil.storeMessage(callId, m);
}
```

6.2.2.4 Routing a Call

Before routing a call, check for a multi-site message for the call, and use any message to warn the destination.

On routeCall() method, use the following code:

```
Message m = m_msgUtil.removeStoredMessage(callId);
if (m != null)
{
    m_msgUtil.sendMultisiteMessage(dest, m);
}
else
{
    // route point has no internal/external ANI, set to null
    m_msgUtil.sendMultisiteMessage(dest, callAni, null, null);
}
```

6.2.2.5 Cleaning Up

For every event that signifies a call leaving the route point, use the following code to clean up the stored message table in the MessengerUtility and the call ANI table:

```
fireCallAbandonedEvent(), fireCallDivertedEvent()
m_msgUtil.removeStoredMessage(callId);
```

6.2.3 Developing Message Class for a Multi-Site

The Message class wraps key-value pairs to be sent by Oracle Telephony Adapter Server Messaging Service. Because of the need to support multi-site environments, the Message class contains multi-site-specific methods and keys. Standard keys convey standard information that allows an Oracle Telephony Adapter Server implementation to send a multi-site message that any other Oracle Telephony Adapter Server implementation can understand.

- Package: package oracle.apps.cct.sdk;
- Class: public class Message

6.2.3.1 Keys

The Oracle Telephony Adapter Server (OTAS) implementation adds keys for telesets and route points. The following table lists and describes the keys.

Table 6–1 Key Telesets and Route Points

Key	Teleset	Route Point	Used by	Description
OCCT_OTAS_ KEY_CALL_ANI	ANI of the base call	ANI of the route point call	Target to match incoming routed or blind transferred calls.	The customer or base call phone number, which is null for outbound and internal calls.
OCCT_OTAS_ KEY_MEDIA_ ITEM_ID	Media Item ID of base call	Media Item ID of the route point call	Target to pass to the ITS/OTM layer in a beginCallEvent or updateMediaItemId Event	The Media Item ID assigned to the base/consultation call by way of a makeCall, consultationCall, blindTransfer, routeCall, or assignMediaItemId command.
OCCT_OTAS_ KEY_MESSAGE_ TYPE	OCCT_OTAS_ VALUE_ MESSAGE_ TYPE_ MULTISITE	OCCT_OTAS_ VALUE_ MESSAGE_ TYPE_ MULTISITE	OTAS Messenger Service and target to determine that this is a multi-site message	Value is OCCT_OTAS_VALUE_ MESSAGE_TYPE_MULTISITE, defined as a String constant.
OCCT_OTAS_ KEY_SENDER_ INTERNAL_ANI	Extension of the agent teleset or teleset line	Not Applicable	Target to match incoming internal calls	OTAS uses its own internal ANI. See Section 6.2.5, "Telephony Adapter Multi-Site Implementation Templates".
OCCT_OTAS_ KEY_SENDER_ EXTERNAL_ANI	ANI of the agent teleset or teleset line	Not Applicable	Target to match incoming external calls	OTAS uses its own external (outbound) ANI. See Section 6.2.5, "Telephony Adapter Multi-Site Implementation Templates".

6.2.3.2 Methods

Message class has three methods: basic, data manipulation and multi-site messaging.

Basic Methods

The method createMessage is for non-multi-site messages. The method messageType is implementation specific:

```
static Message createMessage(String messageType)
```

The method createMessageFromEncodedString is for decoding a message received from the Oracle Telephony Adapter Server Messenger Service:

```
static Message createMessageFromEncodedString(String encodedString)
```

Data Manipulation Methods

The following methods are analogous to the Hashtable methods, except that they take and return String values only. Null values are converted to empty string ("").

The following syntax converts the Message object to a String for sending by way of the Messenger service:

```
String get(String key)
Enumeration keys()
void put(String key, String value)
String remove(String key)

String encodeToString()
```

Multi-Site Messaging Methods

In the following syntax, for RoutePoint calls, senderInternalAni and senderExternalAni are null:

```
static Message createMultisiteMessage(String mediaItemId, String callAni,
String senderInternalAni, String senderExternalAni)
```

6.2.4 Developing MessengerUtility Class for a Multi-Site

Public class MessengerUtility encapsulates methods to perform two functions:

- Register devices with the Oracle Telephony Adapter Server Messenger Service
- Help telephony adapter implementers to implement multi-site support

The MessengerUtility class provides methods for four areas of support:

- Methods to register TeleDevices with the Oracle Telephony Adapter Server Messenger Service:
  - addDeviceTargetNumber()
  - addDeviceTargetNumbers()
  - getTelesetHardwareNumber()
- Utility methods to determine the path of a placed call:
  - isExternalCall()
  - isSwitchInternalCall()
- Methods to get TeleDevice information for use in multi-site messaging:
  - getTelesetAni()
  - getTelesetAnis()
- Utility methods to help keep track of multi-site messages, and for sending multi-site messages on behalf of TeleDevices:
  - MessengerUtility()
  - putMessage()
  - removeMessage()
  - storeMessage()
  - removeStoredMessage()
  - sendMultisiteMessage()

The MessengerUtility class package is package oracle.apps.cct.sdk.

6.2.4.1 Adapter Glue Methods

To support general messaging, the adapter TelesetDevice must register its device targets. These are internal numbers that can be dialed to reach the TelesetDevice.

The Adapter calls the method addDeviceTargetNumber to register an internal dialable number for the given teleset:

```
static String addDeviceTargetNumber(String acdAgentId, String targetNumber)
```

The Adapter calls the method addDeviceTargetNumbers to register internal dialable numbers for the given agent:

```
static String addDeviceTargetNumbers(String acdAgentId, String[]
```

```
targetNumbers)
```

Use the following method if the adapter needs the hardware number for a teleset, which should only be necessary if the hardware number is a dialable internal number:

```
static String getTelesetHardwareNumber(String acdAgentId)
```

The following method is for the teleset external ANI, which must be included in multi-site messages:

```
static String[] getTelesetAnis(String acdAgentId)
```

The following method is for the teleset external ANI, which must be included in multi-site messages. This method will be for implementations that use the Oracle Telephony Adapter Server Common Layer, which will be available in a future release:

```
static String getTelesetAni(String acdAgentId, String extension)
```

6.2.4.2 Adapter Helper Methods

The MessengerUtility can also send multi-site messages and keep track of pending incoming multi-site calls. These methods are not static. Each adapter instance (TeleDevice instance) should instantiate and use its own MessengerUtility object. See Chapter 5, "Developing the Telephony Adapter" for an example.

The maxRecords value equals the maximum number of pending incoming multi-site calls to keep track of. If more calls than the default of three are stored, then the oldest messages are discarded. The Timeout value equals the maximum time in seconds before a multi-site message has aged out:

```
MessengerUtility()
MessengerUtility(int maxRecords, int timeout)
```

When the adapter receives a message (by way of receiveMessage()), use putMessage to store it. If the message is not a multi-site message, then the return value is "false," and the message is discarded:

```
boolean putMessage(Message message)
boolean putMessage(String message)
```

Before firing a beginCallEvent, use removeMessage() to get any multi-site message for this call, then call get(Message.OCCT_OTAS_KEY_MEDIA_ITEM_ID) to get any associated media item ID:

```
Message removeMessage(String ani)
```

The following method sends a multi-site message to the destination, regardless of whether the destination is an internal or external number. Any of the ANIs can be null if they do not exist. This method automatically waits one second after sending a message to an external target, to make sure that the message is received before the call is received:

```
void sendMultisiteMessage(String dest, String mediaItemId, String callAni,
String senderInternalAni, String senderExternalAni)
```

6.2.4.3 Adapter Helper Methods (Route Point)

If route points receive a message prior to the call, then they should reuse the same multi-site message when routing calls. The following methods help cache those multi-site messages. The key should be the call ID for most implementations.

After removeMessage, use storeMessage to keep the message in storage until retrieving it or restarting the server:

```
void  storeMessage(String key, Message message)
```

The method removeStoredMessage removes and returns a message from the long-term cache, and returns null if there is no message. Because this is the only way to clean up the cache, call this method when the call is routed or if the call is abandoned or diverted:

```
Message removeStoredMessage(String key)
```

The method sendMultisiteMessage sends a stored message to the destination:

```
void sendMultisiteMessage(String dest, Message message)
```

See Also

Section 6.2.5, "Telephony Adapter Multi-Site Implementation Templates"

6.2.4.4 Other Methods

The method isSwitchInternalCall returns the value true if the dialString points to a destination on the same switch. If the dialString does not begin with an access code, such as international, long distance, local number and tie line, then it is a switch internal number:

```
static boolean isSwitchInternalCall(String dialString)
```

6.2.5 Telephony Adapter Multi-Site Implementation Templates

Use the following sample code to implement TelesetDevice and RoutePointDevice.

This section includes the following topics:

6.2.5.1 Oracle Telephony Adapter Server Implementation of TelesetDevice

```
// Array for keeping track of current call ani on each teleset line
MAX_NUMBER_OF_TELESET_LINES = 3;
private String m_callAnis[] = new String[MAX_NUMBER_OF_TELESET_LINES];
// ANI sent for internal calls...
private String[] m_internalANI = new String[3];

// ANI sent for external and tie-line calls...
private String[] m_externalANI = null;

// acd agent id, for getting information later...
private String m_acdAgentId = null;

// MessengerUtility to keep track of pending incoming multi-site calls:
MessengerUtility m_msgUtil = new MessengerUtility();

public <constructor>(String extension1, String extension2, String extension3)
{
    // [OPTIONAL] For implementations that use teleset extension as internal
    // ANI:
    m_internalANI[0] = extension1;
    m_internalANI[1] = extension2;
    m_internalANI[2] = extension3;

    // Insert implementer's code.
}

public void loginAgent
    String acdAgentId, String acdAgentPassword, String acdAgentQueue)
{
    m_acdAgentId = acdAgentId;

    // [OPTIONAL] For implementations in which other agents can call a teleset
```

```
    //extension:
    MessengerUtility.addDeviceTargetNumbers(m_acdAgentId, m_internalANI);

    // [OPTIONAL] For implementations that use agent id as internal ANI:
    m_internalANI[0] = acdAgentId;
    m_internalANI[1] = acdAgentId;
    m_internalANI[2] = acdAgentId;

    // [OPTIONAL] For implementations in which other agents can call an acd
    // agent id:
    MessengerUtility.addDeviceTargetNumber(m_acdAgentId, acdAgentId);

    // [OPTIONAL] For implementations that use teleset hardware number as
    // internal ANI:
    String hardwareNumber = MessengerUtility.getTelesetHardwareNumber(m_
    acdAgentId);
    m_internalANI[0] = hardwareNumber;
    m_internalANI[1] = hardwareNumber;
    m_internalANI[2] = hardwareNumber;

    // [OPTIONAL] For implementations in which other agents can call the
    // hardware number:
    MessengerUtility.addDeviceTargetNumber(m_acdAgentId, hardwareNumber);

    // Set external ANIs
    // Define ANIs in the HTML Admin
      externalANI = MessengerUtility.getTelesetAnis(m_acdAgentId);

    // Insert implementer's code to login agent...
}

public void makeCall(String mediaItemId, int lineIndex, String
destinationNumber)
throws TeleDeviceException
{
    // Insert adaptor implementation code to check state or set state before
    // placing a call on the switch...

    // Create and send a message.
    //
    // MessengerUtility automatically waits 1 second after sending a message
    // to an external number.
    m_msgUtil.sendMultisiteMessage
        (destinationNumber, mediaItemId, null,
        m_internalANI[lineIndex], m_externalANI[lineIndex]);
```

```
// Insert implementer's code to place the call...
}

public void consultationCall(String mediaItemId, int lineIndex,
    String destinationNumber, ConsultCallType consultationType)
    throws TeleDeviceException
{
// Insert adaptor implementation code to check state or set state before placing
// a call on the switch...

// Create and send a message.
//
// MessengerUtility automatically waits 1 second after sending a message
// to an external number.
//
// Retrieve the previously saved call ani
String callAni = m_callAnis[lineIndex];
m_msgUtil.sendMultisiteMessage
(destinationNumber, mediaItemId, callAni,
m_internalANI[lineIndex], m_externalANI[lineIndex]);

// Insert implementer's code to place the call...
}

public void blindTransfer(String mediaItemId, int lineIndex,
String destinationNumber)
throws TeleDeviceException
{
    // Insert adaptor implementation code to check state or set state before
    // placing a call on the switch...

    // Create and send a message.
    //
    // MessengerUtility automatically waits 1 second after sending a message
    // to an external number.
    //
    // Retrieve the previously saved call ani
    String callAni = m_callAnis[lineIndex];
    m_msgUtil.sendMultisiteMessage
    (destinationNumber, mediaItemId, callAni,
    m_internalANI[lineIndex], m_externalANI[lineIndex]);

    // Insert implementer's code to place the call...
}
```

```
public void receiveMessage(String destnExtnNumber, String message)
{
    m_msgUtil.putMessage(message);

    // To get information from the message...
    Message m = Message.createFromEncodedString(message);

    // Insert implementer's code... [for example, String data = m.get(KEY_FOR_
    DATA)]
}

public void fireBeginCallEvent(...)
{
    // Optionally, implementers can skip this code if they already know the
    // media item id from the event or another (implementation-specific) source.
    // mi will either be "" or it will have the media item id for the call.
    String mi = "";
    String callAni = <get call ani from receive call event>
    // Save call ani for use in consultationCall() and blindTransfer() methods.
        m_callAnis[lineIndex] = callAni;
    Message mesg = m_msgUtil.removeMessage(callAni);
    if (mesg != null)  mi = mesg.get(Message.OCCT_OTAS_KEY_MEDIA_ITEM_ID);

    // Insert implementer's code.
}
```

6.2.5.2 Oracle Telephony Adapter Server Implementation of RoutePointDevice

```
// Route points do not have their own ANI. They also have only one
// dialable extension, so implementers do not have to call addDeviceTargetNumber
// because it is called automatically.

// Implementers need to reuse any multi-site message that is associated with
// the call, including internal calls that are routed externally (at that
// point, callId is an internal number and the route point does not know the
// external ANI).

// Use MessengerUtility to keep track of pending incoming multi-site calls.
// Implementers need to specify a higher maxRecords value because a route point
// will receive more simultaneous calls than an agent receives.
MessengerUtility m_msgUtil = new MessengerUtility(20,60);

// Hashtable to keep track of call id to call ani mappings.
// key = call id, value = call ani
```

```
private Hashtable m_callAnis = new Hashtable();
// Add the following code snippet to the routeCall method:
public void routeCall(String callId, String destinationNumber, String
mediaItemId)
    throws TeleDeviceException
{
    // Insert adaptor implementation code to check state or set state before
    // placing a call on the switch...

    // Create and send message.
    //
    // MessengerUtility automatically waits 1 second after sending a message
    // to an external number.
    String callId = <get the call id of the call being routed>
    // Retrieve the previously saved call ani
    String callAni = m_callAnis.get(callId);
    Message m = msgUtil.removeStoredMessage(callId);
    if (m == null)
    {
    m_msgUtil.sendMultisiteMessage
    (destinationNumber, mediaItemId, callAni, null, null);
    }
    else
    {
    msgUtil.sendMultisiteMessage(destinationNumber, m);
    }

    // Insert implementer's code to route the call...
}

public void receiveMessage(String destnExtnNumber, String message)
{
    m_msgUtil.putMessage(message);

    // To get information from the message...
    Message m = Message.createFromEncodedString(message);

    // Insert implementer's code... [for example, String data = m.get(KEY_FOR_
    DATA)]
}

public void fireBeginCallEvent(...)
{
    // mi will either be "" or it will have the media item id for the call.
    String mi = "";
```

```
    String callAni = <get call ani from receive call event>
    String callId = <get call id that is sent to RoutePointListener>

    // save call ani for use in routeCall() method
    m_callAnis.put(callId, callAni);

    Message mesg = m_msg_util.removeMessage(callAni);
    if (mesg != null)
    {
    mi = mesg.get(Message.OCCT_OTAS_KEY_MEDIA_ITEM_ID);
    m_msgUtil.storeMessage(callId, mesg);
    }

    // Insert implementer's code...
    }

    public void fireCallAbandonedEvent(...)
{
    String callId = <get call id that is sent to RoutePointListener>
    m_msgUtil.removeStoredMessage(callId);
    // Remove the entry from m_callAnis hashtable to prevent memory problems.
    m_callAnis.remove(callId);

    // Insert implementer's code...
}

public void fireCallDivertedEvent(...)
{
    String callId = <get call id that is sent to RoutePointListener>
    m_msgUtil.removeStoredMessage(callId);
    // Remove the entry from m_callAnis hashtable to prevent memory problems.
    m_callAnis.remove(callId);

    // Insert implementer's code...
}
```

7

Testing the Telephony Adapter

The Oracle Telephony SDK Integrated Test Utility is a tool for testing the Telephony Adapter implementation, which is loaded and run by the Oracle Telephony Adapter Server. The Telephony SDK Integrated Test Utility contains user interfaces for running the Oracle Telephony Adapter Server, the Verification Tool, the TeleDevice Test Utility, Switch Simulator, and the Javadoc.

Developers can use the Oracle Telephony SDK Integrated Test Utility to test the full functions of the TeleDevice implementations.

The Oracle Telephony SDK Integrated Test Utility interface has five tabs:

- Oracle Telephony Adapter Server, for configuring and running the Oracle Telephony Adapter Server
- Verification Tool, for configuring and running the TeleDevices and softphone
- TeleDevice Test Utility, for configuring and running the TeleDevice and softphone
- Switch Simulator, for configuring and running the switch simulator
- Help, which displays the Javadoc

This section includes the following topics:

- Section 7.7, "Javadoc"

7.1 Starting and Stopping the Test Utility

Use the following procedures to start and stop the Oracle Telephony SDK Integrated Test Utility.

Log in

Not applicable

Responsibility

Not applicable

Prerequisites

1. Obtain the Oracle Telephony SDK Zip file.
2. Update the SDK_JRE_HOME environment variable in the file sdkenv.cmd to point to a JDK installation, such as D:/jdk1.2.2.

Steps

1. Navigate to the directory oracle/apps/cct/bin.
2. Run the file sdkrun.cmd.

   The Oracle Telephony Adapter SDK Integrated Test Utility appears, opened to the Oracle Telephony Adapter Server tab > Server sub tab.
3. To stop the Oracle Telephony SDK Integrated Test Utility, close the Oracle Telephony Adapter SDK Integrated Test Utility window.

7.2 Using Log Windows

By default each main tab (excluding Help) of the Oracle SDK Integrated Test Utility includes a Log window that shows the progress and completion of configurations. To view the log messages in a separate, individual window, right click in the Log window and select **Undock**. The Log window detaches as a smaller, separate window that contains the log messages.

To close the separate log window and restore the log file to the main Log window, right click in the separate window and select **Dock**.

To clear log messages, right click in the selected log window and select **Clear**.

7.3 Oracle Telephony Adapter Server

The Oracle Telephony SDK Integrated Test Utility's Oracle Telephony Adapter Server user interface consists of the Server tab, to start and stop Oracle Telephony Adapter Server and to configure and view logs, and the Middleware tab, to configure CTI middleware and start Oracle Telephony Adapter Server.

> **Note:** Oracle Telephony Adapter Server cannot be started in the Middleware tab.

This section includes the following topics:

- Section 7.3.1, "Configuring the Oracle Telephony Adapter Server Log"
- Section 7.3.2, "Configuring Middleware"
- Section 7.3.3, "Starting and Stopping Oracle Telephony Adapter Server"

See Also

Section 7.1, "Starting and Stopping the Test Utility"

7.3.1 Configuring the Oracle Telephony Adapter Server Log

Use the following procedure to configure the Oracle Telephony Adapter Server log.

Log in

Not applicable

Responsibility

Not applicable

Prerequisite

Install and configure Oracle Telephony Adapter SDK.

Steps

1. Select the Oracle Telephony Adapter Server tab > Server sub tab.
2. From the Log Level list, select one of the four log levels:
   - error, to print only error messages

- warning, to print error and warning messages
- info, to print error, warning and information messages
- verbose, to print all messages

3. In the Log Directory field, enter the directory path of the log file. The log file created by Oracle Telephony Adapter Server is named using the pattern OTASyyyymmdd_hhmmss.log. The default directory is oracle/apps/cct/bin/log.
4. Select **File > Save** to save the configuration.

7.3.2 Configuring Middleware

In the Oracle Telephony SDK Integrated Test Utility, the Oracle Telephony Adapter Server middleware configuration consists of three types of data:

- Data that Oracle Telephony Adapter Server requires to load the correct type of adapter, which includes the Middleware Type, Library Name and Factory Class Name.
- Custom data that the adapter requires to be connected to the third-party CTI middleware systems, which includes IP addresses, ports and custom attributes. This data is passed to the TeleDeviceFactory.init() method.
- Dialing data in the Dialing section that Oracle Telephony Manager requires to generate the local dial string based on the location from which the call is placed. Typically, developers are not required to use the data in the Dialing section.

Use the following procedure to configure middleware.

Log in

Not applicable

Responsibility

Not applicable

Prerequisite

Install and configure Oracle Telephony Adapter SDK.

Steps

1. In the Oracle Telephony SDK Integrated Test Utility, select the Oracle Telephony Adapter Server tab > Middleware sub tab.

The Middleware page appears.

2. Do one of the following:

   For an adapter developed in Java:

   – From the Middleware Type list, choose **Java Adapter**.

   – In the Factory Class Name field, enter the TeleDeviceFactory class name.

   For an adapter developed in C:

   – From the Middleware Type list, choose **C Adapter**.

   – In the Library Name field, enter the DLL library name.

3. All other fields are passed to the TeleDeviceFactory.init() method as key-value pairs stored in a Hashtable. The hashtable keys used for each field are defined in interface Constants (for Java) and occt_pub.h (for C) and are listed in the following table.

Table 7–1 Oracle Telephony Adapter Server Field and Hashtable Keys

Field	Hashtable Key (Java)
Passive mode	KEY_PASSIVE_MODE
CTI Server IP Address 1	KEY_CTI_SERVER_IP_1
CTI Server IP Address 2	KEY_CTI_SERVER_IP_2
CTI Server Port 1	KEY_CTI_SERVER_PORT_1
CTI Server Port 2	KEY_CTI_SERVER_PORT_2
Adapter Server Info 1	KEY_ADAPTER_SERVER_INFO_1
Adapter Server Info 2	KEY_ADAPTER_SERVER_INFO_2
Adapter Server Info 3	KEY_ADAPTER_SERVER_INFO_3
Adapter Server Info 4	KEY_ADAPTER_SERVER_INFO_4
Adapter Server Info 5	KEY_ADAPTER_SERVER_INFO_5
Adapter Server Info 6	KEY_ADAPTER_SERVER_INFO_6
Domestic Dialing Prefix	KEY_DOMESTIC_PREFIX
International Dialing Prefix	KEY_IDD_PREFIX
Outgoing Prefix	KEY_OUTGOING_PREFIX

Table 7–1 Oracle Telephony Adapter Server Field and Hashtable Keys (Cont.)

Field	Hashtable Key (Java)
Site Overlay	KEY_SITE_OVERLAY
Site Area Code	KEY_SITE_AREA_CODE
Site Country Code	KEY_SITE_COUNTRY_CODE
Site Internal Number Length	KEY_SITE_INTERNAL_NUM_LENGTH
Site Local Number Maximum Length	KEY_SITE_LOCAL_NUM_MAX_LENGTH

4. Select **File > Save** to save the configuration.

7.3.3 Starting and Stopping Oracle Telephony Adapter Server

Use the following procedures to start and stop the Oracle Telephony Adapter Server of the Oracle Telephony SDK Integrated Test Utility.

Log in

Not applicable

Responsibility

Not applicable

Prerequisites

1. Install and configure Oracle Telephony SDK.
2. Configure the CTI middleware.

Steps

To start Oracle Telephony Adapter Server, use the following procedure.

1. In the Oracle Telephony Adapter Server window select the Server tab.
2. Click **Start**.

   Oracle Telephony Adapter Server launches as a background Java process and uses the current middleware configuration to start. A log file appears in the Oracle Telephony Adapter Server Log window.

Note: Any configuration changes made while Oracle Telephony Adapter Server is running will not take effect until Oracle Telephony Adapter Server is stopped and restarted.

To stop Oracle Telephony Adapter Server, use the following procedure.

1. In the Oracle Telephony Adapter Server window select the Server tab.
2. Click **Stop**.

   The Oracle Telephony Adapter Server background process terminates. The log file remains open in the Oracle Telephony Adapter Server Log window.

7.4 Verification Tool

The Verification Tool automates testing the adapter implementation by performing the following functions:

- Runs a series of common test cases and validates an implementation
- Reads a test case file written in proprietary Oracle internal syntax
- Executes the series of steps in the test case sequentially
- Waits and collects the events after each step in the test case
- Verifies the collected set of events with the expected set of events (from the solutions file) for every test step
- Prints the result of the verifications in the Results file and in the log window

This section contains the following topics:

- Section 7.4.1, "Configuring the Verification Process"
- Section 7.4.2, "Configuring Agent Information"
- Section 7.4.3, "Starting and Stopping the Verification Tool"

See Also

Section 7.1, "Starting and Stopping the Test Utility"

7.4.1 Configuring the Verification Process

Use the following procedure to configure the settings in the Verification Tool Control tab, including starting and stopping the verification process, and configuring input and output files and log levels.

Log in

Not applicable

Responsibility

Not applicable

Prerequisites

Implement the telephony adapter.

Steps

1. Select the Verification Tool tab > Control sub tab.

   The Control page appears.

2. In the Solutions File field, enter or browse for the default solution file Solutions.txt. The path is oracle/apps/cct/scripts directory. To use a different solutions file, enter the full path and file name of the file in the Solutions File box. The solutions file contains the expected events and event details for every SDK API invocation and compares the collected events at runtime with the expected events.

3. In the Test Cases File field, enter or browse for the default test case file TestCases.txt in the directory oracle/apps/cct/scripts. The Test Cases file contains the listing of all the test cases that the Verification Tool will execute. Each line in this file points to a specific script file. Each script file name should be either the absolute file name or relative to the bin directory of the installation. Optionally, to use a different test case file, enter the full path and file name of the file in the Test Cases File field. You can edit the TestCases.txt file to modify the order of Script files to execute, and add or drop a script file to execute.

4. In the Results File field, enter or browse for the default results file output.txt. The Results File is the output file to which the Verification Tool writes results. Optionally, to use a different results file, enter the full path and file name of the file in the Results File field.

5. From the Log Level list, select one of the four levels of logging sensitivity.
   - Error: Error messages only
   - Warning: Error and warning messages
   - Info: Error, warning and informational messages
   - Verbose: All messages
6. From the Event Delay Threshold list select the number of seconds that the Verification Tool must wait between executing every two steps in a script. Set the value of the EventDelay Threshold to correspond to greater than or equal to the time delay between command execution and event reception for your telephony platform.
7. Select **File > Save** to save the configuration.

7.4.2 Configuring Agent Information

Use the following procedure to configure the Verification Tool settings in the Configure tab, including the agent and teleset and route point numbers that are required to run the tool.

Prerequisites

- Implement the telephony adapter.
- Configure the Verification Tool Control tab settings.

Steps

1. Select the Verification Tool tab > Configure sub tab.

   The Configure page appears.
2. The Verification Tool uses three telesets (Agent A, Agent B and Agent C) for most of the test cases. In the Line*n* Extn# fields, enter the line extension number of each teleset.
3. For each teleset, enter the AgentLoginId, AgentPassword, and AgentQueue. The Verification Tool uses these configuration values to log in to each teleset before executing the scripts.
4. Some test cases involve outbound calls to external numbers and inbound calls from external numbers. In the External Numbers fields, enter external telephone numbers which the Verification Tool can use to make calls and

answer calls to verify the ANI and DNIS values for the events collected in those test cases.

5. Some test cases involve making calls to Busy Numbers and Invalid Numbers. In the Busy & Invalid Numbers fields enter the telephone numbers that the Verification Tool can use to verify the ANI and DNIS values for the events collected in those test cases.
6. The Verification Tool uses two route points, one active and one passive, for testing the RoutePointDevice implementations. In the Active RP # field, enter the extension number of an active route point. In the Passive RP # field, enter the extension number of a passive route point.
7. Select **File > Save** to save the configuration.

7.4.3 Starting and Stopping the Verification Tool

To start the Verification Tool, use the following procedure.

Log in

Not applicable

Responsibility

Not applicable

Prerequisite

Install the Verification Tool.

Steps

1. Select the Verification Tool tab > Control sub tab.
2. Click **Start**.

   The Verification Tool uses the current configuration and starts as a background process. The log file appears in the Verification Tool Log window.

> **Note:** Any changes you make while the Verification Tool is running will not take effect until you stop and restart the Verification Tool.

To stop the Verification Tool, use the following procedure.

1. Select the Verification Tool tab > Control sub tab.
2. Click **Stop**.

   The background process terminates.

7.5 TeleDevice Test Utility

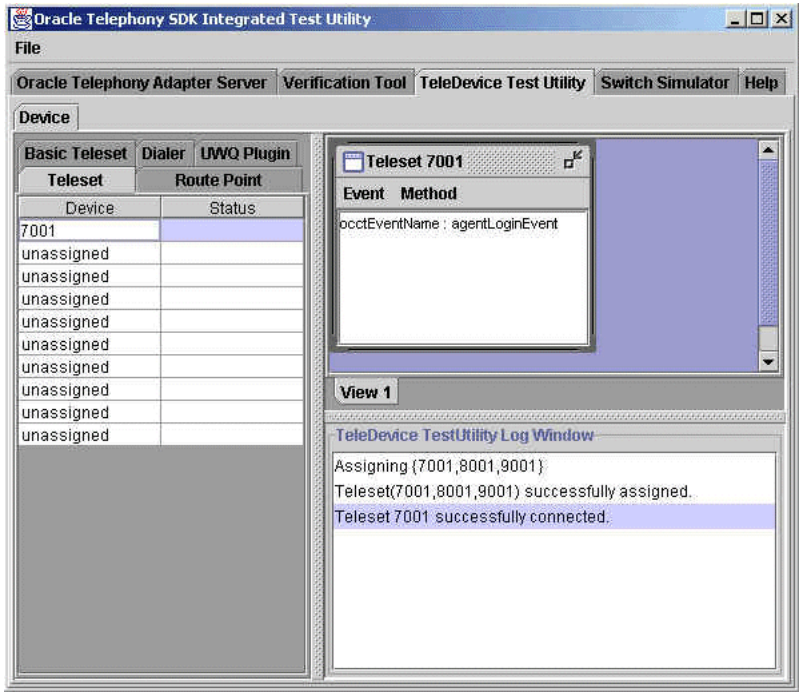
TeleDevice Test Utility is an interface for controlling and monitoring TeleDevices. Developers can use TeleDevice Test Utility as a client to Oracle Telephony Adapter Server, and invoke API methods on TeleDevice implementations. Events that are sent by TeleDevice implementations are received by TeleDevice Test Utility and displayed on the user interface. Additionally, developers can use TeleDevice Test Utility to launch the softphone and perform unit testing for TelesetDevice implementations.

The user interface consists of the following components:

- Left device tab for configuring and starting TeleDevice clients
- Top right side contains multiple internal frames (one per Device)
- Bottom right side message window for displaying messages

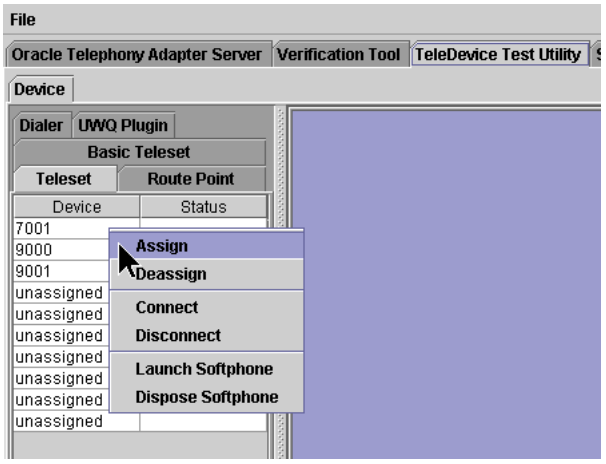
These components are illustrated in the following figure.

Figure 7–1 TeleDevice Test Utility Interface



To configure a device, right click on it to open a menu and select an option: Assign, Deasign, Connect, Disconnect, Launch Softphone, or Dispose Softphone, as shown in the following illustration.

Figure 7–2 TeleDevice Configuration Options



This section contains the following topics:

7.5.1 TeleDevices

Each TeleDevice simulates a communication channel between an Oracle component and the third-party component, through the different SDK interfaces.

Teleset

Represents a Teleset device in Oracle Telephony Adapter Server, and simulates the channel between Oracle Telephony Manager and Oracle Telephony Adapter Server. Use this TeleDevice only for Standard or Advanced Integration.

Route Point

Represents a Route Point device in Oracle Telephony Adapter Server, and simulates the channel between Inbound Telephony Server and Oracle Telephony Adapter Server. This TeleDevice should only be used for Advanced Integration.

Dialer

Represents a Dialer device in Oracle Telephony Adapter Server, and simulates the channel between Advanced Outbound Dial Server and Oracle Telephony Adapter Server. Use this TeleDevice only for Standard or Advanced Integration with Oracle Advanced Outbound.

Basic Teleset

Represents a Teleset device in a third party provider's process, and simulates the channel between Media Provider Plug-in (in the Universal Work Queue Client process space) and the third party provider. Use this TeleDevice only for Basic Integration.

UWQ Plug-in

Represents an Oracle Universal Work Queue Client plug-in, and simulates the channel between Oracle Universal Work Queue Client, and the third-party provider. Use this TeleDevice only for Basic Integration.

7.5.2 Assigning TeleDevices

Each of the Oracle SDK Integrated Test Utility TeleDevice tabs has a table of ten rows. Each row represents a TeleDevice object. The TeleDevice objects must be associated to specific configurations to be able to connect to Oracle Telephony Adapter Server. This association is called "Assign" in the TeleDevice Test Utility. Developers can assign up to ten devices.

This section contains the following topics:

- Section 7.5.2.1, "Assigning TelesetDevice"
- Section 7.5.2.2, "Assigning Route Point Device"

- Section 7.5.2.3, "Assigning Dialer"
- Section 7.5.2.4, "Assigning Basic Teleset"
- Section 7.5.2.5, "Assigning UWQ Plug-in"

7.5.2.1 Assigning TelesetDevice

Use the following procedure to assign a TelesetDevice.

Log in

Not applicable

Responsibility

Not applicable

Prerequisites

Successfully start Oracle Telephony Adapter Server.

Steps

1. Select the TeleDevice Test Utility tab.

   The Device page appears.

2. Select the Teleset sub tab.
3. Select the Device or Devices to assign.
4. Right click on the selected rows.

   A menu appears.

5. Choose **Assign**.

   The Assign Teleset window appears.

6. Enter the three extension numbers.
7. Click **OK**.

   The TeleDevice Test Utility Log window displays the progress and completion of the assignments.

7.5.2.2 Assigning Route Point Device

Use the following procedure to assign a Route Point Device.

Log in

Not applicable

Responsibility

Not applicable

Prerequisite

Successfully start Oracle Telephony Adapter Server.

Steps

1. Select the TeleDevice Test Utility tab.

   The Device page appears.

2. Select the Route Point sub tab.

3. Select the Device or Devices to assign.

4. Right click on the selected rows.

   A menu appears.

5. Choose **Assign**.

   The Assign RoutePoint window appears.

6. Enter the Route Point Number.

7. Optionally, if you want this route point to receive route requests, click **Route Requested**.

8. For Nortel Meridian with Intel CT Connect/NetMerge Call Processing Software only, in the Immediate Treatment field specify the immediate treatment of inbound calls arriving at this route point CDN (Control Directory Number). Enter ##R for ringback, ##M for music, or ##S for silence.

9. For Nortel Meridian with Intel CT Connect/NetMerge Call Processing Software only, if music treatment (##M) is specified in step 7, in the Music Route Number field specify the route number of a music source that is configured in the Meridian PBX. Enter # followed by a two-digit route number specified in hexidecimal. For example, if the music route number is 10, then enter #0A in the Music Route Number field.

10. Click **OK**.

The TeleDevice Test Utility Log window displays the progress and completion of the assignments.

7.5.2.3 Assigning Dialer

Use the following procedure to assign a Dialer.

Log in

Not applicable

Responsibility

Not applicable

Prerequisite

Successfully start Oracle Telephony Adapter Server.

Steps

1. Select the TeleDevice Test Utility tab.

   The Device page appears.

2. Select the Dialer sub tab.

3. Select the Device or Devices to assign.

4. Right click on the selected rows.

   A menu appears.

5. Choose **Assign**.

   The Assign Dialer window appears.

6. Enter the Dialer ID and Vdu ID. Dialer ID is the ID to uniquely identify this Dialer Device. Vdu ID is the ID to identify the Vdu that the Dialer Device will control.

7. Press **OK**.

   The TeleDevice Test Utility Log window displays the progress and completion of the assignments.

7.5.2.4 Assigning Basic Teleset

Use the following procedure to assign a Basic Teleset.

Log in
Not applicable

Responsibility
Not applicable

Prerequisite
Successfully start Oracle Telephony Adapter Server.

Steps
1. Select the TeleDevice Test Utility tab.

   The Device page appears.
2. Select the Basic Teleset sub tab.
3. Select the Device or Devices to assign.
4. Right click on the selected rows.

   A menu appears.
5. Choose **Assign**.

   The Assign Basic Teleset window appears.
6. Enter the 3rd Party url, Agent ID, Teleset ID and Listener Port.
7. Click **OK**.

   The TeleDevice Test Utility Log window displays the progress and completion of the assignments.

7.5.2.5 Assigning UWQ Plug-in
Use the following procedure to assign an UWQ Plug-in.

Log in
Not applicable

Responsibility
Not applicable

Prerequisite

Successfully start Oracle Telephony Adapter Server.

Steps

1. Select the TeleDevice Test Utility tab.

   The Device page appears.

2. Select the UWQ Plug-in sub tab.

3. Select the Device or Devices to assign.

4. Right click on the selected rows.

   A menu appears.

5. Choose **Assign**.

   The Assign UWQ Plug-in window appears.

6. Enter the Agent ACD ID, Agent ACD Password, Agent ACD Queue, Listener Port, 3rd Party url, Reconnect Interval(sec), and Log Level.

   - Agent ACD ID: The Agent ACD ID to log in to the switch
   - Agent ACD Password: The Agent ACD Password to login into the switch
   - Agent ACD Queue: The Agent ACD Queue to log in to the switch
   - Listener Port: The Listener port for Basic SDK HTTP Listener
   - 3rd Party url: Third-party URL implementing the Callout handling
   - Reconnect Interval: Number of seconds before the Basic SDK Plug-in tries to ping or reconnect to the third-party URL
   - Log Level: Error, warning, info, or verbose

7. Click **OK**.

   The TeleDevice Test Utility Log window displays the progress and completion of the assignments.

7.5.3 Connecting and Disconnecting a TeleDevice

When a TeleDevice is assigned, the next step is to connect it to Oracle Telephony Adapter Server. Use the following procedure to connect a TeleDevice.

Note: Check that Oracle Telephony Adapter Server is started and running before connecting a TeleDevice.

Log in

Not applicable

Responsibility

Not applicable

Prerequisite

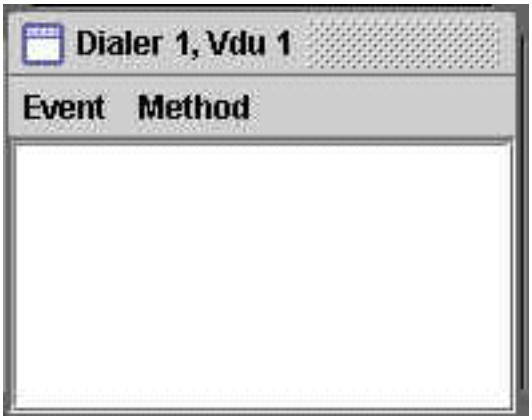
Assign at least one TeleDevice.

Steps

1. Select the appropriate TeleDevice tab: Dialer, UWQ Plug-in, Basic Teleset, Teleset, or Route Point.
2. Right click on the TeleDevice that you want to connect.

   A menu appears.
3. Choose **Connect**.
4. If the TeleDevice has connected successfully, a TeleDevice internal window appears, as shown in the following illustration.

Figure 7–3 TeleDevice Window



The message window shows that "RoutePoint X successfully connected" or "Teleset X successfully connected." Other TeleDevices display similar windows and messages.

See Also

Section 7.1, "Starting and Stopping the Test Utility"

7.5.4 Deassigning a TeleDevice

Removing a device assignment is called *deassigning*. Use the following procedure to deassign a TeleDevice.

Log in

Not applicable

Responsibility

Not applicable

Prerequisite

Assign at least one TeleDevice.

Steps

1. Select the appropriate TeleDevice tab (Dialer, UWQ Plug-in, Basic Teleset, Teleset, or Route Point).
2. Right click on the TeleDevice that you want to deassign.

   A menu appears.
3. Select **Deassign**.

   The configuration associated with the row is deassigned and removed.

7.5.5 Viewing Events

In the Event History window you can view up to thirty events that are sent by TeleDevice object implementations and cached.

To open the Event History window, open a device message window (see Section 7.5.3, "Connecting and Disconnecting a TeleDevice") and select **Event** > **Event History**, as shown in the following illustration.

Figure 7–4 Event History Option

The Event History window opens, showing any event messages.

7.5.6 Invoking API Methods

You can invoke API methods for TelesetDevice, RoutePointDevice, DialerDevice, BasicTelesetDevice, and UWQPluginDevice.

Use the following procedures to invoke API methods for TeleDevices.

Log in

Not applicable

Responsibility

Not applicable

Prerequisites

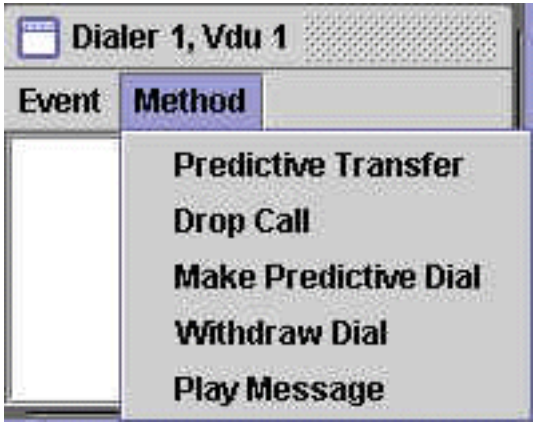
Connect TeleDevice.

Steps

1. In an opened TeleDevice internal message window, select **Method**.

   The Methods menu appears, as shown in the following illustration for the Dialer method.

Figure 7–5 Methods Options



2. Select a method.

For some methods, no result is apparent. For other methods, a window appears.

3. If a window appears, then enter the required information in the fields.
4. Click **OK**.

7.5.6.1 Route Point

RoutePointDevice invokes the following methods:

- assignMediaItemId
- routeCall

See Also

- Section 5.8, "Implementing RoutePointDevice"
- Section 5.1, "Advanced Integration"

7.5.6.2 Teleset

TelesetDevice invokes the following methods:

- makeCall
- answerCall
- releaseCall
- holdCall
- retrieveCall
- loginAgent
- logoutAgent
- agentReadyOn
- agentReadyOff
- blindTransfer
- consultationCall(Transfer)
- completeTransfer
- consultationCall(Conference)
- completeConference

See Also

- Section 5.7, "Implementing TelesetDevice"
- Section 5.1, "Advanced Integration"
- Section 5.2, "Standard Integration"

7.5.6.3 Dialer

DialerDevice invokes the following methods:

- Predictive Transfer
- Drop Call
- Make Predictive Dial
- Withdraw Dial
- Play Message

See Also

- Section 10.2, "Implementing Advanced Outbound Extension"
- Section 5.1, "Advanced Integration"
- Section 5.2, "Standard Integration"

7.5.6.4 UWQ Plug-in

UWQ Plug-in invokes the following methods:

- Make Call
- Next Work
- Cancel Work Request
- Login Agent
- Logout Agent

See Also

Section 9.4, "Implementing Basic Integration"

7.5.6.5 Basic Teleset

Basic Teleset invokes the following methods:

- Make Call
- Agent Ready
- Agent Not Ready
- Login Agent
- Logout Agent
- Register Agent
- Unregister Agent

See Also

Section 9.4, "Implementing Basic Integration"

7.5.7 Disconnecting a TeleDevice

Use the following procedure to disconnect a TeleDevice.

Log in

Not applicable

Responsibility

Not applicable

Prerequisite

Connect a TeleDevice.

Steps

1. Select the appropriate TeleDevice tab (Dialer, UWQ Plug-in, Basic Teleset, Teleset, or Route Point).
2. Right click on the TeleDevice that you want to disconnect.

   A menu appears.
3. Choose **Disconnect**.

7.5.8 Launching the Softphone

To verify the softphone functionality, use the following procedures to launch and close the softphone.

Log in

Not applicable

Responsibility

Not applicable

Prerequisite

Assign a Teleset Device.

Steps

1. Select the TeleDevice Test Utility tab > Teleset sub tab.
2. In the Device column, right click on the applicable, assigned teleset device number.

   A menu appears.
3. Select **Launch Softphone**.

   The loginAgent window appears.
4. Enter the acdAgentId, acdAgentPassword and acdQueue.
5. Click **OK**.

   The softphone interface opens.

7.5.9 Closing the Softphone

Use the following procedure to close the softphone.

Log in

Not applicable

Responsibility

Not applicable

Prerequisite

None

Steps

1. Select the TeleDevice Test Utility tab > Teleset sub tab.
2. In the Device column, right click on the applicable teleset device number.

   A menu appears.
3. Select **Dispose Softphone**.

   The Agent Logout window appears.
4. Click **OK**.

   The softphone interface closes.

7.6 Switch Simulator

The Switch Simulator is one of the Interaction Center server processes that is used in demonstration environments where a real switch is not available. The switch simulator supports teleset and route point functions, and three integration levels: Basic, Standard and Advanced.

In Standard and Advanced Integration, the Switch Simulator acts as a simulated switch that listens for Oracle Telephony Adapter's Server's teleset and route point commands (makeCall) and fake events (callRinging) to the appropriate teleset or route point extensions.

In Basic Integration, the Switch Simulator acts as a third party HTTP listener that listens for callouts from the Basic Telephony SDK plug-ins and fake HTTP and XML events to the appropriate Basic Telephony SDK plug-ins.

The Switch Simulator interface consists of two tabs:

- Server tab, to start and stop the Switch Simulator server, and to configure and view logs.
- Configure tab, the extension configuration for starting the Switch Simulator server.

This section includes the following topics:

- Section 7.6.3, "Configuring Simulated Extensions and Route Points"
- Section 7.6.4, "Configuring the Switch Simulator for Standard or Advanced Integration"

See Also

Section 7.1, "Starting and Stopping the Test Utility"

7.6.1 Starting and Stopping the Switch Simulator

Use the following procedures to start or stop the Oracle SDK Integrated Test Utility Switch Simulator.

Log in

Not applicable

Responsibility

Not applicable

Prerequisite

Configure the Switch Simulator.

Steps

1. In the Oracle SDK Integrated Test Utility, select the Switch Simulator tab > Server sub tab.

   The Server page opens.

2. Do one of the following:

   - To start the Switch Simulator, click **Start**.

     The Switch Simulator launches as a background Java process, and starts by using the current extension configuration.

   - To stop the Switch Simulator, click **Stop**.

     The Switch Simulator background process terminates.

7.6.2 Configuring Switch Simulator Server Settings

Use the following procedure to configure the Switch Simulator server settings.

Log in

Not applicable

Responsibility

Not applicable

Prerequisites

None

Steps

1. In the Oracle SDK Integrated Test Utility, select the Switch Simulator tab > Server sub tab.

   The Server page opens.

   Note: Any changes you make while the Switch Simulator is running do not take effect until you stop and restart the Switch Simulator.

2. Choose the extent of detail in the Switch Simulator Log by selecting a logging level:
   - error: Print only error messages.
   - warning: Print error and warning messages.
   - info: Print error, warning and informational messages.
   - verbose: Print all messages.
3. Optionally, you can configure the directory of the log file by typing the directory path into the Log Directory text box. To name the log file, use the pattern swsimyyyymmdd_hhmmss.log. The default directory is oracle/apps/cct/bin/log.
4. Configure the port number of the Switch Simulator by entering the port number in the Server Port field. This is the port number that Oracle Telephony Adapter Server uses to communicate with the Switch Simulator for Standard and Advanced Integration.
5. Configure the port number of the Switch Simulator HTTP listener by entering the port number in the HTTP Server Port text box. This is the port number that

UWQ Client uses to communicate with the Switch Simulator in Basic Integration.

6. Select **File** > **Save** to save the configurations.

7.6.3 Configuring Simulated Extensions and Route Points

Use the following procedure to configure simulated extensions and route points.

Log in

Not applicable

Responsibility

Not applicable

Prerequisites

None

Steps

1. Select the Switch Simulator tab > Configure sub tab.

   The Extensions and Route Points page appears.

2. In the Extension Range fields, enter extension ranges 1 through 3 for simulated telesets.
3. In the three Route Point fields, enter route point extensions for simulated route points.
4. Select **File** > **Save** to save the configurations.

7.6.4 Configuring the Switch Simulator for Standard or Advanced Integration

Use the following procedure to configure the Oracle SDK Integrated Test Utility's Oracle Telephony Adapter Server settings for Standard Integration or Advanced Integration.

Log in

Not applicable

Responsibility

Not applicable

Prerequisites

- Configure Oracle Telephony Adapter Server to connect to the Switch Simulator.
- Configure the Switch Simulator.

Steps

1. Select the Oracle Telephony Adapter Server tab > Middleware sub tab.

   The Middleware page opens.

2. From the Middleware Type list, select **Java Adapter**.
3. In the Factory Class Name field, enter

   ```
   oracle.apps.cct.sdk.sample.advanced.SampleTeleDeviceFactory
   ```

4. In the CTI Server IP Address 1 field, enter the IP Address of the machine on which the Switch Simulator starts.
5. In the CTI Server IP Port 1 field, enter the Server Port (*not* the HTTP Server Port) of the Switch Simulator.
6. Select the TeleDevice Test Utility tab > Device sub tab.

   The Device page opens.

7. In Teleset Device fields 1 through 3, assign extensions within the range that are the same as those configured in the Switch Simulator.
8. In the Route Point Status fields, assign Route Point Number to be one of the Route Point extensions that are configured in the Switch Simulator.
9. Select **File** > **Save** to save the configuration.

7.6.5 Configuring the Switch Simulator for Basic Integration

When developing Basic Integration, the Switch Simulator acts as the third-party provider that listens for callouts from UWQ Client. The Basic Teleset and UWQ Plug-in TeleDevices can be used to simulate connectivity between UWQ Client and the Switch Simulator.

Use the following procedure to configure the Oracle SDK Integrated Test Utility's Switch Simulator for Basic Integration.

Log in
Not applicable

Responsibility
Not applicable

Prerequisites
Configure the Switch Simulator.

Steps
1. Select the TeleDevice Test Utility tab > Device sub tab.

   The Device page opens.

2. Do *either* a or b:

   a. Select the Basic Teleset sub tab.

      – Right click on a Device that you want to configure.

        A menu appears.

      – Select **Assign**.

        The Assign Basic Teleset window opens.

      – In the 3rd Party url field, enter

        ```
        http://<IP Address of Switch Simulator>:<HTTP Server Port of Switch Simulator>/
        ```

   b. Select the UWQ Plug-in tab.

      – Right click on a Device that you want to configure.

        A menu appears.

      – Select **Assign**.

        The Assign UWQ Provider Plug-in window opens.

      – In the 3rd Party url field, enter

        ```
        http://<IP Address of Switch Simulator>:<HTTP Server Port of Switch Simulator>/
        ```

7.7 Javadoc

Javadoc is a facility provided within the Java Development Kit that produces HTML documentation from a program. Javadoc reads the source code and parses specially formatted and positioned comments into documentation.

To access the Javadoc for Oracle Telephony Adapter SDK, select the Help tab in the Oracle Telephony SDK Integrated Test Utility.

See Also

Section 7.1, "Starting and Stopping the Test Utility"

8

Deploying the Telephony Adapter

This section describes the steps involved in packaging and uploading Oracle Telephony Adapter to deploy the implementation in a live interaction center environment. The following steps outline the deployment.

- Package the Telephony Adapter.
- Upload the Telephony Adapter to Interaction Center Server Manager.
- Configure CTI Middleware, Telesets and Route Points.
- Configure the Server Group.
- Test with Interaction Center, softphone and Oracle Universal Work Queue.

Prerequisites

- Install Oracle eBusiness Suite.
- Install Oracle Interaction Center Server Manager.

Steps

1. Package the telephony adapter. The type of file package for Oracle Telephony Adapter depends on the development language of the adapter implementation.

   Do one of the following:

   - For an adapter developed in Java, all Java classes should be packaged into a single jar file using the Java JAR utility, such as impl.jar.
   - For an adapter developed in C, compile all code into a single DLL file, such as impl.dll.

2. Identify the node where Oracle Telephony Adapter Server will be deployed.

3. Copy the jar or dll file to the directory "3rdParty" that is under the directory where Interaction Center Server Manager is installed.
4. Configure CTI middleware, telesets and route points. Define a CTI middleware configuration before Oracle Telephony Adapter Server can start loading the adapter implementation. Do one of the following:
   - For a Java adapter implementation, use the Custom Java Adapter middleware type.
   - For a C adapter implementation, use the Custom C Adapter middleware type.
5. Configure the server group. Define Oracle Telephony Adapter Server and related servers before starting Oracle Telephony Adapter Server.

See Also

- *Oracle Interaction Center Implementation Guide*
- *Oracle Advanced Inbound Implementation Guide*

9

Basic Integration

This chapter describes Oracle Telephony Adapter SDK Basic Integration, and explains how to use the infrastructure and facilities of Basic Integration to integrate the Oracle eBusiness Suite with a third-party telephony system.

This information is intended for developers and consultants who intend to use Oracle Telephony Adapter SDK Basic Integration to integrate Oracle eBusiness Suite with a third-party telephony vendor.

This chapter contains the following topics:

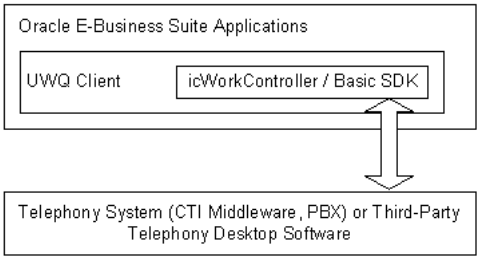
9.1 Architecture

Basic Integration of Oracle Telephony Adapter SDK consists of the following:

- Oracle Universal Work Queue Client
- Basic Telephony SDK plug-in

As the following diagram shows, Basic Integration integrates the telephony system (middleware, PBX), or third-party desktop software, with the Oracle Universal Work Queue client in the Oracle eBusiness Suite.

Figure 9–1 Basic Integration Architecture

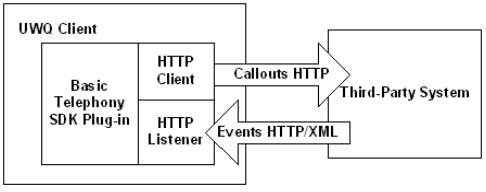


The Basic Telephony SDK plug-in enables third-party telephony systems to communicate with the Oracle eBusiness Suite by using one of two types of HTTP messages:

- Callout: Messages are sent from Oracle eBusiness Suite to the third-party system using a HTTP GET request.
- Event: Messages are sent from a third-party system using an HTTP POST request.

The following diagram illustrates sending callouts from the HTTP client to the third-party telephony system, and events from the third-party telephony system to the HTTP listener.

Figure 9–2 Callouts and Events



9.2 Understanding Basic Integration

The components of Basic Integration are explained in the following sections.

9.2.1 User Interface

The Basic Integration user interface includes the following components.

Oracle Universal Work Queue

On the Oracle Universal Work Queue form, "Basic Telephony" appears on the left Queue Selector Pane whenever Basic Telephony is enabled. Selecting Basic Telephony on the Queue Selector Pane displays the right Queue Details Pane. A row with the title "<ANY>" appears. When in the Basic Telephony mode, the Get Work button does not appear. Double clicking in the <ANY> row triggers Get Work.

icWork Controller

icWork Controller is a floating window that is used to Stop Media and to get Next Media.

See Also

Section 9.5, "Deploying Basic Integration"

9.2.2 User Actions and Corresponding Basic Telephony Integration Actions

The following paragraphs explain the Basic Integration actions and their corresponding user actions.

User Opens the Oracle Universal Work Queue Form

When a user opens the Oracle Universal Work Queue form, the basic framework elements set up and prepare Oracle Universal Work Queue for use. The Basic SDK client plug-in is loaded and performs the following actions:

- Loads the user profile values from the database and determines how to connect to the third-party system.
- Starts the HTTP listener on a randomly-allocated port or the optionally-specified port.
- Sends Register and Login callouts to the third-party system to establish a session.
- Starts the reconnect checking process.

User Selects the Basic Telephony Work Queue

The right Queue Details Pane displays <ANY>, and the Basic SDK client plug-in takes no action.

User Selects the Basic Telephony <ANY> Record

Basic SDK client plug-in sends Ready callout to the third-party system and sets itself in Get Work mode if there is no prior ScreenPop event received. Otherwise, the ScreenPop event is delivered.

User Selects End Interaction on eBusiness Suite Form

Basic SDK client plug-in sends Ready callout to the third-party system and sets itself in Get Work mode if there is no prior ScreenPop event received. Otherwise, the ScreenPop event is delivered.

User Selects Stop Media Button on icWork Controller

Basic SDK client plug-in sends NotReady callout to the third-party system and sets itself in non-Get Work mode.

User Closes Oracle Universal Work Queue Form

- The Basic SDK client plug-in sends Logout and Unregister to the third-party system.
- Basic SDK client plug-in stops HTTP listener and frees any resources.

User Receives an Inbound Call

- Basic SDK integration sends a ScreenPop event to the Basic SDK client plug-in.
- Basic SDK client plug-in delivers the data to Oracle Universal Work Queue for screen pop if it is in Get Work mode. Otherwise, it puts the event in its FIFO queue.
- Basic SDK client plug-in creates a media item record in Oracle Customer Interaction History.

User Answers an Inbound Call

- Basic SDK integration sends a CallEstablished event to the Basic SDK client plug-in.
- Basic SDK client plug-in creates a media item life cycle segment in Oracle Customer Interaction History.

User Releases an Inbound Call

- Basic SDK integration sends a CallReleased event to the Basic SDK client plug-in.
- Basic SDK client plug-in updates the media item life cycle segment in Oracle Customer Interaction History.

User Selects Dial on the Basic Panel

The dial string entered in the text field on Basic Panel is sent with MakeCall callout to a third-party system.

9.3 Software Requirements and Development Strategies

The following software requirements and development strategies apply to the Oracle Telephony Adapter SDK Basic Integration.

Minimum Patch Level

The minimal patch level required by Basic Telephony SDK is Release 11.5.8 + Interaction Center Family Pack P.

Programming Languages

Because the Basic Telephony SDK is defined in HTTP and XML, developers can use any programming language in which HTTP request processing can be implemented easily.

Development Strategies

The following two examples demonstrate different ways of developing Basic Telephony Integration.

- Client-based (recommended): In the client-based strategy, integration is done on client-side third-party software, such as a softphone, and is installed on each desktop machine. The Basic SDK communicates locally with the third-party software.
- Server based: In the server-based strategy, custom integration is done on a server-based platform and is installed centrally on the server. No client software is installed on the desktop machine. Basic SDK communicates with third-party software through the LAN.

9.4 Implementing Basic Integration

This section describes the implementation of Basic Integration.

9.4.1 Implementing Callouts

Callouts are messages sent from Oracle eBusiness Suite to the third-party system using HTTP GET requests. Developers must implement a HTTP listener that is capable of handling the HTTP GET requests specified in this document. The URL to the HTTP listener is then provided to the Basic SDK client plug-in through the use of a profile option. Parameters are passed to the URL using GET request semantics (CGI parameters). All callouts are handled by one single URL with the FunctionName parameter being a different value.

Register

Definition: Establish a session with the telesetNumber specified and notify third-party system on the IP address and port to which the Basic SDK client plug-in is listening.

Table 9–1 Register Parameters

Parameter Name	Value
FunctionName	Register
telesetNumber	telesetNumber entered by agent
agentID	ACD agent ID
ipAddress	ipAddress of the Basic SDK client plug-in HTTP Listener
port	Port of the Basic SDK client plug-in HTTP Listener

Example 9–1 URL

```
http://host:port/?FunctionName=Register&telesetNumber=1001&agentID=10001&ipAddre
ss=130.46.1.1&port=8088
```

Unregister

Definition: End the session with the telesetNumber specified.

Table 9–2 Unregister Parameters

Parameter Name	Value
FunctionName	Unregister
telesetNumber	the telesetNumber entered by agent
agentID	ACD agent ID

Example 9–2 URL

```
http://host:port/?FunctionName=Unregister&telesetNumber=1001&agentID=10001
```

Login

Definition: (Optional) Log in the agent to the specified Teleset Number. The third-party system or developer can decide whether to react to this callout, if manual login is desired.

Table 9–3 Login Parameters

Parameter Name	Value
FunctionName	Login
telesetNumber	the telesetNumber entered by agent
agentID	ACD agent ID
agentPassword	ACD password of the agent
agentQueue	ACD queue the agent should login to

Example 9–3 URL

```
http://host:port/?FunctionName=Login&telesetNumber=1001&agentID=10001&agentPassw
ord=123&agentQueue=1234
```

Logout

Definition: (Optional) Log out the agent to the specified Teleset Number. The third-party system or developer can decide whether to react to this callout.

Table 9–4 Logout Parameters

Parameter Name	Value
FunctionName	Logout
telesetNumber	the telesetNumber entered by agent
agentID	ACD agent ID
agentPassword	ACD password of the agent
agentQueue	ACD queue the agent should login to

Example 9–4 URL

```
http://host:port/?FunctionName=Logout&telesetNumber=1001&agentID=10001&agentPass
word=123&agentQueue=1234
```

Ready

Definition: Agent is ready for work.

Table 9–5 Ready Parameters

Parameter Name	Value
FunctionName	Ready
telesetNumber	TelesetNumber entered by agent
agentID	ACD agent ID

Example 9–5 URL

```
http://host:port/?FunctionName=Ready&telesetNumber=1001&agentID=10001
```

NotReady

Definition: Agent is not ready for work.

Table 9–6 NotReady Parameters

Parameter Name	Value
FunctionName	NotReady
telesetNumber	TelesetNumber entered by an agent
agentID ACD	ACD agent ID

Example 9–6 URL

```
http://host:port/?FunctionName=NotReady&telesetNumber=1001&agentID=10001
```

MakeCall

Definition: Make a call.

Table 9–7 MakeCall Parameters

Parameter Name	Value
FunctionName	MakeCall
telesetNumber	TelesetNumber entered by agent
agentID ACD	ACD agent ID
dialString	Destination number to dial

Example 9–7 URL

```
http://host:port/?FunctionName=MakeCall&telesetNumber=1001&agentID=10001&dialStr
ing=6509991111
```

CheckStatus

Definition: Check if the third-party System is up and running. This callout is called periodically by the Basic SDK client plug-in to detect lost connection and reconnect. You can configure the interval between each callout.

Table 9–8 CheckStatus Parameters

Parameter Name	Value
FunctionName	CheckStatus

Example 9–8 URL

```
http://host:port/?FunctionName=CheckStatus
```

9.4.2 Error Reporting

For each callout the developer must return a 200 HTTP response code indicating that the callout has processed the request. Successful callouts return a HTTP response with the 200 response code and no content. Errors in the HTTP response message may be reported using the following XML format:

```
<CCTSDK>
```

```
        <event name="Error">
            <data name="occtErrorMsg" value="error message here"/>
        </event>
</CCTSDK>
```

9.4.3 Implementing Events

Events are messages sent from the third-party system to the Basic SDK client plug-in using HTTP POST request. You can construct the URL of the Basic SDK client plug-in by using the IP address and port parameters from Register Callout. The request body must be an XML formatted message that is defined as follows:

```
<CCTSDK>
    <event name="eventName">
        <data name="name1" value="value1"/>
        <data name="name2" value="value2"/>
        <data name="name3" value="value3"/>
            ...
    </event>
</CCTSDK>
```

ScreenPop

Definition: ScreenPop event triggers a screen pop in the Oracle eBusiness Suite. Developers must determine when to send a ScreenPop event, typically when the agent gets an inbound call. Developers must decide whether to send a ScreenPop event for every call or only for certain types of calls. For example, internal or outbound calls are not often considered for screen pops.

Table 9–9 ScreenPop Parameters

Parameter Name	Value
occtSourceID	Logical ID used to connect between multiple events of the same physical call. Multiple events generated from the same physical call must have the same occtSourceID value, that is, telephony middleware call IDs. SourceID values must be a number. Valid values are 0 and positive and negative numbers with magnitude 1.0E-130 to 9.99..E125
occtANI	ANI. Must be <= 30 characters.
occtDNIS	DNIS. Must be <= 30 characters.
occtClassification (Optional)	Field passed to Oracle Universal Work Queue for determining the appropriate media action for this ScreenPop event. Must be <= 64 characters.

Example 9–9 XML

```
<CCTSDK>
<event name="ScreenPop">
<data name="occtSourceID" value="1"/>
<data name="occtANI" value="50666546"/>
<data name="occtDNIS" value="7404"/>
</event>
</CCTSDK>
```

CallEstablished (Optional)

Definition: Indicates that a call is established. CallEstablished event triggers the creation of a life cycle segment in Oracle Customer Interaction History. The start timestamp of the segment will be logged. This event must be sent after ScreenPop event.

Table 9–10 CallEstablished Parameters

Parameter Name	Value
occtSourceID	Logical ID used to connect between multiple events of the same physical call. Multiple events generated from the same physical call must have the same occtSourceID value, that is, telephony middleware call IDs. This value must be a number. Valid values are 0, and positive and negative numbers with magnitude 1.0E-130 to 9.99..E125

Example 9–10 XML

```
<CCTSDK>
<event name="CallEstablished">
<data name="occtSourceID" value="1"/>
</event>
</CCTSDK>
```

CallReleased (Optional)

Definition: Indicates a call is released. CallReleased event triggers the life cycle segment that is being updated in Oracle Customer Interaction History. The end timestamp and the duration of the segment are logged. This event must be sent after CallEstablished.

Table 9–11 CallReleased Parameters

Parameter Name	Value
occtSourceID	Logical ID used to connect between multiple events of the same physical call. Multiple events generated from the same physical call must have the same occtSourceID value, that is, telephony middleware call IDs. This value must be a number. Valid values are 0, and positive and negative numbers with magnitude 1.0E-130 to 9.99..E125

Example 9–11 XML

```
<CCTSDK>
<event name="CallReleased">
<data name="occtSourceID" value="1"/>
</event>
</CCTSDK>
```

9.4.4 Additional Interaction Keys

In addition to the keys specified in the ScreenPop event section, the following table lists keys that may also be sent by the third-party system as name/value pairs as part of the ScreenPop event. These keys may be used for out-of-the-box screen pops by Oracle Customer Care and Oracle TeleSales.

Table 9–12 Interaction Keys

Interaction Key	Description
ContactNum	Oracle TeleSales Key 3
PromotionCode	Oracle TeleSales Key 4
AccountCode	Oracle TeleSales Key 5
QuoteNum	Oracle TeleSales Key 6
CustomerID	Oracle TeleSales Key 7
Product	Oracle TeleSales Key 8
SystemName	Oracle TeleSales Key 9
ContractNum	Oracle TeleSales Key 10
PreferredID	Oracle TeleSales Key 11
ScreenPopType	Oracle TeleSales Key 12
occtCallBirthTime	Oracle TeleSales Key 13

Table 9–12 Interaction Keys (Cont.)

Interaction Key	Description
occtCallAnswerTime	Oracle TeleSales Key 14
CustomerProductID	Oracle TeleSales Key 15
InventoryItemID	Oracle TeleSales Key 16
InvoiceNum	Oracle TeleSales Key 17
LotNum	Oracle TeleSales Key 18
OrderNum	Oracle TeleSales Key 19
ProductName	Oracle TeleSales Key 20
PurchaseOrderNum	Oracle TeleSales Key 21
ReferenceNum	Oracle TeleSales Key 22
RevisionNum	Oracle TeleSales Key 23
RMANum	Oracle TeleSales Key 24
SerialNum	Oracle TeleSales Key 25
ServiceRequestNum	Oracle TeleSales Key 26
CustomerNum	Oracle TeleSales Key 27
CustomerName	Oracle TeleSales Key 28
occtANI	Oracle TeleSales Key 29
occtClassification	Oracle TeleSales Key 31
LanguageCompetency	Language Competency
KnowledgeCompetency	Knowledge Competency
ProductCompetency	Product Competency
occtMediaType	Media Type
employeeID	Employee ID
occtMediaItemID	Media Item ID
AccountNum	Account Number
SiteNum	Site Number
RepairNum	Repair Number
DefectNum	Defect Number

Table 9–12 Interaction Keys (Cont.)

Interaction Key	Description
CustomerStatus	Customer Status
EventCode	Event Registration Code
CollateralReq	Collateral Request Number
SocialSecurityNumber	Social Security Number
CAMPAIGN_SCHEDULE_NAME	Campaign Schedule Name
CompetencyType	Competency Type
Competency	Competency
occtCreationTime	Time of the Day
MarketingPIN	Marketing PIN
ServiceKey	Service Key
ContractNumModifier	Contract Number Modifier

Example 9–12 XML for ScreenPop Event with CustomerID

```
<CCTSDK>
    <event name="ScreenPop">
        <data name="occtSourceID" value="134"/>
        <data name="CustomerID" value="1234567"/>
        <data name="occtANI" value="40666546"/>
        <data name="occtDNIS" value="7404"/>
    </event>
</CCTSDK>
```

See Also

Oracle Advanced Inbound Implementation Guide

Oracle Customer Care User Guide

Oracle TeleSales User Guide

9.5 Deploying Basic Integration

This section includes the following topics:

- Section 9.5.1, "Media Action Configuration"

- Section 9.5.2, "User Profile Configuration"

9.5.1 Media Action Configuration

Media Action must be defined in the Oracle Universal Work Queue Media Action HTML administration page for the proper Oracle eBusiness Suite Application to pop on the ScreenPop event. The following is an example to pop Customer Care Form on any classification and Oracle Universal Work Queue Diagnostics Form on Diagnostics classification.

Table 9–13 Media Type Classifications

Media Type	Classification	Media Action
Basic Telephony		Customer Care Media Function
Basic Telephony	Diagnostics	Universal Work Queue Media Diagnostics Function

9.5.2 User Profile Configuration

Note: Basic Telephony is typically implemented with inbound telephony as the only media. In these cases, Basic Telephony is enabled and all other media queues are disabled.

User profile options (site-level or user-level) must be set as shown in the following table.

Table 9–14 User Profile Options

Profile	Value
IEU: Queue: Basic Telephony	Yes
IEU: Queue: Inbound Telephony	No
IEU: Queue: Advanced Outbound	No
IEU: Queue: Inbound E-mail	No
IEU: Queue: Acquired E-mail	No
IEU: Queue: Web Collaboration	No
IEU: Queue: Web Callback	No

Additional profile options at the user-level must be set as shown in the following table.

Table 9–15 Additional Profile Options

Profile	Meaning
CCT: Basic Telephony: ACD ID	<agent acd id>
CCT: Basic Telephony: ACD Password	<agent acd password>
CCT: Basic Telephony: ACD Queue	<agent acd queue>
CCT: Basic Telephony: Listener Port	(Optional) listener port for Basic SDK HTTP Listener. If not specified, a random port will be allocated.
CCT: Basic Telephony: Reconnect Interval	(Optional) Number of seconds for the reconnect interval
CCT: Basic Telephony: Third Party URL	(Required) Third-party URL implementing Callout handling
CCT: Basic Telephony: Log Level	(Optional) Error, warning, info or verbose
CCT: Basic Telephony: IC Plugin: Enable Log Window	(Optional) Yes/no, enable Log tab on Basic Panel
CCT: Basic Telephony: IC Plugin: Enable Event Window	(Optional) Yes/no, enable Event tab on Basic Panel
CCT: Basic Telephony: IC Plugin: Enable ScreenPop Window	(Optional) Yes/no, enable ScreenPop tab on Basic Panel

9.6 Testing Basic Implementation

Basic Implementation can be tested in one of two ways:

- Section 9.6.1, "Testing with SDK Integrated Test Utility"
- Section 9.6.2, "Testing with Oracle eBusiness Suite Application"

9.6.1 Testing with SDK Integrated Test Utility

The Basic SDK Implementation can be tested using the SDK Integrated Test Utility. This is useful in the development stage of the Basic SDK Implementation when access to Oracle eBusiness Suite is not available. The SDK Integrated Test Utility

emulates the Basic SDK client plug-in and enables developers to invoke callouts to their Basic SDK implementations and receive HTTP/XML events.

1. Start the SDK Integrated Test Utility
2. Run sdkrun.bat.
3. In the TeleDevice Test Utility tab, select the Basic Teleset tab.
4. Select a row and right click.

   A menu appears.
5. Select **Assign**.

   A dialog box appears.
6. Enter the URL, agent ID, teleset ID and listener port.
7. Click **OK**.
8. The selected row is filled with agentID and telesetID information.
9. Right click the selected row.
10. Choose **Connect**.

    An internal window (the Basic Teleset Watch window) appears.

The Basic Teleset Watch Window consists of the following components:

- Pull Down menu Event and Method
- Current Event Display

Developers can use the pull down menu Method to invoke HTTP callouts to their own implementation. HTTP/XML events received by the Basic Teleset Watch Window can be viewed in the Current Event Display area. Developers can also use the Event pull down menu to display the list of events received by the Basic Teleset.

9.6.2 Testing with Oracle eBusiness Suite Application

Use the configuration described in Section 9.5, "Deploying Basic Integration" to create a user with the appropriate Oracle eBusiness Suite responsibility. After you launch the Universal Work Queue form, perform the inbound call test listed in the following table.

Table 9–16 Test Cases

#	Procedure	Expected Outcome	Result	Comments
A1	Launch Universal Work Queue form	Extension Number Dialog Appears		
A2	Enter Extension Number	Universal Work Queue Client should be launched without error message. Basic SDK Integration should receive Register and Login callouts.		
A3	Double click Basic Telephony <ANY> record	Basic SDK Integration should receive Ready callout		
A4	Generate an inbound call to the agent logged on	Basic SDK Integration should send ScreenPop event. A screen pop should occur.		
A5	Agent answer the call	Basic SDK Integration should send CallEstablished event.		
A6	Agent release the call	Basic SDK Integration should send CallReleased event		
A7	Press End Interaction	Basic SDK Integration should receive Ready callout		
A8	Press Stop Media button on icWork Controller	Basic SDK Integration should receive NotReady callout		
A9	Close from Universal Work Queue	Oracle Universal Work Queue form should be successfully closed, Basic SDK Integration should receive Logout and Unregister callouts		

9.7 Sample Code

The Basic Integration sample code consists of four Java source files in the directory oracle/apps/cct/java/sample/basic.

- BasicSDKException.java
- BasicSDKManager.java
- BasicSDKBootstrap.java

- SampleBasicSDKManager.java

In the following example implementation of the Basic SDK integration with the Switch Simulator, the following functions are implemented.

- Callouts: Register, Unregister, MakeCall
- Events: Screenpop, CallEstablished, CallReleased

The following table describes the location of the Basic Integration sample code and the functionality supported by the corresponding sample implementation.

Table 9–17 Sample Code Location and Functionality

Location	Sample Code	Supported Functions
oracle/apps/cct/java/sample/basic	BasicSDKBootsrap.java, BasicSDKException.java, BasicSDKManager.java, SampleBasicSDKManager.java	Basic SDK Integration supports all functions.

9.8 Switch Simulator

The Switch Simulator is one of the Interaction Center server processes that is used in demonstration environments where a real switch is not available. The simulator supports teleset and route point functions. Access the Switch Simulator programmatically by using the OpenTel Bean API (a Java API, package oracle.apps.cct.openTel.bean). The simulator models each line of the teleset and route point as extension objects, which you can access by using the OpenTelExtensionBean. The Switch Simulator sends OpenTel events.

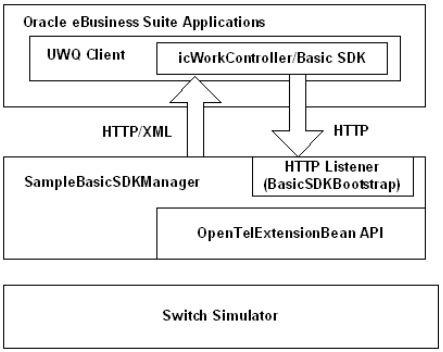
See Also

Oracle Telephony Adapter SDK Javadoc for oracle.apps.cct.openTel.bean package.

9.9 Design

In the following example implementation, Basic SDK Integration acts as a bridge between the Basic SDK client plug-in residing in Oracle eBusiness Suite and the Switch Simulator. SampleBasicSDKManager handles HTTP callouts, which are translated to OpenTel Bean API. OpenTel events sent by the Switch Simulator are received and translated into XML, and then sent to the Basic SDK client plug-in using HTTP.

Figure 9–3 Basic Integration Example Scenario



The following descriptions explain how each function is implemented.

Register Callout

When SampleBasicSDKManager receives a Register request, it creates an OpenTelExtensionBean object that connects to the specified extension running in Switch Simulator. An event listener is added to the OpenTelExtensionBean object so that OpenTel events can be received.

Unregister Callout

When SampleBasicSDKManager receives an Unregister request, it disposes of the OpenTelExtensionBean object and removes any event listener from it.

MakeCall Callout

When SampleBasicSDKManager receives a MakeCall request, it looks up the specified OpenTelExtensionBean object and invokes the makeCall API.

ScreenPop Event

ScreenPop event is sent when an OpenTel event of INBOUND_CALL and state RECEIVE is received.

CallEstablished Event

CallEstablised event is sent when an OpenTel event of TP_ANSWERED and state CONNECT is received. The primary call ID of the current call is retained.

CallReleased Event

CallReleased event is sent when an OpenTel event of state NULL is received and there is a current call.

See Also

Source code and Oracle Telephony Adapter SDK Javadoc

10

Oracle Advanced Outbound Extension

The Advanced Integration of Oracle Telephony Adapter SDK supports preview and progressive calls for Oracle Advanced Outbound. The Advanced Outbound Extension of Oracle Telephony Adapter SDK supports predictive calls in the Oracle Telephony Adapter Server framework.

Predictive calls are dialed automatically by a predictive dialer, such as a voice detection unit (VDU), and transferred programmatically to an interaction center agent. Oracle Advanced Outbound requires access to analog extensions and VDU boards. The Oracle Telephony Adapter SDK Advanced Outbound Extension extends the Oracle Telephony Adapter Server framework to support development of telephony adapters that are capable of accessing these two types of devices.

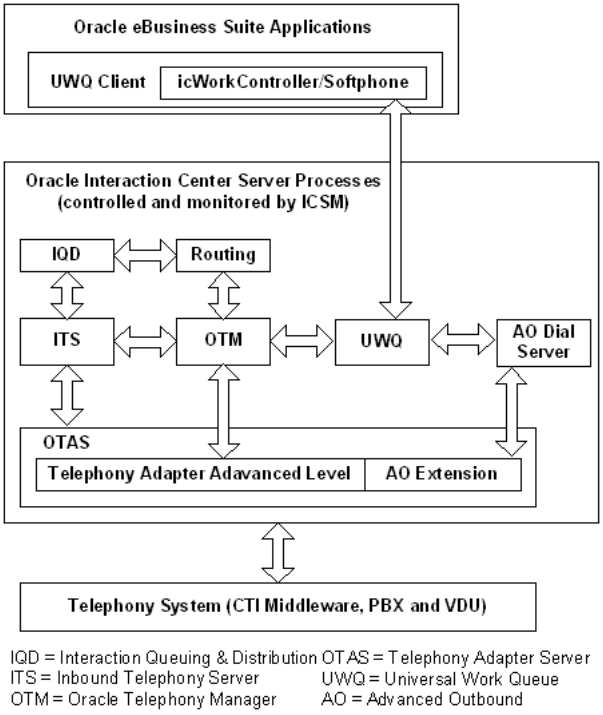
This chapter contains the following topics:

10.1 Advanced Outbound Extension Architecture

The architecture of the Oracle Telephony Adapter SDK Advanced Outbound Extension consists of all the components that are required by the Advanced Integration level of Oracle Advanced Inbound plus the Advanced Outbound Dial Server. Advanced Outbound Dial Server is a server process that integrates list management and a predictive dialer. Oracle Telephony Adapter Server provides the Advanced Outbound Dial Server with the abstraction that is necessary to access and control predictively-dialed calls.

As the following diagram illustrates, in Advanced Integration, the Telephony Adapter Server is the intermediary between the telephony system and Oracle Advanced Inbound. Oracle Universal Work Queue is the intermediary between Oracle Advanced Inbound and the agent's client system, including the icWork Controller and the softphone. The Oracle Telephony Adapter Server is extended to handle requests from Advanced Outbound Dial Server.

Figure 10–1 Advanced Outbound Extension Architecture



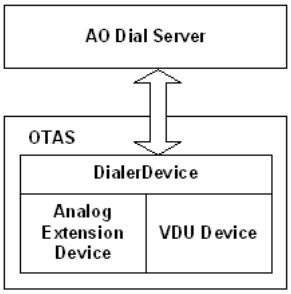
The Advanced Outbound Extension consists of two major devices:

- VDU: Predictive dialer hardware
- Analog Extension: Extension for analog lines. The analog extension is similar to the TelesetDevice but has only one line extension.

Oracle Telephony Adapter Server provides abstraction interfaces for these types of devices so that adapter developers can develop specific implementations of these

devices to their target switch, middleware and VDU hardware. The Advanced Outbound Dial Server accesses these devices by using the dialer object hosted by Oracle Telephony Adapter Server. The following diagram illustrates the architecture.

Figure 10–2 Advanced Outbound Extension and Oracle Telephony Adapter Server



As the previous diagram illustrates, Advanced Outbound Dial Server communicates with AnalogExtensionDevice and VDUDevice through DialerDevice running in Oracle Telephony Adapter Server.

DialerDevice controls and receives events from AnalogExtensionDevice and VduDevice to perform the functions required by the Advanced Outbound Dial Server. DialerDevice is developed Oracle and is shipped with Oracle Advanced Inbound. Adapter developers are not required to develop DialerDevice.

The Advanced Outbound Extension adds the following to the Oracle Telephony Adapter SDK package.

- Java interface: Defined in package oracle.apps.cct.sdk.ao
- C header files: Located in directory oracle/apps/cct/c/ao

10.2 Implementing Advanced Outbound Extension

This section contains the following topics:

10.2.1 Programming Languages

Developers can implement Advanced Outbound Extension in the Java or C programming languages. The choice of programming language is determined by the underlying API provided by the third-party vendor of AnalogExtension and the VDU board. Typically, the AnalogExtension API is included in the same API that is necessary for Advanced Integration (that is, TelesetDevice and RoutePointDevice) while the VDU API is available separately.

Developers may implement Advanced Outbound Extension in the same programming language as the Advanced Integration or in a different programming language. For example, developers can implement Advanced Integration in Java and implement the Advanced Outbound Extension in C. The following table lists the supported combinations of programming languages and implementation types.

Table 10–1 Supported Programming Languages and Implementation Types

Implementation Type	Description	Coding	Packaging
All Java	Implement all interfaces in Java	TeleDeviceFactoryImpl implements TeleDeviceFactory, AnalogExtensionDevice Factory, VduDeviceFactory	One JAR file
All C	Implement all interfaces in C	Implement all C functions	Two DLL files: one for Advanced Integration, one for Advanced Outbound Extension
Advanced Integration in Java, Advanced Outbound Extension completely in C	Implement TelesetDevice and RoutePointDevice in Java Implement AnalogExtensionDe vice and VduDevice in C	TeleDeviceFactoryImpl implements TeleDeviceFactory	One JAR file and one DLL file

Table 10–1 Supported Programming Languages and Implementation Types (Cont.)

Implementation Type	Description	Coding	Packaging
VDU in C only	Implement TelesetDevice RoutePointDevice and AnalogExtensionDevice in Java Implement VduDevice in C	TeleDeviceFactoryImpl implements TeleDeviceFactory, AnalogExtensionDeviceFactory	One JAR file and one DLL file

Advanced Integration is always packaged separately from Advanced Outbound Extension except for the all-Java implementations.

10.2.2 Implementing AnalogExtensionDevice

AnalogExtensionDevice consists of three interfaces:

- AnalogExtensionDeviceFactory: Factory interface that creates and destroys AnalogExtensionDevice objects
- AnalogExtensionDevice: Interface for all methods of Analog Device
- AnalogExtensionEventListener: Interface for sending events required by Analog Device

10.2.2.1 AnalogExtensionDeviceFactory

Developers must implement the AnalogExtensionDeviceFactory interface in their TeleDeviceFactory implementations. The TeleDeviceFactory implementation must implement both the TeleDeviceFactory and AnalogExtensionDeviceFactory interfaces. For example,

```
public class TeleDeviceFactoryImpl implements TeleDeviceFactory,
AnalogExtensionDeviceFactory
```

init

Definition: Initialize AnalogExtensionDeviceFactory.

Table 10–2 init Parameters

Parameter Name	Type Java or C	Value
config	Hashtable / OHashtable	Configuration parameters in a Hashtable

Returns: Not applicable

createAnalogExtensionDevice

Definition: Create an AnalogExtension object.

Table 10–3 createAnalogExtensionDevice Parameters

Parameter Name	Type Java or C	Value
extension	String/const char*	Number of the analog extension

Returns: AnalogExtensionDevice object

destroyAnalogExtensionDevice

Definition: Destroy an AnalogExtension object.

Parameter Name: extension

Type Java or C: String/const char*

Value: Number of the analog extension

Returns: AnalogExtensionDevice object

10.2.2.2 AnalogExtensionDevice

AnalogExtensionDevice is the interface that represents an extension for analog lines. AnalogExtensionDevice is similar to the TelesetDevice, but has only one line extension. It is used by the VDU to make outbound calls for Advanced Outbound Integration.

addAnalogExtensionEventListener

Definition: Add AnalogExtensionEventListener

Table 10–4 addAnalogExtensionEventListener Parameters

Parameter Name	Type Java or C	Value
listener	AnalogExtensionEventListener , N/A	The listener to add

Returns: Not applicable

removeAnalogExtensionEventListener

Definition: Remove AnalogExtensionEventListener

Table 10–5 removeAnalogExtensionEventListener Parameters

Parameter Name	Type Java or C	Value
listener	AnalogExtensionEventListener / N/A	The listener to remove

Returns: Not applicable

makeCall

Definition: Make a call.

Table 10–6 makeCall Parameters

Parameter Name	Type: Java or C	Value
destNumber	AnalogExtensionEventListener /NA	The listener to remove,
appData	String/const char*	Application data

Returns: Not applicable

releaseCall

Definition: Release a call.

Parameters: Not applicable

Returns: Not applicable

transferScreened

Definition: Initialize a two-step transfer.

Table 10–7 transferScreened Parameters

Parameter Name	Type Java or C	Value
destNumber	String / const char	Destination number to dial
appData	String / const char	Application data

Returns: Not applicable

transferUnscreened

Definition: Initialize a single-step transfer.

Table 10–8 transferUnscreened Parameters

Parameter Name	Type: Java or C	Value
destNumber	String / const char *	Destination number to dial
appData	String / const char *	Application data

Returns: Not applicable

completeTransfer

Definition: Complete transfer.

Parameters: Not applicable

Returns: Not applicable

destroy

Definition: Destroy the AnalogExtensionDevice

Parameters: Not applicable

Returns: Not applicable

10.2.2.3 AnalogExtensionEventListener

Oracle Telephony Adapter Server internally implements AnalogExtensionEventListener. AnalogExtensionEventListener is associated with AnalogExtensionDevice by calling addAnalogExtensionEventListener and

removeAnalogExtensionEventListener. Developers must keep a list of listeners in the AnalogExtensionDevice implementation and send events to Oracle Telephony Adapter Server by calling the method defined in AnalogExtensionEventListener.

callEstablishedEvent

Definition: Call established event.

Parameters: Not applicable

Returns: Not applicable

callReleasedEvent

Definition: Call released event

Parameters: Not applicable

Returns: Not applicable

callTransferredEvent

Definition: Call transferred event.

Parameters: Not applicable

Returns: Not applicable

10.2.3 Implementing VduDevice

VduDevice is the interface that represents predictive dialer hardware.VduDevice consists of three interfaces:

- VduDeviceFactory: Factory interface that creates and destroys VduDevice objects
- VduDevice: Interface for all methods of VDU Device
- VduEventListener: Interface for sending events required by VDU Device

Developers may implement these interfaces in Java or C. For Java implementations, developers must implement this interface in their TeleDeviceFactory implementations similar to AnalogExtensionDeviceFactory. For C implementations, developers must implement the function that is defined in the VduDevice.h and VduFactory.h header files and create a DLL. In these cases, VduDeviceFactory is not required by TeleDeviceFactory implementations.

Note: You can implement VduDevice in C and implement the rest of the adapter in Java. The reason for this is that VDU is likely to be a different system than the CTI middleware with which the developer is integrating. For example, Java APIs may be available for the CTI middleware but not for the VDU board.

10.2.3.1 VduDeviceFactory

init

Definition: Initialize VduDeviceFactory.

Table 10-1 initParameters

Parameters Name	Type: Java or C	Value
config	Hashtable / OHashtable	Configuration parameters in a Hashtable

Returns: Not applicable

createVduDevice

Returns: Create a VduDevice object.

Table 10–9 createVduDevice Parameters

Parameter Name	Type: Java or C	Value
vduId	String / const char*	Vdu ID
config	Hashtable / OHashtable	Configuration parameters in a Hashtable

Returns: Vdu Device object

destroyVduDevice

Definition: Destroy a VduDevice object and free any allocated resources.

Table 10–10 destroyVduDevice Parameters

Parameter Name	Type: Java or C	Value
config	VduDevice / OcctVduDevice*	Object to be destroyed

Returns: Not applicable

10.2.3.2 VduDevice

addVduDeviceListner

Definition: Add VduEventListener

Table 10–11 addVduDeviceListener

Parameter Name	Type: Java or C	Value
listener	VduEventListener / N/A	Listener to add

Returns: Not applicable

removeVduEventListner

Definition: Remove VduEventListener.

Table 10–12 removeVduEventListner Parameters

Parameter Name	Type: Java or C	Value
listener	VduEventListener / N/A	Listener to remove

Returns: Not applicable

open

Definition: Open the Vdu port.

Table 10–13 open Parameters

Parameter Name	Type: Java or C	Value
portNumber	int / oint	vdu port number
vduType	int / oint	vdu type
voiceBoard	int / oint	voice board number
dtiBoard	int / oint	dti board number

Returns: Not applicable

close

Definition: Close the Vdu port.

Parameters: Not applicable

Returns: Not applicable

offHook

Definition: "offhook" the associated analog extension. After successfully executing this method, the extension remains in the "offhook" state for one or two seconds.

Parameters: Not applicable

Returns: Not applicable

makeCall

Definition: Make a call through this VDU port. This call sets the port offHook before making the call.

Table 10–14 makeCall Parameters

Parameter Name	Type: Java or C	Value
destNumber	String / const char*	Destination number, if null, this call will do call progress detection only
cpd	boolean / oboolean	Call progress detection. If false, all followed parameters are ignored.
sit	boolean / oboolean	STI tone detection
amd	boolean / oboolean	Answering machine detection
amdMethod	int / oint	Answering machine detection method
narc	int / oint	No answer ring count

Returns: Not applicable

dropCall

Definition: Drop the current call on the VDU port, and stop call progress detection.

Parameters: Not applicable

Returns: Not applicable

withdrawCall

Definition: Withdraw the current call on the VDU port. If an outcome had already been reached, then "withdraw" does nothing.

Parameters: Not applicable

Returns: Not applicable

playMessage

Definition: Play voice message.

Table 10–15 playMessage Parameters

Parameter Name	Type: Java or C	Value
fileName	String / const char*	Message file name
repeatCount	int / oint	Repeat count

Returns: Not applicable

setConfiguration

Definition: Update configuration.

Table 10–16 setConfiguration Parameters

Parameter Name	Type: Java or C	Value
config	Hashtable / OHashtable	Configuration parameters in a Hashtable

Returns: Not applicable

10.2.3.3 VduEventListener

VduEventListener is implemented internally by Oracle Telephony Adapter Server. VduEventListener is associated with VduDevice by calling addVduEventListener and removeVduEventListener.

Developers must keep a list of listeners in the VduDevice implementation and send events to Oracle Telephony Adapter Server by calling the method that is defined in VduEventListener.

callOutcomeEvent

Definition: Event reported by Vdu device.

Table 10–17 callOutcomeEvent Parameters

Parameter Name	Type: Java or C	Value
vduState	int / oint	vdu state
callOutcome	int / oint	call outcome

Returns: Not applicable

10.3 Testing Advanced Outbound Extension

The Advanced Outbound Extension has two testing methods:

- SDK Integrated Test Utility
- Oracle eBusiness Suite Advanced Outbound

The TeleDevice component of the SDK Integrated Test Utility provides a user interface for creating and controlling Dialer devices. Developers can use the Dialer device UI for invoking Dialer methods and monitor Dialer Events.

Developers should set up an environment for Oracle Advanced Outbound and perform integration testing.

See Also

Oracle Advanced Outbound Implementation Guide

10.4 Deploying Advanced Outbound Extension

Implementations that are done exclusively in Java require no extra steps to deploy the Advanced Outbound Extension because the Adapter Jar file contains the extension. For partial Java and C implementations, perform the following procedures:

- Package the C implementation in a DLL file, such as ao_extension.dll.
- Copy the file ao_extension.dll to the third-party directory where Interaction Center Server Manager is installed and where Oracle Telephony Adapter Server will run.

- Update the Server Option of the target Oracle Telephony Adapter Server with -vdu_dll ao_extension.dll.

A

SDK Qualification Worksheet

Use the following questions to determine if the integration of Oracle Telephony Adapter SDK at a specific customer site is necessary and, if it is necessary, what the SDK requirements are.

1. What PBX/ACDs are supported?

- Name, version and release (current or legacy? If old, plan to upgrade?)_____
  _____
- Is there a CTI interface?
  - Yes
  - No
- What ACD teleset is used?_____
- How many extensions/line appearances are on each agent teleset?_____
- On which line are inbound calls received?_____
- On which line are outbound calls made? _____

2. What, if any, CTI middlewares are supported?

- Name, version and release_____
  - Support for server-side integration
  - C API
  - Java API

3. Does the customer already have CTI middleware or a preference for a particular CTI middleware?

- No
- Yes, the ACD/PBX is supported by the following CTI middleware:
  - CT Connect (Java or C API)
  - Cisco ICM, (C API only)
  - Genesys, (C API only)

4. How many agents does the interaction center have?

1000 agents per Oracle Telephony Adapter Server/Call Center Connectors on a single site

5. How many sites does the customer want to CTI enable?

- Oracle Advanced Inbound Oracle Telephony Adapter SDK supports single site routing, queueing and distribution in this release
- If the customer has multiple sites, multi-site routing and call and data transfer, the customer may want to consider a CTI middleware which provides multi-site functionality, such as Cisco ICM or Genesys.

6. (Only if # 5 is yes) If the customer wants to enable multiple sites, they want:

- Enterprise routing (arrive at any site and be distributed to any site)
- Enterprise call and data transfer

7. Is the customer an existing Oracle eBusiness Suite customer?

- Yes, they have the following applications:
  - TeleService
  - TeleSales
  - *i*Support
  - *i*Meeting
  - Collections
  - *i*Store
  - Marketing Online

- No, they are buying the following applications:
  - TeleService
  - TeleSales
  - *i*Support
  - *i*Meeting
  - Collections
  - *i*Store
  - Marketing Online

8. In which Oracle Advanced Inbound features is the customer interested?

- Call routing:
  - PBX skill-based
  - DNIS
  - Routing on business data
  - Rules-based routing
- Screen pops
- Other call data (If IVR, see #9)
- Call and data transfer
- They have or want a softphone GUI

9. Does the interaction center have automated outbound dialing? (outbound telesales campaigns)

- Yes. How do they want to dial?
  - Preview: Record delivered. An agent initiates an outbound call.
  - Progressive: Record delivered to an agent. Dialing is automatic.
  - Predictive: System dials customer. Only live contacts are transferred to the agent.
- No.

10. Does the interaction center have an IVR?

- Yes.
  - The IVR manufacturer is_____
  - CTI-enabled/intelligent (IVR maintains the call ID of the call exiting the IVR)
  - IVR connects (PL/SQL) to an Oracle database
  - IVR updates the call's application data field
- No.

11. What is the $US total license revenue?

- eBusiness suite_____
- Interaction Center_____

12. What is the implementation project time line?_____
_____

13. What is the consulting budget?_____
_____

B

SDK Scope Analysis

Use the following questions to determine the SDK requirements of a customer site.

1. Does Oracle Advanced Inbound Development certify or support this PBX-middleware combination, or is the certification in progress?

- Yes
- No

2. If this PBX-middleware combination is not certified or supported by Oracle Advanced Inbound Development, is there an Oracle consultant or Oracle SSI-provided telephony adapter solution available, or is a solution in progress?

To check if any solutions may be available, refer to the Oracle Solution Services ePortal (http://i3sn011e.idc.oracle.com:7777/servlet/page?_pageid=180,85,89,153,115,123,101,159,111,133,137,149,163&_dad=portal30&_schema=PORTAL30).

- Yes
- No

3. Is a consulting-based pipeline available for this PBX and CTI middleware?

Consider contacting the Oracle professional community (such as the itccon_us mailing list) to ask whether there are opportunities similar to yours.

- Yes
- No:

4. PBX product research and data collection

a. Model:

b. Current release:

c. CTI interface:

d. Route points, how routing works:

e. Is a software CTI translator required for the proprietary PBX CTI interface?

- Yes
- No

f. Does the PBX or CTI middleware permit routing to be controlled by an adjunct platform, specifically, Oracle Advanced Inbound?

- Yes
- No

g. The PBX/ACD is:

- TDM-based
- IP-based

h. Does this PBX provide ACD (automatic call distribution) functionality?

- Yes
- No

i. The platform is:

- CSTA CTI specification (the PBX is a CSTA switch)
- Propriety call model

j. Is a software CTI translator required for the proprietary PBX CTI interface?

- Yes
- No

5. CTI Middleware

a. Is the manufacturer a leading brand of CTI middleware, and does the manufacturer still exist?

- Yes

- No

b. The CTI middleware is:

- Intel CT Connect/NetMerge Call Processing Software
- Cisco ICM
- Genesys
- Other

c. Does this CTI middleware support the customer's PBX platform?

- Yes
- No

d. List available documentation.

-
-
-
-

e. List available tools and simulators.

-
-
-
-
-

f. Are developer licenses for API required?

- Yes
- No

g. Are developer licenses for API available?

- Yes
- No

h. Does the CTI middleware provide routing?

- Yes

- No

i. Does the CTI middleware provide IVR integration?
   - Yes
   - No

j. Does the CTI middleware support server-level integration?
   - Yes, C-based API
   - Yes, Java-based API
   - No

6. Additional Information on Advanced Inbound Requirements

- Log in, log out
- Ready, not ready
- Make call
- Answer a call
- Transfer a call
  - Consultative transfer to route point
  - Blind transfer to agent
  - Blind transfer to route point
- Conference calls
  - Consultative conference
  - Blind transfer to agent
  - Blind transfer to route point
- Hold
- Call and data transfer

7. What are the reporting requirements?

- PBX reports
- CTI middleware reports
- Interaction Center Intelligence reports

8. Is Advanced Outbound Predictive required?

- No
- Yes:

a. Does the PBX support a lineside T1 interface to the VDUs?

- Yes
- No

b. Does the CTI middleware support monitoring and control of the lineside analog PBX extensions?

- Yes
- No

C

Telephony Adapter Test Cases

Events, denoted by [<Agent>/<EventName>] are Oracle Telephony Manager events to check at the given time.

> **Note:** Not all possible Oracle Telephony Manager events are checked at all times.

This Appendix includes the following test cases:

- Table C–1, "Agent State Test Procedures"
- Table C–2, "Make Call / Answer Call Test Procedures"
- Table C–3, "Hold/ Retrieve Test Procedures"
- Table C–4, "Consultative Transfer Test Procedures"
- Table C–5, "Blind Transfer Test Procedures"
- Table C–6, "Consultative Conference Test Procedures"
- Table C–7, "Consultation (Transfer/Conference) /Cancel Test Procedures"
- Table C–8, "Route Point Test Procedures"
- Table C–9, "DTMF Tones Test Procedures"

Table C–1 Agent State Test Procedures

Number	Test Procedure
A1	With agent A's actual phone logged out, log in agent A with the softphone.
A2	With agent A's actual phone logged in and not ready, log in agent A with the softphone; see if the phone is automatically logged out and then logged in.

Table C–1 Agent State Test Procedures

Number	Test Procedure
A3	With agent A's actual phone logged in and ready, login agent A with the softphone; see if the phone is automatically logged out and then logged in.
A4	With agent A's actual phone logged in and on a call, log in agent A with the softphone; see if the proper error message is displayed; manually hang up any remaining calls on the phone; log in agent A with the softphone again; see if the phone is automatically logged out and then logged in.
A5	Log out agent A.
A6	Set agent A to ready state.
A7	Set agent A to not-ready state.

Table C–2 Make Call / Answer Call Test Procedures

Number	Test Procedure
B1	A calls B; B does not answer; A hangs up.
B2	A calls B; B answers [NO B/ExternalWithData]; A hangs up.
B3	A calls B; B answers; B hangs up.
B4	A makes outbound call to X; X does not answer; A hangs up.
B5	A makes outbound call to X; X answers [NO A/ExternalWithData]; A hangs up.
B6	A makes outbound call to X; X answers; X hangs up.
B7	A receives inbound call from X [A/ExternalWithData]; A does not answer; X hangs up.
B8	A receives inbound call from X [A/ExternalWithData]; A answers; A hangs up.
B9	A receives inbound call from X; A answers; X hangs up.
B10	A calls B which is busy.
B11	A makes outbound call to X which is busy.
B12	A calls an invalid internal number.
B13	A makes outbound call to an invalid external number.

Table C–3 Hold/ Retrieve Test Procedures

Number	Test Procedure
C1	A calls B; B answers; A puts call on hold; A retrieves the call.
C2	A calls B; B answers; B puts call on hold; B retrieves the call.
C3	A calls B; B answers; A puts call on hold; B puts call on hold; A retrieves the call; B retrieves the call.
C4	A calls B; B answers; A puts call on hold; B hangs up.
C5	A calls B; B answers; B puts call on hold; A hangs up.
C6	A makes outbound call to X; X answers; A puts call on hold; A retrieves the call.
C7	A makes outbound call to X; X answers; A puts call on hold; X hangs up.
C8	A receives inbound call from X [A/ExternalWithData]; A answers; A puts call on hold; A retrieves the call.
C9	A receives inbound call from X; A answers; A puts call on hold; X hangs up.
C10	A calls B; A puts call on hold; B answers; A retrieves the call.
C11	A makes outbound call to X; A puts call on hold; X answers; A retrieves the call.

Table C–4 Consultative Transfer Test Procedures

Number	Test Procedure
D1	A receives inbound call from X [A/ExternalWithData]; A answers; A makes consultation call to B; B answers [B/TransferWithData; call data should be same as A/ExternalWithData previously]; A completes transfer.
D2	A receives inbound call from X; A answers; A makes outbound consultation call to Y; Y answers; A completes transfer.
D3	A makes outbound call to X; X answers [NO A/ExternalWithData]; A makes consultation call to B; B answers [NO B/TransferWithData]; A completes transfer.
D4	A makes outbound call to X; X answers; A makes outbound consultation call to Y; Y answers; A completes transfer.
D5	A calls B; B answers; B makes consultation call to C; C answers [NO C/TransferWithData]; B completes transfer.
D6	A calls B; B answers; B makes outbound consultation call to X; X answers; B completes transfer.

Table C–4 Consultative Transfer Test Procedures (Cont.)

Number	Test Procedure
D7	A receives inbound call from X; A answers; A makes consultation call to B; B answers; B puts consultation call on hold; A completes transfer; B retrieves transferred call.
D8	A makes outbound call to X; X answers; A makes consultation call to B; B answers; B puts consultation call on hold; A completes transfer; B retrieves transferred call.
D9	A calls B; B answers; B makes consultation call to C; C answers; C puts consultation call on hold; B completes transfer; C retrieves transferred call.
D10	A calls B; B answers; A puts call on hold; B makes consultation call to C; C answers [NO C/TransferWithData]; B completes transfer; A retrieves transferred call.
D11	A calls B; B answers; A puts call on hold; B makes outbound consultation call to X; X answers; B completes transfer; A retrieves transferred call.

Table C–5 Blind Transfer Test Procedures

Number	Test Procedure
E1	A receives inbound call from X [A/ExternalWithData]; A answers; A makes consultation call to B; A completes transfer before B answers; B answers [B/TransferWithData].
E2	A receives inbound call from X; A answers; A makes outbound consultation call to Y; A completes transfer before Y answers; Y answers.
E3	A makes outbound call to X; X answers [NO A/ExternalWithData]; A makes consultation call to B; A completes transfer before B answers; B answers [NO B/TransferWithData].
E4	A makes outbound call to X; X answers; A makes outbound consultation call to Y; A completes transfer before Y answers; Y answers.
E5	A calls B; B answers; B makes consultation call to C;B completes transfer before C answers; C answers [NOC/TransferWithData].
E6	A calls B; B answers; B makes outbound consultation call to X; B completes transfer before X answers; X answers.
E7	A calls B; B answers; A puts call on hold; B makes consultation call to C; B completes transfer before C answers; C answers [NO C/TransferWithData]; A retrieves transferred call.
E8	A calls B; B answers; A puts call on hold; B makes outbound consultation call to X; B completes transfer before X answers; X answers; A retrieves transferred call.

Table C–6 Consultative Conference Test Procedures

Number	Test Procedure
F1	A receives inbound call from X [A/ExternalWithData]; A answers; A makes consultation call to B; B answers [B/ConferenceWithData; call data should be same as A/ExternalWithData previously]; A completes conference.
F2	A receives inbound call from X; A answers; A makes outbound consultation call to Y; Y answers; A completes conference.
F3	A makes outbound call to X; X answers [NO A/ExternalWithData]; A makes consultation call to B; B answers [NO B/ConferenceWithData]; A completes conference.
F4	A makes outbound call to X; X answers; A makes outbound consultation call to Y; Y answers; A completes conference.
F5	A calls B; B answers; B makes consultation call to C; C answers [NO C/ConferenceWithData]; B completes conference.
F6	A calls B; B answers; B makes outbound consultation call to X; X answers; B completes conference.
F7	A receives inbound call from X; A answers; A makes consultation call to B; B answers; B puts consultation call on hold; A completes conference; B retrieves conferenced call.
F8	A makes outbound call to X; X answers; A makes consultation call to B; B answers; B puts consultation call on hold; A completes conference; B retrieves conferenced call.
F9	A calls B; B answers; B makes consultation call to C; C answers; C puts consultation call on hold; B completes conference; C retrieves conferenced call.
F10	A calls B; B answers; A puts call on hold; B makes consultation call to C; C answers [NO C/ConferenceWithData]; B completes conference; A retrieves conferenced call.
F11	A calls B; B answers; A puts call on hold; B makes outbound consultation call to X; X answers; B completes conference; A retrieves conferenced call.

Table C–7 Consultation (Transfer/Conference) /Cancel Test Procedures

Number	Test Procedure
G1	A receives inbound call from X [A/ExternalWithData]; A answers; A makes consultation call to B; X hangs up before B answers; B answers [B/TransferWithData].
G2	A receives inbound call from X; A answers; A makes consultation call to B; A hangs up consultation call before B answers; A reconnects to X.

Table C–7 Consultation (Transfer/Conference) /Cancel Test Procedures (Cont.)

Number	Test Procedure
G3	A receives inbound call from X [A/ExternalWithData]; A answers; A makes consultation call to B; B answers [B/TransferWithData]; X hangs up before A completes the transfer/conference.
G4	A receives inbound call from X; A answers; A makes consultation call to B; B answers; A hangs up consultation call; A reconnects to X.
G5	A receives inbound call from X; A answers; A makes consultation call to B; B answers; B hangs up consultation call; A reconnects to X.
G6	A receives inbound call from X; A answers; A makes outbound consultation call to Y; X hangs up before Y answers; Y answers.
G7	A receives inbound call from X; A answers; A makes outbound consultation call to Y; A hangs up consultation call before Y answers; A reconnects to X.
G8	A receives inbound call from X; A answers; A makes outbound consultation call to Y; Y answers; X hangs up before A completes the transfer/conference.
G9	A receives inbound call from X; A answers; A makes outbound consultation call to Y; Y answers; A hangs up consultation call; A reconnects to X. P
G10	A receives inbound call from X; A answers; A makes outbound consultation call to Y; Y answers; Y hangs up consultation call; A reconnects to X.
G11	A makes outbound call to X; X answers; A makes consultation call to B; X hangs up before B answers; B answers.
G12	A makes outbound call to X; X answers; A makes consultation call to B; A hangs up consultation call before B answers; A reconnects to X.
G13	A makes outbound call to X; X answers; A makes consultation call to B; B answers; X hangs up before A completes the transfer/conference.
G14	A makes outbound call to X; X answers; A makes consultation call to B; B answers; A hangs up consultation call; A reconnects to X.
G15	A makes outbound call to X; X answers; A makes consultation call to B; B answers; B hangs up consultation call; A reconnects to X.
G16	A makes outbound call to X; X answers; A makes outbound consultation call to Y; X hangs up before Y answers; Y answers.
G17	A makes outbound call to X; X answers; A makes outbound consultation call to Y; A hangs up consultation call before Y answers; A reconnects to X.
G18	A makes outbound call to X; X answers; A makes outbound consultation call to Y; Y answers; X hangs up before A completes the transfer/conference.
G19	A makes outbound call to X; X answers; A makes outbound consultation call to Y; Y answers; A hangs up consultation call; A reconnects to X.

Table C–7 Consultation (Transfer/Conference) /Cancel Test Procedures (Cont.)

Number	Test Procedure
G20	A makes outbound call to X; X answers; A makes outbound consultation call to Y; Y answers; Y hangs up consultation call; A reconnects to X.
G21	A calls B; B answers; B makes consultation call to C; A hangs up before C answers; C answers.
G22	A calls B; B answers; B makes consultation call to C; B hangs up consultation call before C answers; B reconnects to A.
G23	A calls B; B answers; B makes consultation call to C; C answers; A hangs up before B completes the transfer/conference.
G24	A calls B; B answers; B makes consultation call to C; C answers; B hangs up consultation call; B reconnects to A.
G25	A calls B; B answers; B makes consultation call to C; C answers; C hangs up consultation call; B reconnects to A.
G26	A calls B; B answers; B makes outbound consultation call to X; A hangs up before X answers; X answers.
G27	A calls B; B answers; B makes outbound consultation call to X; B hangs up consultation call before X answers; B reconnects to A.
G28	A calls B; B answers; B makes outbound consultation call to X; X answers; A hangs up before B completes the transfer/conference.
G29	A calls B; B answers; B makes outbound consultation call to X; X answers; B hangs up consultation call; B reconnects to A.
G30	A calls B; B answers; B makes outbound consultation call to X; X answers; X hangs up consultation call; B reconnects to A.
G31	A receives inbound call from X; A answers; A makes consultation call to B which is busy; A reconnects to X.
G32	A receives inbound call from X; A answers; A makes outbound consultation call to Y which is busy; A reconnects to X.
G33	A receives inbound call from X; A answers; A makes consultation call to an invalid internal number; A reconnects to X.
G34	A receives inbound call from X; A answers; A makes outbound consultation call to an invalid external number; A reconnects to X.
G35	A makes outbound call to X; X answers; A makes consultation call to B which is busy; A reconnects to X.
G36	A makes outbound call to X; X answers; A makes outbound consultation call to Y which is busy; A reconnects to X.

Table C–7 Consultation (Transfer/Conference) /Cancel Test Procedures (Cont.)

Number	Test Procedure
G37	A makes outbound call to X; X answers; A makes consultation call to an invalid internal number; A reconnects to X.
G38	A makes outbound call to X; X answers; A makes outbound consultation call to an invalid external number; A reconnects to X.
G39	A calls B; B answers; B makes consultation call to C which is busy; B reconnects to A.
G40	A calls B; B answers; B makes outbound consultation call to X which is busy; B reconnects to A.
G41	A calls B; B answers; B makes consultation call to an invalid internal number; B reconnects to A.
G42	A calls B; B answers; B makes outbound consultation call to an invalid external number; B reconnects to A.

Table C–8 Route Point Test Procedures

Number	Test Procedure
K1	A calls R; R routes to B [NO B/ExternalWithData]; B answers; A makes consultation call to C; C answers [NO C/TransferWithData]; A completes transfer.
K2	A calls R; R routes to B; B answers; A makes consultation call to C; A completes transfer before C answers; C answers [NO C/TransferWithData].
K3	A calls R; R routes to B; B answers; A makes consultation call to C; C answers [NO C/ConferenceWithData]; A completes conference.
K4	A calls B; B answers; B makes consultation call to R; R routes to C; C answers [NO C/TransferWithData]; B completes transfer.
K5	A calls B; B answers; B makes consultation call to R; B completes transfer before R routes to C; R routes to C; C answers [NO C/TransferWithData].
K6	A calls B; B answers; B makes consultation call to R; R routes to C; B completes transfer before C answers; C answers [NO C/TransferWithData].
K7	A calls B; B answers; A makes consultation call to R; R routes to C; C answers [NO C/TransferWithData]; A completes transfer.
K8	A calls B; B answers; A makes consultation call to R; A completes transfer before R routes to C; R routes to C; C answers [NO C/TransferWithData].
K9	A calls B; B answers; A makes consultation call to R; R routes to C; A completes transfer before C answers; C answers [NO C/TransferWithData].

Table C–8 Route Point Test Procedures (Cont.)

Number	Test Procedure
K10	A calls B; B answers; B makes consultation call to R; R routes to C; C answers [NO C/ConferenceWithData]; B completes conference.
K11	A calls R; R routes to B; B answers; A makes consultation call to C; B hangs up before C answers; C answers [NO C/ConferenceWithData].
K12	A calls R; R routes to B; B answers; A makes consultation call to C; A hangs up consultation call before C answers.
K13	A calls R; R routes to B; B answers; A makes consultation call to C; C answers; B hangs up.
K14	A calls R; R routes to B; B answers; A makes consultation call to C; C answers; A hangs up consultation call.
K15	A calls R; R routes to B; B answers; A makes consultation call to C; C answers; C hangs up consultation call.
K16	A calls B; B answers; B makes consultation call to R; A hangs up before R routes to C; R routes to C; C answers.
K17	A calls B; B answers; B makes consultation call to R; R routes to C; A hangs up before C answers; C answers.
K18	A calls B; B answers; B makes consultation call to R; B hangs up consultation call before R routes to C.
K19	A calls B; B answers; B makes consultation call to R; R routes to C; B hangs up consultation call before C answers.
K20	A calls B; B answers; B makes consultation call to R; R routes to C; C answers; A hangs up.
K21	A calls B; B answers; B makes consultation call to R; R routes to C; C answers; B hangs up consultation call.
K22	A calls B; B answers; B makes consultation call to R; R routes to C; C answers; C hangs up consultation call.
K23	A receives inbound call from X [A/ExternalWithData]; A answers; A makes consultation call to R; R routes to B; B answers [B/TransferWithData]; A completes transfer.
K24	A receives inbound call from X; A answers; A makes consultation call to R; A completes transfer before R routes to B; R routes to B; B answers [B/TransferWithData].
K25	A receives inbound call from X; A answers; A makes consultation call to R; R routes to B; A completes transfer before B answers; B answers [B/TransferWithData].

Table C–8 Route Point Test Procedures (Cont.)

Number	Test Procedure
K26	A receives inbound call from X; A answers; A makes consultation call to R; R routes to B; B answers [B/ConferenceWithData]; A completes conference.
K27	A makes an outbound call to X; X answers; A makes consultation call to R; R routes to B; B answers [B/TransferWithData]; A completes transfer.
K28	A makes an outbound call to X; X answers; A makes consultation call to R; A completes transfer before R routes to B; R routes to B; B answers [B/TransferWithData].
K29	A makes an outbound call to X; X answers; A makes consultation call to R; R routes to B; A completes transfer before B answers; B answers [B/TransferWithData].
K30	A makes an outbound call to X; X answers; A makes consultation call to R; R routes to B; B answers [B/ConferenceWithData]; A completes conference.

Table C–9 DTMF Tones Test Procedures

Number	Test Procedure
L1	A received inbound call from X; A answers; A presses some digits and clicks on "Dial" button; X hears DTMF tones of digits dialed by A.
L2	A makes outbound call to X; X answers; A presses some digits and clicks on "Dial" button; X hears DTMF tones of digits dialed by A.
L3	A calls B; B answers; A presses some digits and clicks on "Dial" button; B hears DTMF tones of digits dialed by A.
L4	A calls B; B answers; B presses some digits and clicks on "Dial" button; A hears DTMF tones of digits dialed by B.

D

Sample Code

This appendix contains the following topics:

D.1 Sample Codes for Integration Levels and Extension

The following tables describe the locations of the sample codes for Standard Integration, Advanced Integration and Advanced Outbound Extension, and the functions supported by the corresponding sample implementations.

Table D–1 Standard Integration Sample Code

Location	Sample Code	Supported Functions
oracle/apps/cct/java/sample/standard	SampleTeleDeviceFactory.java, SampleTelesetDevice.java	Supports all functions that are required by Standard SDK Integration.

Table D–2 Advanced Integration Sample Code

Location	Sample Code	Supported Functions
oracle/apps/cct/java/sample/advanced	SampleTeleDeviceFactory.java, SampleTelesetDevice.java, SampleRoutePointDevice.java	agent login / logout, agent ready / not ready, make call, consultation call, complete transfer, complete conference, hold call, retrieve call, route call, abandon call

Table D–3 Advanced Outbound Extension Sample Code

Location	Sample Code	Supported Functions
oracle/apps/cct/java/sample/ao	SampleAOTeleDeviceFactory.java, SampleAnalogExtensionDevice.java, SampleVduDevice.java	The sample AnalogExtensionDevice and VduDevice implementation log the API calls.

D.2 Switch Simulator Overview

The switch simulator is one of the Interaction Center server processes that is used in demonstration environments where a real switch is not available. The switch simulator supports teleset and route point functions, and can be accessed programmatically using the OpenTel Bean API (a Java API, package oracle.apps.cct.openTel.bean). The switch simulator models each line of the teleset and route point as extension objects. Extension objects can be accessed using the OpenTelExtensionBean. Events are sent by the switch simulator as OpenTel events.

See Also

Source code and oracle.apps.cct.openTel.bean package in Oracle Telephony Adapter SDK Javadoc

D.3 Sample Adapter Design for Standard and Advanced Integration

The Standard Integration and Advanced Integration sample adapter connects to the switch simulator and forwards Adapter SDK method calls to the OpenTel Bean API method calls and also process OpenTel events. The sample adapter then translates the OpenTel events into Adapter events.

The following example demonstrates the relationship among the Oracle Interaction Center servers.

Interaction Center <---> Oracle Telephony Adapter Server <--> SampleAdapter <---> Switch Simulator.

SampleTeleDeviceFactory

SampleTeleDeviceFactory reads the IP address and port from the Hashtable in its init() method.

See Also

Source code and Oracle Telephony Adapter SDK Javadoc

SampleTelesetDevice

SampleTelesetDevice maintains one OpenTelExtensionBean object for each line. Adapter SDK methods are directed to the appropriate OpenTelExtensionBean object based on the line index specified. SampleTelesetDevice registers a dedicated OpenTelEventListener for each OpenTelExtensionBean object to process switch simulator events.

See Also

Source code and Oracle Telephony Adapter SDK Javadoc

SampleRoutePointDevice

SampleRoutePointDevice maintains one OpenTelExtensionBean object. Adapter SDK methods are directed to the route point OpenTelExtensionBean object. SampleRoutePointDevice registers itself as OpenTelEventListener to process switch simulator events.

See Also

Source code and Oracle Telephony Adapter SDK Javadoc

D.4 Sample Adapter Design for Basic Integration

For Basic Integration sample adapter design, see Section 9.7, "Sample Code".

E

Diagnostics and Troubleshooting

This document describes how to troubleshoot Oracle Advanced Inbound implementations using Oracle Telephony Adapter SDK.

E.1 Upgrading from Release 11.5.6 or 11.5.7

Telephony SDK is part of Oracle Application Release 11.5.8 (Interaction Center Family Pack O).

When upgrading from Release 11.5.6 or 11.5.7 to Release 11.5.8 SDK, review the following documents:

- Metalink Note: 204766.1: Oracle Advanced Inbound Feature Support Changes for Oracle Interaction Center Family Pack-O
- MetaLink Note 204767.1: Oracle Advanced Inbound Certified Switches/Middleware for Oracle Interaction Center Family Pack-O
- Interaction Center Family Pack O 2374818 Readme
- Upgrade Patch 2435611 Readme: 115.6 to 11.5.8 SDK UPGRADE

E.2 SDK Integrated Test Utility

Table E–1 SDK Integrated Test Utility

Common Problems	Recommended Action
SDK Test Utility does not start.	Check JDK path and SDK_JRE_HOME and SDK_JRE. environment variable in sdkenv.cmd. Make sure no spaces are in the PATH or CLASSPATH environment variable.
Oracle Telephony Adapter Server does not start.	Refer to Oracle Telephony Adapter Server Startup Sequence for more details.
Verification Tool does not start.	Check Verification Tool configuration. Make sure Oracle Telephony Adapter Server is running before Verification Tool.

E.2.1 Verification Tool

The Verification Tool is a development tool to automate Oracle Telephony Adapter SDK API testing. The Verification Tool's main use is to run SDK adapter test cases and to generate outputs easily. The scripts and solution files are written for a limited set of switches and middlewares. The supplied solution files do not cover all possible valid outcomes. Therefore, an adapter that passes all verifications might not necessarily work properly when integrated with Oracle Interaction Center. Likewise, an adapter that fails to pass some verifications might not necessarily work when integrated with Oracle Interaction Center.

Note: The Verification Tool is not a certification tool. Do not use the Verification Tool as the final check for Adapter implementation or for live customer diagnostics. Adapter developers should always run and test their implementations fully integrated with Interaction Center before going into production. Support should always use Oracle Telephony Adapter Server logging as the diagnostic tool.

E.3 Oracle Telephony Adapter Server Logging

Oracle Telephony Adapter Server logs runtime and diagnostics information into a log file. You can configure the content and the level of details logged in the file. This section describes how to configure and access Oracle Telephony Adapter Server logs.

E.3.1 Accessing Oracle Telephony Adapter Server Logs

Every time Oracle Telephony Adapter Server starts, it creates a log file in the directory <ICSM_TOP>/admin/scripts/<SERVER_NAME>, in which ICSM_TOP is the path where Interaction Center Server Manager is installed, and SERVER_NAME is the server name of Oracle Telephony Adapter Server that is defined in the Interaction Center Server Manager HTML Administration.

The Oracle Telephony Adapter Server log file has the naming convention

```
<OTAS_server_name>mmddyyyy_hhMMss.log
```

where " mmddyyyy" is the date when the log file is created, and "hhMMss" is the time when the log file is created.

Use the following procedure to view and download Oracle Telephony Adapter Server log files by way of the Interaction Center Server Manager HTML Administration.

Log In

HTML Login URL

Responsibility

Interaction Center Server Manager

Prerequisites

Create a user in Interaction Center Server Manager HTML Administration.

Steps

1. Log in to the Interaction Center Server Manager HTML Admin.
2. Locate the Server Group / Node where the target Oracle Telephony Adapter Server is running.
3. Select **OTAS**.
4. Go to Server Details > Advanced.

   The Advanced page opens.
5. View and download any of the listed log files.

Note: You can view log files when Interaction Center Server Manager is running on the target node. When running Oracle Telephony Adapter Server by using the SDK Integrated Test Utility, the Oracle Telephony Adapter Server log file is in the directory that is specified in the SDK test utility.

You can also access Oracle Telephony Adapter Server log files locally on the node where Oracle Telephony Adapter Server runs.

E.3.2 Configuring Oracle Telephony Adapter Server Logs

Use the following procedure to configure Oracle Telephony Adapter Server to log different levels of detail.

Log In

HTML Login URL

Responsibility

Interaction Center Server Manager

Prerequisites

Create a user in Interaction Center Server Manager HTML Administration.

Steps

1. Locate the Server Group / Node where the target Oracle Telephony Adapter Server is running.
2. Select **ICSM**.
3. Go to Server Details > Server Parameter page.
4. Edit the Trace Level Parameter.

Oracle Telephony Adapter Server classifies different log entries into modules and levels of details. You can configure Oracle Telephony Adapter Server to log all modules with a specific level of details or log only specific modules with a specific level of details. Each log entry has the following format:

[timestamp] - [module: level] - message

For example,

```
    [Fri Jul 19 12:18:05 PDT 2002] - [adapter:verbose] -
    SampleTeleDeviceFactory.createTelesetDevice(7001, 8001, 9001);
```

A message could span multiple lines, as in the following example.

```
[Fri Jul 19 12:18:06 PDT 2002] - [device:info] - [API]
device = TelesetDevice[7001,8001,9001]
object = oracle.apps.cct.sdk.sample.SampleTelesetDevice@1ece00
method = loginAgent
        acdAgentId = 7001
        acdAgentPassword = 8001
        acdQueue = 9001
```

E.3.2.1 Log Detail Levels

The following table lists the level of details that are supported by Oracle Telephony Adapter Server, in the order of descending severity.

Table E–2 Log Levels

Level	Meaning
fatal	Fatal and unrecoverable error, Oracle Telephony Adapter Server will shutdown
error	Major error, affects some Oracle Telephony Adapter Server functions
warning	Minor error, does not affect functionality
info	Informational only, for diagnostics purpose
verbose	Extended diagnostics information, use only when instructed by development

The following table lists the log modules that are defined in Oracle Telephony Adapter Server.

Table E–3 Log Modules

Module	Meaning
otas	Generic Oracle Telephony Adapter Server messages
comm	Communication layer messages
db	Database access layer messages
adapter	Messages generated by Telephony Adapter

Table E–3 Log Modules (Cont.)

Module	Meaning
device	Telephony API and Events diagnostics messages

E.3.2.2 Trace Level Format

By default, all modules are set to error level, so that logging includes only messages that are tagged at the error level or a higher level (error + fatal). Users can change the level by using the Trace Level server parameter.

The Trace Level server parameter uses the following format. Users can globally set the level for all modules or set the level individually.

- <level> or
- <module_1>=<level_1>,<module_2>=<level_2>,...

The following table lists and describes trace levels.

Table E–4 Trace Levels

Trace Level	Description
	(Default) Log error and fatal level messages for all modules
fatal	Log only fatal level messages for all modules
error	Log error and fatal level messages for all modules
warning	Log warning, error and fatal level messages for all modules
info	Log informational, warning, error and fatal level messages for all modules
device=info	■ Log info level messages for device ■ Log error level messages for all others
device=info,adapter=verbose,otas=warning	■ Log info level messages for device ■ Log verbose level messages for adapter ■ Log warning level messages for Oracle Telephony Adapter Server ■ Log error level messages for all others

E.3.2.3 Common and Recommended Usage

- info: Turn on everything from fatal to informational messages for all modules.

- device=info: For tracing Adapter API and Events while filtering out informational messages from other modules.
- device=info,adapter=verbose: For tracing Adapter API and Events and any internal messages generated by Adapter.

E.3.2.4 General Guidelines in Interpreting Oracle Telephony Adapter Server Logs

- Identify fatal and error messages.
- Info and verbose messages are not serious, even for an exception trace.

E.3.2.5 Oracle Telephony Adapter Server Log Header

The Oracle Telephony Adapter Server Log Header contains information about the release and build number and startup sequence information. This information is very useful in determining the release, patch level and adapter configuration at a particular site. The Oracle Telephony Adapter Server Log Header contains the following.

> **Note:** The following example shows the default logging configuration. If the log level is set to a higher error level, then the log displays more entries than those shown here.

- Server Wrapper Build and Release Information indicates the version of Interaction Center Server Manager.

```
[Tue Jul 23 14:37:15 PDT 2002] -
************************************
Server Wrapper - Release 11.5.8
Build 219 on Mon Jul 15 15:54:52 PDT 2002
(c) Copyright 2002 Oracle Corporation. All rights reserved.
************************************
```

- Oracle Telephony Adapter Server Build and Release Information indicates the version of Oracle Telephony Adapter Server.

```
[Tue Jul 23 14:37:17 PDT 2002] -
************************************
Oracle Telephony Adapter Server - Release 11.5.8
Build 2547 on Thu Jul 18 18:36:26 PDT 2002
(c) Copyright 2002 Oracle Corporation. All rights reserved.
************************************
```

Startup Sequence Information shows users which steps Oracle Telephony Adapter Server has gone through in startup. If there is an error in a particular step, then an error is logged and Oracle Telephony Adapter Server will not start. Adapter information is logged in this step:

```
[Tue Jul 23 14:37:18 PDT 2002] - Database Layer started
[Tue Jul 23 14:37:18 PDT 2002] - Socket Layer started
[Tue Jul 23 14:37:18 PDT 2002] - Adapter Layer started
```

The final status indicates that Oracle Telephony Adapter Server has been started successfully:

```
[Tue Jul 23 14:37:18 PDT 2002] - Oracle Telephony Adapter Server [xxx_otas]
started.
```

See Also

Section E.4, "Oracle Telephony Adapter Server Startup Sequence"

E.4 Oracle Telephony Adapter Server Startup Sequence

Oracle Telephony Adapter Server starts in the following sequence:

1. Database Layer: Establish connection to database and load configurations
2. Socket Layer: Create server TCP/IP socket for communication
3. Adapter Layer: Load and Initialize Adapter

If any part of the phase fails, then a fatal error is logged and Oracle Telephony Adapter Server shuts down.

E.4.1 Database Layer

The database layer phase establishes the database connection. Oracle Telephony Adapter Server executes this phase in the following order:

1. Load dbc file (passed by Interaction Center Server Manager).
2. Create database connection.
3. Load Server Configurations.
4. Load Middleware Configurations.
5. Update Server Status.

If any of the above procedures fail, Oracle Telephony Adapter Server logs the following:

```
[Tue Jul 23 15:33:04 PDT 2002] - Database Layer starting FAILED
[Tue Jul 23 15:33:04 PDT 2002] - [otas:fatal] - Nested Exception :
< exception stack trace based on different errors >
```

The following table lists possible reasons for failure and describes recommended actions.

Table E–5 Database Layer Recommended Action

Reason	Recommended Action
dbc file does not exist (should not happen in general if Interaction Center Server Manager is running)	Check DBC file integrity
dbc file error (should not usually happen when Interaction Center Server Manager is running)	Check DBC file integrity
database unavailable	Contact Database Administrator
server configuration error	Check Server Configuration in Interaction Center Server Manager HTML Admin
middleware configuration error (missing Middleware Configuration Name in Server Parameter)	Check Server Configuration in Interaction Center Server Manager HTML Admin and Middleware Configuration in Call Center HTML Admin

If the database layer phase succeeds, Oracle Telephony Adapter Server logs the following message:

```
[Tue Jul 23 15:33:04 PDT 2002] - Database Layer started
```

E.4.2 Socket Layer

The socket layer phase creates a TCP/IP server socket. Oracle Telephony Adapter Server executes this phase in the following order.

1. Get the local IP address.
2. Create a server socket.
3. Update database with the port information.

If any of the step above fails, then Oracle Telephony Adapter Server will log the following message:

```
[Tue Jul 23 15:33:04 PDT 2002] - Socket Layer starting FAILED
[Tue Jul 23 15:33:04 PDT 2002] - [otas:fatal] - Nested Exception :
< exception stack trace based on different errors >
```

The following table lists possible reasons for failure and describes recommended actions.

Table E–6 Socket Layer Recommended Action

Reason	Recommended Action
Network error	Contact Network Administrator
Unable to create server socket (address in use)	Check if -svr_port is used, contact network administrator if necessary
Unable to update database records	Contact Database Administrator

If the socket layer phase succeeds, Oracle Telephony Adapter Server logs the following message:

```
[Tue Jul 23 15:33:04 PDT 2002] - Socket Layer started
```

E.4.3 Adapter Layer

In the adapter layer phase, Oracle Telephony Adapter Server loads and initializes the Adapter in the following order:

1. Get Middleware Type and PBX Type from Middleware Configuration.
2. Derive the TeleDeviceFactory Java class name to be loaded.
3. Load the TeleDeviceFactory Java class.
4. Initialize the Adapter by invoking init() method.

If any of the above procedures fails, then Oracle Telephony Adapter Server logs the following message:

```
[Tue Jul 23 15:33:04 PDT 2002] - Adapter Layer starting FAILED
[Tue Jul 23 15:33:04 PDT 2002] - [otas:fatal] - Nested Exception :
< exception stack trace based on different errors >
```

The following table lists possible reasons for failure and describes recommended actions.

Table E–7 Adapter Layer Recommended Action

Reason	Recommended Action
Wrong Middleware Configuration	Check Middleware Configuration
Unable to derive TeleDeviceFactory Java class name from Middleware Configuration	Check Middleware Configuration
Unable to load the TeleDeviceFactory Java class (class does not exist in clasps)	Check if adapter jar file exists in 3rdParty directory
Unable to instantiate the TeleDeviceFactory Java class (no default constructor)	Fix adapter source code
Unable to instantiate the TeleDeviceFactory Java class (constructor not public)	Fix adapter source code
TeleDeviceFactory Java class throws TeleDeviceException on init() (Runtime error checked by Adapter)	Check Middleware Configuration, Check Adapter Log, Consult Adapter Developer
Unable to load 3rd Party Java classes or libraries (NoClassDefFoundError)	Check if the 3rd Party libraries is placed in the 3rdParty directory

If the adapter layer phase succeeds, Oracle Telephony Adapter Server logs the following message:

```
[Tue Jul 23 15:33:04 PDT 2002] - Adapter Layer started
```

In the Adapter Phase, Oracle Telephony Adapter Server logs additional information about the type of adapter that is being configured, as shown in the following example.

```
[Tue Jul 23 15:21:00 PDT 2002] -
Middleware type =  ICC_CUSTOM_ADAPTER
PBX Type =  null
TeleDeviceFactory Class =
oracle.apps.cct.sdk.sample.SampleTeleDeviceFactory

[Tue Jul 23 15:21:00 PDT 2002] -

Integration Level = <Advanced or Standard>
```

Based on the Middleware type and PBX type, users can determine whether the Oracle Telephony Adapter Server is running a custom adapter or an adapter that Oracle developed.

The following table lists and describes possible middleware types.

Table E–8 Possible Middleware Types

Middleware Type in Log File	Description
ICC_CUSTOM_ADAPTER	Custom Java Adapter
ICC_CUSTOM_C_ADAPTER	Custom C Adapter
ICC_CTC_ADAPTER (In-house)	Adapter for CT Connect
ICC_ASPECT_ADAPTER (In-house)	Adapter for Aspect
ICC_ICM_ADAPTER (In-house)	Adapter for Cisco ICM

The following table lists and describes possible PBX types (PBX types are applicable only to in-house adapters).

Table E–9 Possible PBX Types

PBX Type in Log File	Description
A	LUCENT_DEFINITY
C	ALCATEL_4400
E	ERICSSON_MD110
M	NORTEL_MERIDIAN
P	ASPECT
R	ROCKWELL_SPECTRUM
S	SIEMENS_HICOM
X	NORTEL_DMS100
Z	CALLMANAGER

Note: A value defined here does not indicate that switch certification is available for the platform. Please consult the Switch Certification support matrix for details.

E.4.4 Successful Startup

If Oracle Telephony Adapter Server starts successfully, then it logs the following message:

```
[Tue Jul 23 14:37:18 PDT 2002] - Oracle Telephony Adapter Server [ <otas_
server_name> ] started.
```

E.5 Tracing SDK APIs and Events

Oracle Telephony Adapter Server logs all API invoked, Events sent and Exceptions thrown by the Adapter at the info level of the device module. To enable this feature, set Trace Level to one of the following:

- info
- device=info

Oracle recommends that you use device=info so that log entries of other modules will not show.

SDK API and Event traces are useful for diagnosing Adapter behavior. When they are used in conjunction with the call event flow specification of the Telephony SDK, users will be able to determine if the Adapter is working properly.

E.5.1 SDK API Traces

SDK API Traces are Adapter methods, such as makeCall, that Interaction Center Servers invoke. Oracle Telephony Adapter Server logs the device, the method that is invoked and the parameters that are passed to that method.

SDK API Traces have the following format:

```
[<timestamp>] - [device:info] - [API]
    device =  TelesetDevice[<extension1,extension2,extension3>] or
    RoutePointDevice[<extension>]
    object =  <teledevice object>
    method = <method name>
        <parameter1 = value1 >
        <parameter2 = value2 >
        ...
```

Example Log Entry

```
[Wed Jul 24 16:45:48 PDT 2002] - [device:info] - [API]
     device =  TelesetDevice[7001,8001,9001]
```

```
object =  oracle.apps.cct.sdk.sample.SampleTelesetDevice@1ed043
method = blindTransfer
   mediaItemId =  7001
   ineIndex =  8001
   destinationNumber =  9001
```

E.5.2 SDK Exception Traces

SDK Exception Traces are TeleDeviceExceptions that the Adapter throws. The log records both the device that throws the exception and the exception details.

SDK Exception Traces have the following format:

```
[<timestamp>] - [device:info] - [EXCEPTION]
    device =  TelesetDevice[<extension1,extension2,extension3>] or
    RoutePointDevice[<extension>]
    exception :  <exception details>
```

Example Log Entry

```
[Wed Jul 24 16:45:48 PDT 2002] - [device:info] - [EXCEPTION]
    device =  TelesetDevice[7001,8001,9001]
    exception :  TeleDeviceException:ERROR_CODE_FUNCTION_NOT_
    SUPPORTED;blindTransfer NOT IMPLEMENTED
```

E.5.3 SDK Event Traces

SDK Event Traces are events that the Adapter sends. The log records both the device that sends the event and the event details.

SDK Event Traces have the following format:

```
[<timestamp>] - [device:info] - [EVENT]
    device =  TelesetDevice[<extension1,extension2,extension3>] or
    RoutePointDevice[<extension>]
    object =  <teledevice object>
    event = <event name>
        <parameter1 = value1 >
        <parameter2 = value2 >
        ...
```

Example Log Entry

```
[Wed Jul 24 17:41:39 PDT 2002] - [device:info] - [EVENT]
    device =  TelesetDevice[7002,8002,9002]
    object =  oracle.apps.cct.sdk.sample.SampleTelesetDevice@1eca40
    event = beginCallEvent
```

```
        mediaItemId =  7002
        mediaItemIdComplete =  true
        ineIndex =  1
        ani =  7001
        dnis =  8002
        data =  {MI=7002}
        dataComplete =  true
```

E.5.4 Sample Traces

DEFINITION OF SAMPLE TRACES

Creation of TelesetDevice

```
[Wed Jul 24 17:41:24 PDT 2002] - [device:info] - [API]
    createTelesetDevice
        extension1 =  7001
        extension2 =  8001
        extension3 =  9001
```

Destruction of TelesetDevice

```
[Wed Jul 24 17:45:08 PDT 2002] - [device:info] - [API]
     destroyTeleDevice
     device =  T:7001
```

Simple Internal Call Scenario

The following table lists the complete SDK API and Event traces for a simple internal call scenario in which A calls B, B answers, and A hangs up.

Table E–10 Simple Internal Call SDK API and Trace Events

Sequence	Notes	TelesetDevice A: TelesetDevice[7001,8001,9001]	TelesetDevice B: TelesetDevice[7002,8002,9002]
1	A calls B on 8002 using line 0	[Wed Jul 24 17:41:39 PDT 2002] - [device:info] - [API] device = TelesetDevice[7001,8001,9001] object = oracle.apps.cct.sdk.sample.SampleTelesetDevice@1ec457 method = makeCall mediaItemId = 7002 lineIndex = 0 destinationNumber = 8002	
2	B sends beginCallEvent on line 1		[Wed Jul 24 17:41:39 PDT 2002] - [device:info] - [EVENT] device = TelesetDevice[7002,8002,9002] object = oracle.apps.cct.sdk.sample.SampleTelesetDevice@1eca40 event = beginCallEvent mediaItemId = 7002 mediaItemIdComplete = true lineIndex = 1 ani = 7001 dnis = 8002 data = {MI=7002} dataComplete = true
3*	B sends callRingingEvent on line 1		[Wed Jul 24 17:41:39 PDT 2002] - [device:info] - [EVENT] device = TelesetDevice[7002,8002,9002] object = oracle.apps.cct.sdk.sample.SampleTelesetDevice@1eca40 event = callRingingEvent lineIndex = 1

Table E–10 Simple Internal Call SDK API and Trace Events (Cont.)

Sequence	Notes	TelesetDevice A: TelesetDevice[7001,8001,9001]	TelesetDevice B: TelesetDevice[7002,8002,9002]
4	A sends beginCallEvent on line 0	[Wed Jul 24 17:41:39 PDT 2002] - [device:info] - [EVENT] device = TelesetDevice[7001,8001,9001] object = oracle.apps.cct.sdk.sample.SampleTelesetDevice@1ec457 event = beginCallEvent mediaItemId = 7002 mediaItemIdComplete = true lineIndex = 0 ani = 7001 dnis = 8002 data = {MI=7002} dataComplete = true	
5*	A sends callDialingEvent on line 0	[Wed Jul 24 17:41:39 PDT 2002] - [device:info] - [EVENT] device = TelesetDevice[7001,8001,9001] object = oracle.apps.cct.sdk.sample.SampleTelesetDevice@1ec457 event = callDialingEvent lineIndex = 0	
6	B answers call on line 1		[Wed Jul 24 17:41:56 PDT 2002] - [device:info] - [API] device = TelesetDevice[7002,8002,9002] object = oracle.apps.cct.sdk.sample.SampleTelesetDevice@1eca40 method = answerCall lineIndex = 1

Table E–10 Simple Internal Call SDK API and Trace Events (Cont.)

Sequence	Notes	TelesetDevice A: TelesetDevice[7001,8001,9001]	TelesetDevice B: TelesetDevice[7002,8002,9002]
7*	B sends callEstablishedEvent on line 1		[Wed Jul 24 17:41:56 PDT 2002] - [device:info] - [EVENT] device = TelesetDevice[7002,8002,9002] object = oracle.apps.cct.sdk.sample.SampleTelesetDevice@1eca40 event = callEstablishedEvent lineIndex = 1
8*	A sends callEstablishedEvent on line 0	[Wed Jul 24 17:41:56 PDT 2002] - [device:info] - [EVENT] device = TelesetDevice[7001,8001,9001] object = oracle.apps.cct.sdk.sample.SampleTelesetDevice@1ec457 event = callEstablishedEvent lineIndex = 0	

Table E–10 Simple Internal Call SDK API and Trace Events (Cont.)

Sequence	Notes	TelesetDevice A: TelesetDevice[7001,8001,9001]	TelesetDevice B: TelesetDevice[7002,8002,9002]
9	A releases call on line 0	[Wed Jul 24 17:42:03 PDT 2002] - [device:info] - [API] device = TelesetDevice[7001,8001,9001] object = oracle.apps.cct.sdk.sample.SampleTelesetDevice@1ec457 method = releaseCall lineIndex = 0	
10*	A sends callReleasedEvent on line 0	[Wed Jul 24 17:42:03 PDT 2002] - [device:info] - [EVENT] device = TelesetDevice[7001,8001,9001] object = oracle.apps.cct.sdk.sample.SampleTelesetDevice@1ec457 event = callReleasedEvent lineIndex = 0	
11*	B sends callReleasedEvent on line 1		[Wed Jul 24 17:42:03 PDT 2002] - [device:info] - [EVENT] device = TelesetDevice[7002,8002,9002] object = oracle.apps.cct.sdk.sample.SampleTelesetDevice@1eca40 event = callReleasedEvent lineIndex = 1

*The following sequences can be of any order: 3 and 5, 7 and 8, 10 and 11.

See Also

Section 5.4, "Call Scenarios"

E.6 Common Errors

The following table lists common adapter errors that adapter implementers sometimes make, and the symptoms of those errors.

Table E–11 Common Adapter Errors

Adapter Error	Observed Symptom
SDK method not implemented	Softphone functions do not work.
SDK call events not properly implemented	Softphone out of synchronization.
RoutePoint Events not properly implemented	■ Queue Count does not work in Enhanced Passive mode. ■ Missing screen pop data. ■ Reporting shows more than one media items for a single call. ■ Call data transfer does not work.
Media Item ID not properly propagated	
Memory not properly managed in C/C++ implementation	Oracle Telephony Adapter Server crashes.

Glossary

active mode

A routing mode in which Oracle Advanced Inbound controls the routing and distribution of incoming calls to call center agents using business data and rules that are configured in Oracle Advanced Inbound. Specific PBX/ACD configurations are required to grant Oracle Advanced Inbound full control of an inbound call when it reaches a PBX/ACD route point monitored by Oracle Advanced Inbound.

adapter

A telephony driver of the Oracle Telephony Adapter Server developed specifically to integrate Oracle Interaction Center to a specific switch and CTI middleware platform. Oracle develops adapters for certified switch and middleware combinations. Third-parties can use the Oracle Telephony Adapter SDK to develop adapters for switch and middleware combinations that are not certified by Oracle. Typically, each adapter is developed to integrate only with the telephony system of a specific manufacturer.

ACD

Automatic Call Distribution, systems designed to automatically answer, queue and route incoming calls to interaction center agents. An ACD differs from a PBX in that while a PBX allows users to share a limited number of telephone lines, an ACD has at least one telephone line for each agent.

ANI

Automatic Number Identification, a service, similar to caller ID, that long distance carriers provides to identify the calling party's billing number.

API

Application Programming Interface, the calling conventions by which a software application accesses the operating system and other services.

blind transfer

A call transferred from one person to another without the first person telling the other the identity of the caller.

canonical phone number

A standardized phone number of the format:

+<country code> (<area code>) <local exchange>-<subscriber number>

For example, +1 (123) 456-7890.

Given a canonical phone number and an interaction center's local telephone network characteristics, such as local country and area codes, Oracle Advanced Inbound can automatically determine how to call a number.

CTI

Computer Telephony Integration, a system in which a computer is connected to a telephone switch, either PBX or ACD, so that the computer sends instructions to the switch about how to direct telephone calls.

DN

Directory Number, typically the phone number associated with a telephone line.

DNIS

Dialed Number Identification Service, a feature of 800 and 900 lines that identifies the called number to a telephony system, which routes the call to the correct extension.

DTMF

Dual Tone Multi-Frequency, also known as touch-tone dialing.

dynamic route

A route that is based on a PL/SQL query.

enhanced passive mode

A routing mode in which standard PBX/ACD routing and distribution of calls to call center agents occurs with Oracle Advanced Inbound monitoring PBX/ACD

route points to allow classification of calls for targeted screen pops, inbound call queue counts and tracking of calls that are abandoned at the route point for reporting by Oracle Interaction Center Intelligence. Specific PBX/ACD configurations are required to ensure that inbound calls pass through a PBX/ACD route point that is monitored by Oracle Advanced Inbound.

IDE

Interactive Development Environment, a system for supporting the process of writing software. An IDE may include a syntax-directed editor, graphical tools for program entry, and integrated support for compiling and running the program and relating compilation errors back to the source. Examples of IDEs are Visual C^{++} and Visual Basic.

Interaction Center Server Manager

The only server process that is required to be explicitly started on each target machine, ICSM is responsible for starting, stopping and monitoring all the other Oracle Advanced Inbound server processes. The ICSM server processes are controlled by the Interaction Center Server HTML Administration.

Inbound Telephony Server

The Oracle Interaction Center server that handles inbound telephony interactions. ITS supports the following features:

- (Active mode only) ITS enables enterprise data-based routing by listening for route queries offered by the CTI middleware and responding to them to instruct the switch where to route the call.
- ITS monitors calls arriving at route point(s)
- ITS detects calls that are abandoned at route point(s)
- ITS receives IVR data packets from the IVR

interaction center server

Any interaction center server, such as Oracle Interaction Queuing and Distribution, Oracle Universal Work Queue, Oracle Routing server and Oracle Inbound Telephony Server. Same as mid-tier server process and server process.

IVR

Interactive Voice Response, an automated system that, in response to incoming telephone calls, plays a recorded message that gives callers the option of pressing telephone buttons to route the call to one or more extensions.

Javadoc

A facility provided within the Java Development Kit that produces HTML documentation from a program. Javadoc reads the source code and parses specially formatted and positioned comments into documentation.

JDBC

Java Database Connectivity, part of the Java Development Kit that defines an application programming interface for Java for standard SQL access from Java programs to databases.

Java Development Kit (JDK)

A Sun Microsystems product that provides the required environment for Java programming.

Java Native Interface (JNI)

A native programming interface for Java that allows Java code that is running inside a Java Virtual Machine to operate with applications and libraries written in other programming languages such as C and C++.

Java Virtual Machine (JVM)

A specification for software which interprets Java programs that have been compiled into byte-codes and usually stored in a ".class" file. JVMs support object-oriented programming directly by including instructions for object method invocation. JVMs are written in C and can be ported to run on most platforms.

media controller

Software that bridges other systems or software with the underlying media hardware, such as a PBX.

media queue

The interaction center component for queuing and distributing inbound media items. It stores inbound items such as telephone calls or e-mails in a queue and integrates with the routing module so that the items can be sent to a set of agents. The media queue provides an API to other modules, such as Oracle Universal Work Queue, for querying and manipulating items in the queue.

media item

A representation of a telephone call, e-mail, Web callback or other type of media.

mid-tier server process

Any interaction center server, such as Oracle Interaction Queuing and Distribution, Oracle Universal Work Queue, Oracle Routing Server, Oracle Inbound Telephony Server, and Oracle Telephony Media Controller. Same as server process and interaction center server.

monitoring

The ability to view server status.

multi-site

Interaction centers that work together across multiple physical locations.

multi-site routing

The ability to route a call to agents who are located across multiple sites.

multi-site queuing and distribution

A single system storing and maintaining agent queues across multiple sites.

multi-PBX

Support for multiple switches and middleware configurations by the same server.

Observer API

The API that the adapter uses to notify Oracle Interaction Center about incoming calls and to deliver call and caller data that are used for softphone displays, screen pops and call routing. Oracle Interaction Center implements the Observer API.

Oracle Advanced Inbound

The Oracle eBusiness application that is required to telephony enable business applications in the Oracle eBusiness suite. The server architecture of Oracle Advanced Inbound is scalable to run interaction centers with a single physical site or multiple sites. The Oracle Advanced Inbound bundle consists of the following products: Oracle Universal Work Queue, Oracle Telephony Manager, Oracle Telephony Adapter Server and Oracle Interaction Blending.

Oracle Advanced Outbound

The Oracle eBusiness application that provides the outbound telephony capability corresponding to Oracle Advanced Inbound.

Oracle Interaction Center

A group of server processes that serves as the telephony-enabling foundation of Oracle's eBusiness Suite applications.

Oracle Telephony Adapter Server

The CTI adapter server that substitutes for Oracle Call Center Connectors. Oracle Telephony Adapter Server encompasses one telephony adapter per switch.'

Oracle Telephony Manager

The Oracle Interaction Center application that performs queuing, routing and distribution of media items.

package

Groups of procedures, functions, variables and SQL statements grouped together into a single unit.

passive mode

A routing mode in which standard PBX/ACD routing and distribution of calls to call center agents occurs. Oracle Advanced Inbound becomes aware of the call through CTI when the call rings at the agent's teleset. Oracle Advanced Inbound does not monitor or control any PBX/ACD route points in this mode.

PBX

Private Branch eXchange, a telephone system within a company or other organization that switches calls between the company's users and allows them to share a number of outside telephone lines. In passive mode calls are routed by the PBX routes calls.

Provider API

The API that supports telephony and interaction center feature requests such as make call, transfer call, route call and change agent mode. The Provider API Provider is implemented by the adapter.

route point

The first point from which calls are queued and routed. An automatic call distributor (ACD). Route points are also called "pilots" by Alcatel.

scalability

A measure of how well a software or hardware product is able to adapt to future business needs.

screen pop

A user interface presentation of customer data and product and service information that appears on an interaction center agent's monitor simultaneously with the customer's incoming telephone call.

server process

Any interaction center server, such as Oracle Interaction Queuing and Distribution, Oracle Universal Work Queue, Routing server, Oracle Inbound Telephony Server, and Oracle Telephony Media Controller. Same as mid-tier server process and interaction center server.

server status

Information on whether the server process is running or not, how long the server has been running, and so on.

site

A single geographic location where an interaction center is located. A site typically has a PBX and CTI middleware installed.

skill-based routing

A dynamic call routing intelligence that delivers inbound calls to an agent who is appropriately skilled to meet the needs of the caller.

softphone

A functional GUI representation of a telephone that is displayed on interaction agents' monitors.

Software Development Kit

(SDK) Software that is provided by software vendors to allow their products to be used with the products of other software vendors.

static route

A route that is based on cached data.

super group

The topmost, parent server group in a hierarchy of server groups.

switch simulator

A process that uses Intel CT Connect middleware to simulate a Nortel switch and the connection and message behavior of the Oracle Call Center Connectors server. The switch simulator makes it possible to set up an interaction center without connecting to a real switch.

target machine

The machine where mid-tier server processes are run. Same as node.

telephony enabled

The ability of an application to communicate with a telephone system for inbound and outbound calls, or inbound or outbound calls, through the CTI middleware that handles the messaging between a telephone switch and the user's application.

telephony model

A scenario that describes the expected behavior of a call for any given telephony function. For example, in one telephony model, a transferred call has the same call ID as the original call. In another telephony model, a transferred call has the same call ID as the consultation call. In a third telephony model, a transferred call has a completely new call ID that differs from the original call and the consultation call.

telephony system

Any hardware and software components that provide telephony and CTI messaging, such as PBX, ACD, IVR, predictive dialer and CTI middleware.

VDU

Voice detection unit.

Index

Numerics

A

B

C

D

L

M

N

O

P

Q

R

S

T

U

V

W

X