ORACLE
PeopleSoft

PeopleTools 8.62: MultiChannel Framework

December 2025

ORACLE

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://docs.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://docs.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit https://docs.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Contents

Preface

Understanding the PeopleSoft Online Help and PeopleBooks

The PeopleSoft Online Help is a website that enables you to view all help content for PeopleSoft applications and PeopleTools. The help provides standard navigation and full-text searching, as well as context-sensitive online help for PeopleSoft users.

Hosted PeopleSoft Online Help

You can access the hosted PeopleSoft Online Help on the Oracle Help Center. The hosted PeopleSoft Online Help is updated on a regular schedule, ensuring that you have access to the most current documentation. This reduces the need to view separate documentation posts for application maintenance on My Oracle Support. The hosted PeopleSoft Online Help is available in English only.

To configure the context-sensitive help for your PeopleSoft applications to use the Oracle Help Center, see Configuring Context-Sensitive Help Using the Hosted Online Help Website.

Locally Installed PeopleSoft Online Help

If you're setting up an on-premises PeopleSoft environment, and your organization has firewall restrictions that prevent you from using the hosted PeopleSoft Online Help, you can install the online help locally. Installable PeopleSoft Online Help is made available with selected PeopleSoft Update Images and with PeopleTools releases for on-premises installations, through the Oracle Software Delivery Cloud.

Your installation documentation includes a chapter with instructions for how to install the online help for your business environment, and the documentation zip file may contain a README.txt file with additional installation instructions. See *PeopleSoft 9.2 Application Installation* for your database platform, "Installing PeopleSoft Online Help."

To configure the context-sensitive help for your PeopleSoft applications to use a locally installed online help website, see Configuring Context-Sensitive Help Using a Locally Installed Online Help Website.

Downloadable PeopleBook PDF Files

You can access downloadable PDF versions of the help content in the traditional PeopleBook format on the Oracle Help Center. The content in the PeopleBook PDFs is the same as the content in the PeopleSoft Online Help, but it has a different structure and it does not include the interactive navigation features that are available in the online help.

Common Help Documentation

Common help documentation contains information that applies to multiple applications. The two main types of common help are:

- Application Fundamentals

- Using PeopleSoft Applications

Most product families provide a set of application fundamentals help topics that discuss essential information about the setup and design of your system. This information applies to many or all applications in the PeopleSoft product family. Whether you are implementing a single application, some combination of applications within the product family, or the entire product family, you should be familiar with the contents of the appropriate application fundamentals help. They provide the starting points for fundamental implementation tasks.

In addition, the *PeopleTools: Applications User's Guide* introduces you to the various elements of the PeopleSoft Pure Internet Architecture. It also explains how to use the navigational hierarchy, components, and pages to perform basic functions as you navigate through the system. While your application or implementation may differ, the topics in this user's guide provide general information about using PeopleSoft applications.

Field and Control Definitions

PeopleSoft documentation includes definitions for most fields and controls that appear on application pages. These definitions describe how to use a field or control, where populated values come from, the effects of selecting certain values, and so on. If a field or control is not defined, then it either requires no additional explanation or is documented in a common elements section earlier in the documentation. For example, the Date field rarely requires additional explanation and may not be defined in the documentation for some pages.

Typographical Conventions

The following table describes the typographical conventions that are used in the online help.

Typographical Convention	***Description***
Key+Key	Indicates a key combination action. For example, a plus sign (+) between keys means that you must hold down the first key while you press the second key. For **Alt+W,** hold down the **Alt** key while you press the **W** key.
. . . (ellipses)	Indicate that the preceding item or series can be repeated any number of times in PeopleCode syntax.
{ } (curly braces)	Indicate a choice between two options in PeopleCode syntax. Options are separated by a pipe (|).
[ ] (square brackets)	Indicate optional items in PeopleCode syntax.
& (ampersand)	When placed before a parameter in PeopleCode syntax, an ampersand indicates that the parameter is an already instantiated object. Ampersands also precede all PeopleCode variables.

Typographical Convention	*Description*
⇒	This continuation character has been inserted at the end of a line of code that has been wrapped at the page margin. The code should be viewed or entered as a single, continuous line of code without the continuation character.

ISO Country and Currency Codes

PeopleSoft Online Help topics use International Organization for Standardization (ISO) country and currency codes to identify country-specific information and monetary amounts.

ISO country codes may appear as country identifiers, and ISO currency codes may appear as currency identifiers in your PeopleSoft documentation. Reference to an ISO country code in your documentation does not imply that your application includes every ISO country code. The following example is a country-specific heading: "(FRA) Hiring an Employee."

The PeopleSoft Currency Code table (CURRENCY_CD_TBL) contains sample currency code data. The Currency Code table is based on ISO Standard 4217, "Codes for the representation of currencies," and also relies on ISO country codes in the Country table (COUNTRY_TBL). The navigation to the pages where you maintain currency code and country information depends on which PeopleSoft applications you are using. To access the pages for maintaining the Currency Code and Country tables, consult the online help for your applications for more information.

Region and Industry Identifiers

Information that applies only to a specific region or industry is preceded by a standard identifier in parentheses. This identifier typically appears at the beginning of a section heading, but it may also appear at the beginning of a note or other text.

Example of a region-specific heading: "(Latin America) Setting Up Depreciation"

Region Identifiers

Regions are identified by the region name. The following region identifiers may appear in the PeopleSoft Online Help:

- Asia Pacific
- Europe
- Latin America
- North America

Industry Identifiers

Industries are identified by the industry name or by an abbreviation for that industry. The following industry identifiers may appear in the PeopleSoft Online Help:

- USF (U.S. Federal)

- E&G (Education and Government)

Translations and Embedded Help

PeopleSoft 9.2 software applications include translated embedded help. With the 9.2 release, PeopleSoft aligns with the other Oracle applications by focusing our translation efforts on embedded help. We are not planning to translate our traditional online help and PeopleBooks documentation. Instead we offer very direct translated help at crucial spots within our application through our embedded help widgets. Additionally, we have a one-to-one mapping of application and help translations, meaning that the software and embedded help translation footprint is identical—something we were never able to accomplish in the past.

Using and Managing the PeopleSoft Online Help

Select About This Help in the left navigation panel on any page in the PeopleSoft Online Help to see information on the following topics:

- Using the PeopleSoft Online Help.
- Managing hosted Online Help.
- Managing locally installed PeopleSoft Online Help.

PeopleTools Related Links

PeopleTools 8.62 Home Page

PeopleSoft Search and Insights Home Page

“PeopleTools Product/Feature PeopleBook Index” (Getting Started with PeopleTools)

PeopleSoft Online Help

PeopleSoft Information Portal

PeopleSoft Spotlight Series

PeopleSoft Training and Certification | Oracle University

My Oracle Support

Oracle Help Center

Contact Us

Send your suggestions to psoft-infodev_us@oracle.com.

Please include the applications update image or PeopleTools release that you’re using.

Follow Us

Icon	*Link*
	Watch PeopleSoft on YouTube
	Follow @PeopleSoft_Info on X.
	Read PeopleSoft Blogs
	Connect with PeopleSoft on LinkedIn

Chapter 1

Getting Started with PeopleSoft MultiChannel Framework

PeopleSoft MultiChannel Framework Overview

PeopleSoft MultiChannel Framework (MCF) provides the tools that are required to support multiple channels of communication between customers (users) and agents. Some PeopleSoft applications, such as an email response management system (ERMS) from PeopleSoft CRM, use PeopleSoft MultiChannel Framework or you can develop your own applications on the framework that is provided.

PeopleSoft MultiChannel Framework delivers an integrated infrastructure to support multiple interaction channels for call center agents and other PeopleSoft users who must respond to incoming requests and notifications on these channels. PeopleSoft MultiChannel Framework supports email, web-based chat, voice, and generic event channels. It can be used from any PeopleSoft application.

PeopleSoft MultiChannel Framework also supports a broadcast function, which allows a user, such as a supervisor, to broadcast a notification message to a group of agents.

PeopleSoft MultiChannel Framework enables third-party routing systems that enable applications, such as PeopleSoft CRM, to provide embedded multichannel functionality. You may choose either the PeopleSoft queue server or the third-party routing server to route voice, email, chat, and generic events.

PeopleSoft MultiChannel Framework Implementation

This section describes the required steps for implementing PeopleSoft MCF.

Depending on your business use of PeopleSoft MCF, several activities are necessary for implementation:

- At a minimum, you must configure a real-time event notification (REN) server.
- If you are using a PeopleSoft-supplied application, such as ERMS, you must also configure MCF servers, clusters, queues, and agents.
- You can develop your own applications built on the PeopleSoft MCF.
- If you are using PeopleSoft CTI or other third-party MCF integrations, additional configuration is required.

Configuring REN Servers and Clusters

The REN server routes event notifications through the PeopleSoft MultiChannel Framework. Certain PeopleSoft applications, such as Reporting and Optimization, use the REN server to route notifications without using any of the rest of the PeopleSoft MultiChannel Framework. Therefore, a minimal

configuration of PeopleSoft MultiChannel Framework includes at least one REN server and REN server cluster.

REN server-specific security setup is required, including configuration of Real-time Event Notification Permissions for each role using a REN server alone or as part of PeopleSoft MultiChannel Framework.

To provide a secure channel of communication between the clients and the REN servers, the REN servers may be SSL-enabled. SSL-enabled REN servers enable secure communication by providing client and server authentication.

Typically, a system administrator configures and manages REN servers and clusters.

See Understanding REN Servers.

Configuring MCF Servers, Clusters, Queues, and Tasks

MCF servers (queue servers and log servers) work with REN servers to queue and route task notifications to agents. An implementation of PeopleSoft MultiChannel Framework requires configuration of MCF servers, MCF clusters, queues, and tasks.

Typically, a system administrator configures and manages MCF servers and clusters, and either a system administrator or an agent supervisor configures and manages queues and tasks.

See Understanding PeopleSoft MCF Server and Cluster Architecture.

Creating and Defining Agents

Agents are PeopleSoft users who are further defined as PeopleSoft MultiChannel Framework agents. Agents manage tasks that are assigned to them. Agents can log on to and accept tasks from the MultiChannel Console.

Typically, an agent supervisor defines agents and assigns agents to appropriate queues.

See Defining Agents.

Using MCF Sample Pages

PeopleSoft MultiChannel Framework is delivered with sample pages that demonstrate the use of email, web-based chat, and generic channels. An application developer can use these sample page definitions as the basis for application pages. PeopleCode API Reference includes detailed descriptions of PeopleSoft's Mail Classes, MCFIMInfo Classes, and the Universal Queue Classes.

Additional sample pages demonstrate how to use the JavaScript MultiChannel Application Programming Interface (JSMCAPI) to customize the CTI console and to develop supervisor desktops using the available monitoring functions.

See Working with Sample Pages.

See “Understanding PeopleSoft MultiChannel Framework Mail Classes” (PeopleCode API Reference) and“Understanding Universal Queue Classes” (PeopleCode API Reference).

Configuring PeopleSoft CTI

PeopleSoft MultiChannel Framework includes support for PeopleSoft CTI. PeopleSoft CTI requires supporting computer-telephony middleware and additional configuration separate from other channels of PeopleSoft MultiChannel Framework.

See Understanding the PeopleSoft CTI Console.

Chapter 2

Understanding PeopleSoft MultiChannel Framework

PeopleSoft MultiChannel Framework

PeopleSoft MultiChannel Framework delivers an integrated infrastructure to support multiple interaction channels for call center agents or other PeopleSoft users who must respond to incoming requests and notifications on these channels.

PeopleSoft MultiChannel Framework can be used from any PeopleSoft application.

In this context, the word *channel* refers to the technology used to communicate during an interaction. PeopleSoft MultiChannel Framework supports the following channels:

- Voice (telephone).
- Web collaboration (chat).
- Email.
- Generic tasks.

The PeopleSoft MultiChannel Framework includes an HTML agent console, universal queueing, real-time task routing, customer-to-agent and collaborative chat, and centralized event logging.

PeopleSoft MultiChannel Framework Elements and Channels

This section discusses PeopleSoft MultiChannel Framework elements and channels.

PeopleSoft MultiChannel Framework Elements

PeopleSoft MultiChannel Framework comprises the following services and elements:

- Universal queue server, running on the Universal Queue server process (PSUQSRV).
- Real-time event notification (REN) server, running on the REN server process (PSRENSRV).
- MultiChannel Framework (MCF) log server, running on the MCFLOG server process (PSMCFLOG).
- MultiChannel console, the HTML interface through which users manage the channel interactions assigned to them.
- Chat windows, the HTML interfaces used for customer-to-agent and collaborative chat sessions.
- Agents, identified by their expertise and responsibilities.

- PeopleCode built-in functions and an email application package.
- GETMAILTARGET connector running under PeopleSoft Integration Broker.
- PeopleSoft MultiChannel Application Programming Interface (PSMCAPI) to enable server-side computer-telephony integration (CTI) integration.
- JavaScript MultiChannel Application Programming Interface (JSMCAPI) to enable a customizable CTI console and monitoring functions.

Each of these services and elements requires configuration.

In addition, each communication channel handled by PeopleSoft MultiChannel Framework requires supporting elements:

- CTI middleware to notify the system of telephone calls.
- An email server to store and serve email.
- Application pages to request customer-to-agent chat sessions and to provide context data and resolution logic for all interactions.
- Application pages or batch processes to enqueue generic events.

PeopleSoft MultiChannel Framework Channels

This section discusses support for communications channels offered by PeopleSoft MCF.

Voice

The agent console offers a softphone and full CTI support with Oracle-validated third-party CTI systems. Relevant application pages appear based on data attached to the call by the Interactive Voice Response (IVR) and CTI middleware.

Web Collaboration

PeopleSoft application pages can include Live Help buttons that initiate customer-to-agent chat sessions. The customer and agent chat windows are browser-based and do not require a client installation or applet download. The universal queue routes chat requests to the first available agent with the skills required to handle that request. The agent chat window displays relevant customer information and enables the agent to push web content to the customer. The agent can manage multiple chat sessions from the agent console.

Agents can also include peers and supervisors in chat conferences and transfer chat sessions to other agents or queues. Agents can also initiate collaborative chats with other agents on their buddy lists.

Email

PeopleSoft MCF enables applications to fetch Multipart Internet Mail Extensions emails from Post Office Protocol 3 (POP3) and Internet Message Access Protocol 4 (IMAP4) mail servers, store their parts in a database, and route the email to call center agents by either adding the email to worklists or enqueueing them on the universal queue. Email attachments can be stored in a database or stored in an attachment repository. Email attachments that are stored in the database or in the attachment repository are accessible by URLs from a browser. The repository and the database check user-based and role-based

security before retrieving an attachment. The email framework is built on PeopleSoft Integration Broker technology.

PeopleSoft MCF does not provide a mechanism to move existing email attachments that are stored in the attachment repository to the database.

When you choose to store email attachments in the database, and if an attachment cannot be stored in the database due to the unavailability of a connection, an error is displayed. In such cases, you must attempt to save the attachment again.

Note: If you choose to store email attachments in the database, ensure that you configure the default local node in Integration Broker.

PeopleSoft MCF supports emails conforming to the Simple Mail Transfer Protocol (SMTP) specifications including both inbound and outbound HTML email.

Generic Channel

Channels that are not provided by PeopleSoft MCF can be integrated by means of the generic channel to enqueue tasks onto the universal queue.

PeopleSoft MultiChannel Framework Universal Queue

The universal queue accepts, evaluates, and distributes incoming task requests from multiple communication channels: email, web chat, and generic notifications.

The universal queue handles email, chat, and generic tasks. It distributes workload across the call center, or any other pool of qualified users, based on the priority of the task and the availability of agents possessing the required skill level and language skills. Availability is based on agent presence and the cost of the new task (a measure of the task's impact on agent capacity) against the current workload of each agent.

Agents can forward tasks to other agents or to another queue. The task is removed from the transferring agent's workload and added to the accepting agent's workload.

Email and generic tasks that are not closed before the agent signs out persist in the database. Persisted tasks are reassigned to the same agent that accepted the tasks when the agent signs in again. A task that is not accepted, within configurable time limits, by the agent to whom it was assigned is reassigned to another qualified agent, if one is available. Tasks that are not resolved within configurable time limits are automatically escalated. Tasks that cannot be assigned to or are not accepted by any agent within configurable time limits are moved to an overflow table.

Voice tasks (CTI) are not queued or routed by the universal queue. They take precedence over all other tasks. However, the queue server adds the cost of voice tasks to the agent workload calculations it uses to queue and assign incoming tasks.

PeopleSoft MultiChannel Console

The agent console is the web-browser-based desktop from which the user manages all tasks, irrespective of channel. The console combines CTI, chat, email, and generic notice response tools into one window. Agents use the console to sign in, to select their current queue, to accept tasks, and to initiate and accept collaborative chat requests with buddy users. After the agent accepts a task, additional browser-based

windows appear to enable the agent's response. These windows are task-dependent and include elements developed specifically for supporting applications, such as email response management systems.

Note: The Multichannel Console link is available on the Action Menu, which displays when you click the Action button on PeopleSoft Pure Internet Architecture (PIA). The visibility of the link is controlled by the PTPT4700 permission list and the PeopleTools MCF Console role. Only users assigned with this permission list and role can view the Multichannel Console link on PIA. The exception to this rule are the PeopleTools Administrator and Portal Administrator users. See, Enabling PeopleSoft CTI.
To access the link and use its features, you must have the security permissions as required for the respective functions.

A custom CTI console can be created by means of the JSMCAPI.

PeopleSoft MCF Architecture

This section discusses MCF server architecture, chat architecture, and email architecture.

PeopleSoft MCF Server Architecture

The REN server (the PSRENSRV process) is essential to the framework architecture. MCF events are sent to REN servers, which then deliver them to recipients of those topics. The REN server is a modified web server using the HTTP 1.0 or HTTP 1.1 communications protocol. Communication with server processes and MCF browser windows is bidirectional, because the browser windows maintain persistent connections to the REN server. Events can be sent proactively to browser windows without polling or page refreshes. To provide a secure channel of communication to overcome concerns from PeopleSoft customers about sensitive data and its security, the REN server can be Secure Sockets Layer (SSL)-enabled. An SSL-enabled REN server provides secure communication that encrypts and provides client and server authentication.

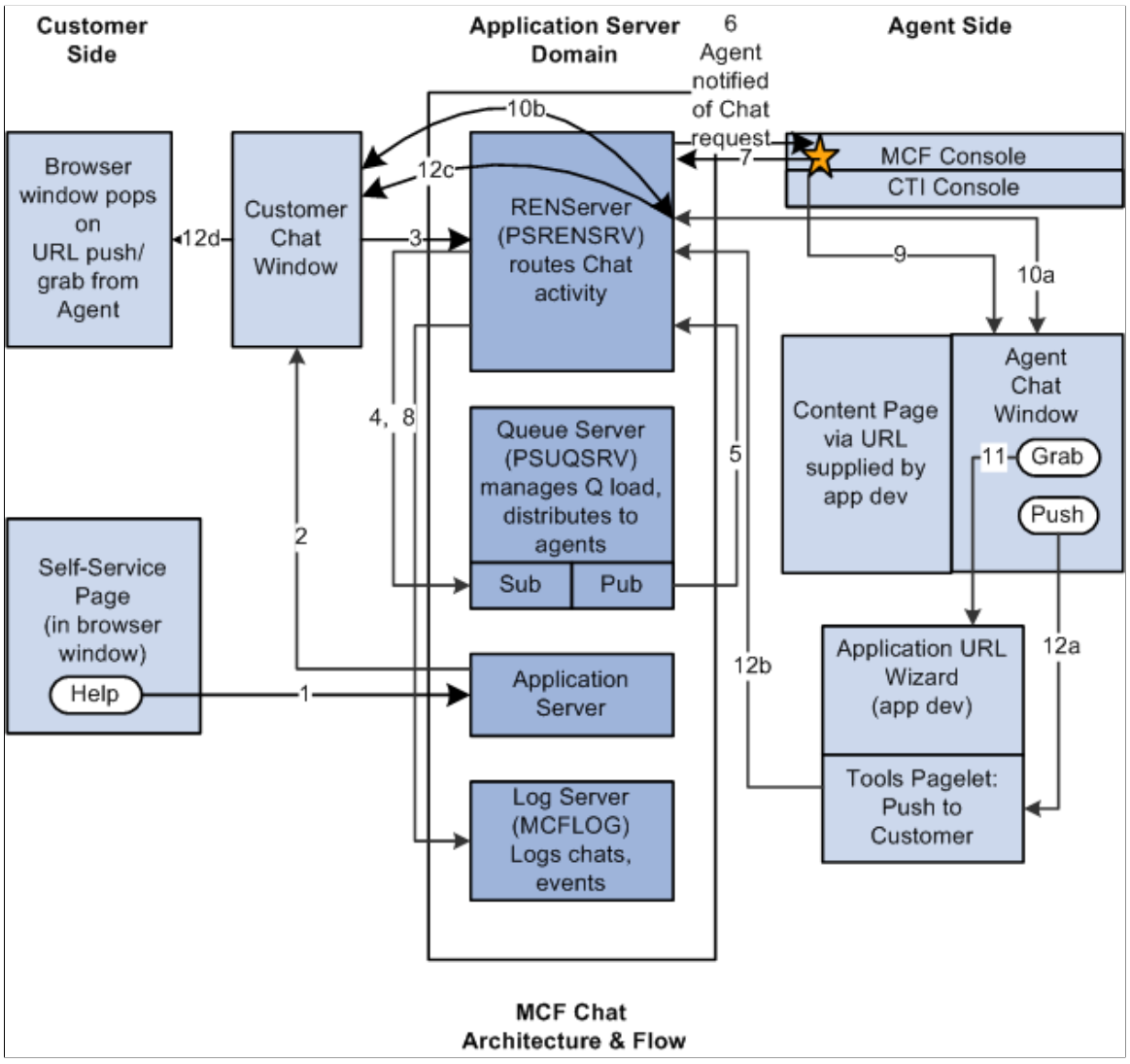
Applications send interaction and action requests (tasks) received from the supported communication channels to logical queues. The REN server notifies the universal queue server (the PSUQSRV process) responsible for that queue that a new task has arrived. Tasks are queued in order of priority until they can be assigned to an available agent qualified to respond to the task, at which time the queue server sends an assignment notification to the user’s MultiChannel Console through the REN server.

The queue server routes work requests (tasks) to users based upon a set of configurable policy properties that define which agents can handle what types of tasks and when the tasks can be assigned. The queue server manages state information about the current status of active agents and active tasks.

The MCF log server logs MCF events and chat content to the database. You configure logging levels on the MCF administration pages.

Chat Architecture

The following diagram illustrates the architecture of MCF chat and the flow of a chat session.



When a customer clicks the Help button on an application page:

1. InitChat() passes the following parameters to the application server:
   - Queue number.
   - Priority override.
   - Context page URL.
   - Query.

   See “InitChat” (PeopleCode Language Reference).

2. The application server posts to the REN server to notify the queue server that a customer chat is waiting.

The application server then returns the name and port of the REN server and the iScript to build the customer chat window.

3. The customer chat window appears and communicates with the REN server to receive all events on that chat topic.
4. The REN server notifies the queue server that a customer chat is waiting.

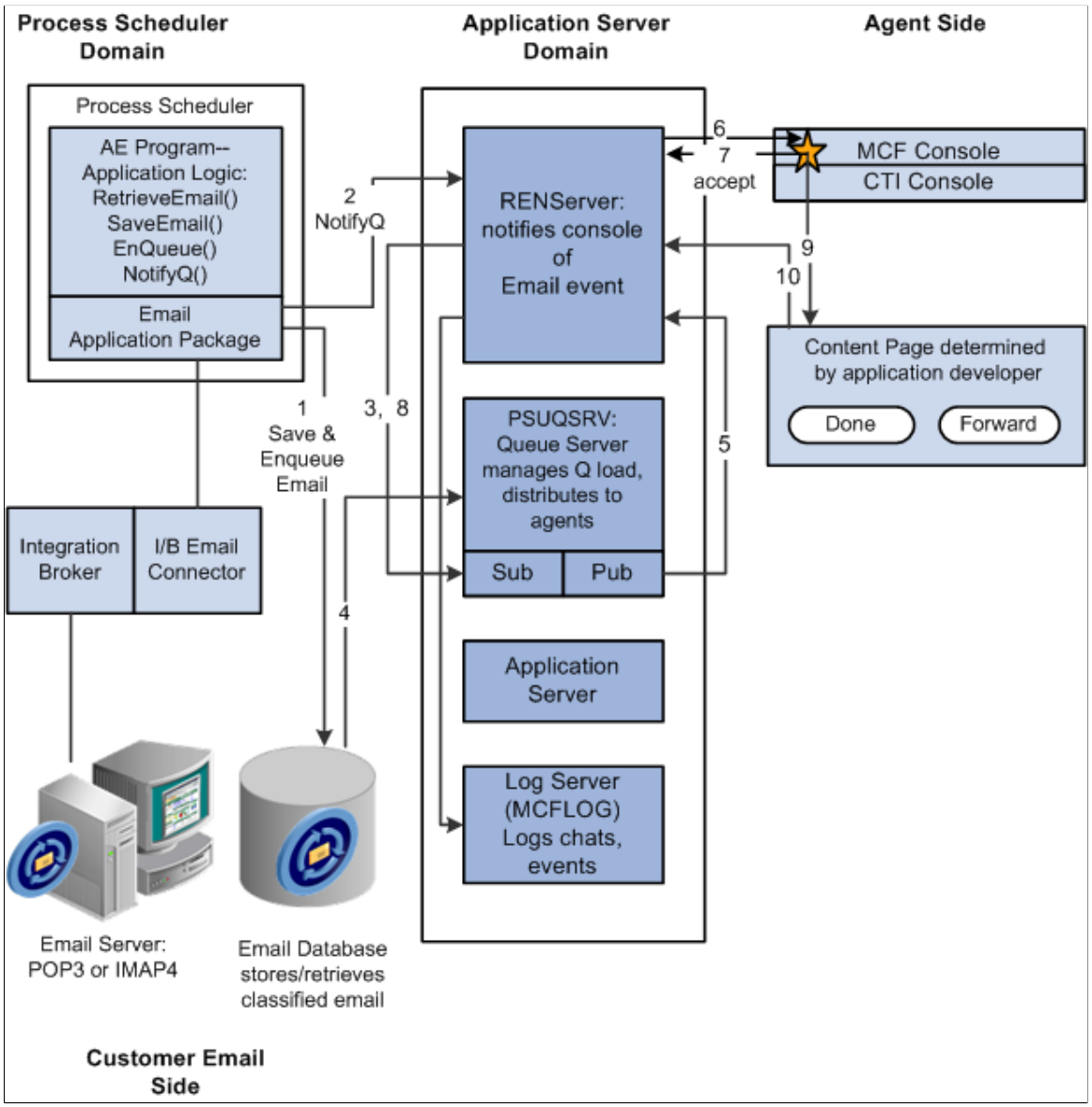
   The queue server determines the appropriate agent according to workload, cost, agent availability, skill level, and language.
5. The queue server tells the REN server to notify the agent that the agent has been assigned a chat.
6. The REN server notifies the agent from the MultiChannel Console that the agent has been assigned a chat.
7. The agent, from the MultiChannel Console, notifies the REN server to notify the queue server that the agent has accepted the task.
8. The REN server notifies the queue that the agent has accepted the task.
9. The MultiChannel Console displays an agent chat window.

   The agent responds to inquiry.
10. Agent to customer two-way communication is mediated by the REN server.
11. If the agent chooses to grab a URL, the Grab button invokes the application URL wizard.
12. If the agent chooses to push a URL, the Push button invokes a pagelet (labeled with an *A* in the diagram) to push a selected URL through the REN server (labeled with a *B* in the diagram) and customer chat window (labeled with a *C* in the diagram) to a new browser window.

The entire chat session can be logged through the MCF log server (labeled with a dotted line in the preceding diagram).

Email Architecture

This diagram illustrates the architecture of MCF email and the flow of email processing.



When a customer sends an email:

1. A PeopleSoft Application Engine program uses the MCF email application package classes and PeopleCode built-in functions to save and enqueue email in a database.
2. The PeopleSoft Application Engine program notifies the REN server that email has been enqueued.
3. The REN server notifies the queue server of the waiting email.
4. The queue server retrieves email information required to determine appropriate routing from the database.

   The queue server determines the appropriate agent to handle each email according to workload, cost, agent availability, skill level, and language.

5. The queue server tells the REN server to notify the agent that the agent has been assigned an email.
6. The REN server notifies the agent from the MultiChannel Console that the agent has been assigned an email.
7. The agent, from the MultiChannel Console, notifies the REN server to notify the queue server that the agent has accepted the task.
8. The REN server notifies the queue that the agent has accepted the task.
9. The MultiChannel Console displays an agent email window, as determined by the application developer.

   The agent responds to the email.
10. The agent's resolution of the task is communicated back to the REN server by either the Done or Forward button.

Email events can be logged to a database by the MCF log server.

Chapter 3

Configuring PeopleSoft Computer Telephony Integration

Understanding PeopleSoft CTI

Note: The CTI Console Java applet is no longer supported from PeopleTools 8.53. Refer to the Product Certifications link on My Oracle Support, https://support.oracle.com (sign-in required), for more information about supported products.

PeopleSoft CTI is a browser-based call-management system that helps call agents work more efficiently with customers. PeopleSoft CTI integrates Oracle-validated third-party CTI systems and your PeopleSoft applications. It exchanges data between the CTI system and your PeopleSoft applications so that the system automatically fills PeopleSoft transaction pages with the appropriate customer information—the information related to the caller.

With PeopleSoft CTI, you can perform the following tasks:

- Operate two lines or two extensions.
- Answer incoming calls.
- Release calls.
- Put a caller on hold.
- Monitor call status.
- Access PeopleSoft applications.
- Transfer callers.
- Initiate conference calls.
- Place an outbound call.

Note: PeopleSoft CTI supports Oracle-validated third-party CTI systems. In the following sections, when the phrase *your CTI middleware* or *CTI vendor* appears, assume that it refers to Oracle-validated third-party CTI systems, as appropriate for your installation.
CTI console developed with the JavaScript MultiChannel Application Programming Interface (JSMCAPI) can be customized, therefore features described in PeopleSoft CTI may not appear or may act differently.

The PeopleSoft CTI interface is incorporated *within* the MultiChannel Console.

Related Links

Understanding the PeopleSoft CTI Console

Managing Tasks with the MultiChannel Console
Communicating with Customers and Agents Using Chat

Understanding the PeopleSoft CTI Console

This section discusses PeopleSoft Computer Telephony Integration (CTI) and its components.

Note: The CTI Console Java applet is no longer supported from PeopleTools 8.53. Refer to the Product Certifications link on My Oracle Support, https://support.oracle.com (sign-in required), for more information about supported products.

PeopleSoft CTI

PeopleSoft CTI enables you to integrate your PeopleSoft applications with your call center. PeopleSoft CTI offers the following benefits:

- Seamlessly integrates your PeopleSoft application with Oracle-validated third-party CTI systems to improve agent productivity.

  Agents refers to the individuals who interact with your customers using CTI.

- Requires only that you install a supported web browser on the agent's workstation.
- Enables agents to take advantage of browser-based call management and automatic population of PeopleSoft transaction pages with the relevant customer data associated with an incoming call.
- Transmits Dual Tone Multi Frequency (DTMF) data.
- Handles outbound calls from automated systems.

PeopleSoft CTI is an optional component that you can integrate with the PeopleSoft MultiChannel Framework. This means that you can incorporate a CTI channel within the MultiChannel Framework. PeopleSoft CTI requires third-party middleware in the form of an Oracle-validated third-party CTI middleware.

In PeopleSoft CTI, the CTI middleware performs the call routing. The universal queue is not involved in routing calls. For an incoming call, the CTI middleware notifies the MultiChannel Console, which then notifies the queue server so that the agent's workload can be updated with the cost of a call.

For vendors using the adapter-based CTI solution, refer to Oracle Validated Application Integrations — Find a Partner Solution (http:www.oracle.com/partnerships/isv/integration/search.html)

PeopleSoft CTI Components

The PeopleSoft CTI Console works together with the Interactive Voice Response (IVR) system, the CTI middleware, an Automatic Call Distributor (ACD), and your PeopleSoft application.

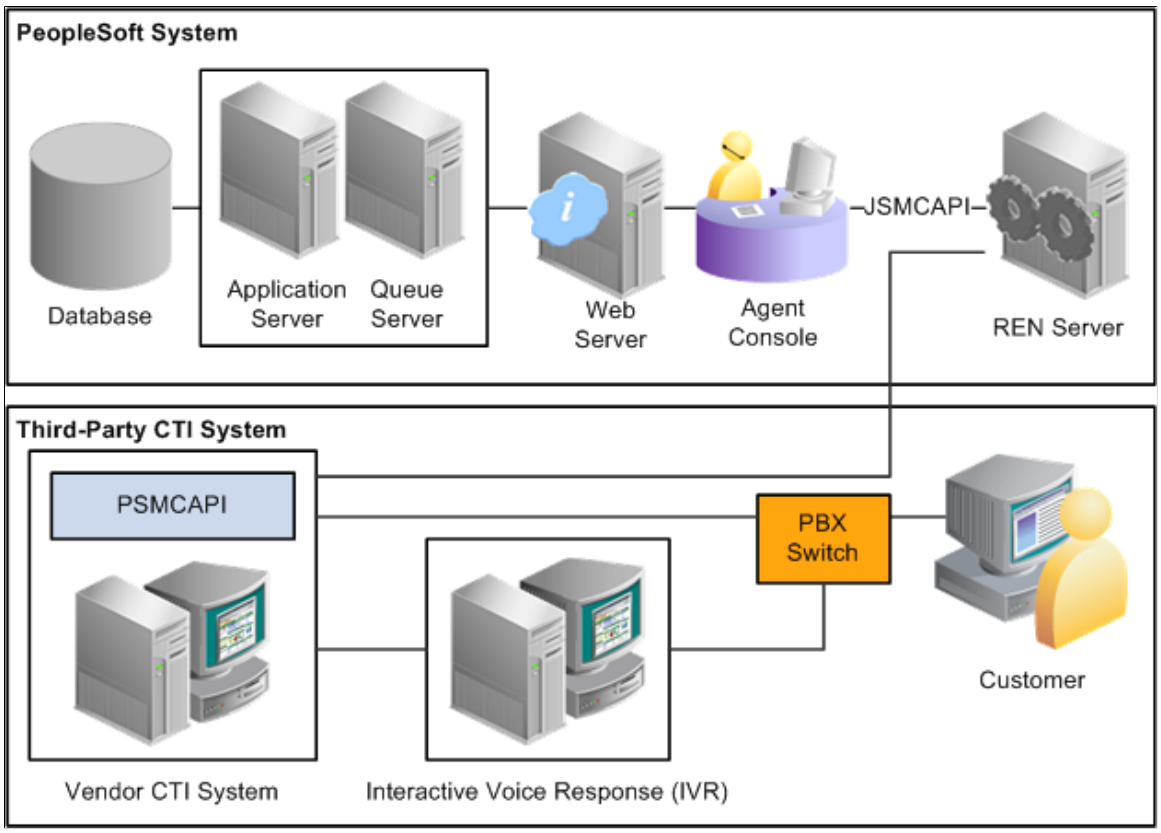
When a customer calls, the caller enters his or her information (for example, an account number) using the IVR system. Using this information, the CTI middleware routes the call to an appropriate ACD queue. The ACD sends the call to the next available agent on that queue and notifies the CTI middleware that an incoming call is on that Directory Number (DN). The CTI middleware, in turn, notifies the CTI

Console and passes the customer's information as attached data. The CTI Console uses the attached data to determine what PeopleSoft transaction page to open (pop-up) for the agent and what application data to retrieve from the database. The agent can manage the call using the CTI Console, which in turn communicates with the Private Branch Exchange (PBX) using the CTI middleware.

For PeopleSoft CTI systems, the CTI Console uses a JavaScript MultiChannel Application Programming Interface (JSMCAPI) to communicate with the CTI middleware that implements the PeopleSoft MultiChannel Application Programming Interface (PSMCAPI).

Configure PeopleSoft CTI to use the CTI console (JSMCAPI) on the CTI Type tab on the Configure CTI page.

This diagram illustrates PeopleSoft CTI architecture that uses PSMCAPI and JSMCAPI.



Related Links

"Understanding Internet Script Classes" (PeopleCode API Reference)

Using PeopleSoft CTI

This section discusses how to use the PeopleSoft Computer Telephony Integration (CTI).

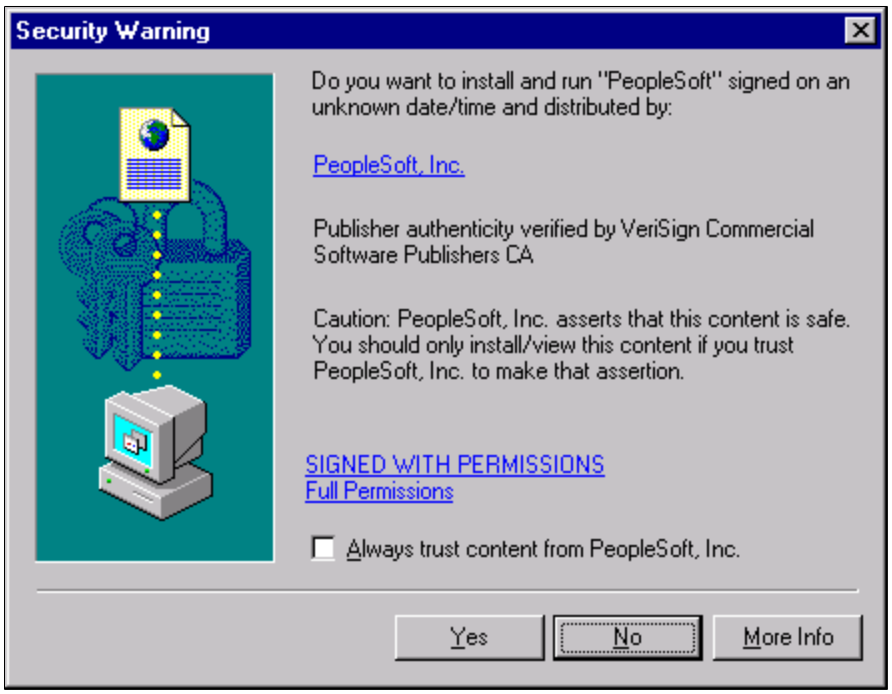
Getting Started

Getting started with PeopleSoft CTI is a three-step process that involves:

- Signing in to the PeopleSoft system.
- Specifying your extension information.
- Connecting to your CTI middleware.

The first time that you access PeopleSoft CTI using the applet-based solution, a Security Warning dialog box *may* appear prompting you to trust information from PeopleSoft. Click the **Yes** button.

This example illustrates the Security Warning dialog box.



The Security Warning dialog box may not appear if:

- You have previously accepted PeopleSoft applets as trusted.
- The system is not correctly installed.

Note: If you have any questions or concerns about this warning, contact your system administrator.

To sign in to PeopleSoft CTI:

1. On the PeopleSoft sign-in page, enter your PeopleSoft user ID and password as you normally do to sign in to the PeopleSoft system.
2. Click the **MultiChannel Console** link in the universal navigation header.
3. On the CTI control bar, set your configuration by performing the following actions:
    - Ensure that your agent ID appears beneath the **CTI Agent ID** label.
    - Ensure that the appropriate queue name appears beneath the **Queue** label.

      Queues are discussed in a subsequent section.

- If you are signing in from a workstation with a different extension, enter the current extensions in the **Extension** field.
- The **Extension Type** field indicates whether an agent can receive calls by way of a queue or just from a directory number (DN).

  If this value is set to *Queue,* the console offers options that are queue-specific, such as being able to log on to a queue.
- Click the **Activate** button to sign in to your CTI middleware.

  The CTI console appears.

Using the CTI Console

Change any registration parameters if necessary and click **Set** or **Activate** (depending on your version). After you do so, the CTI interface appears, which looks similar to the following:

Note: Note that the PeopleSoft CTI interface operates *within* the MultiChannel Framework Console. CTI users will see the MultiChannel Console only if their user ID belongs to a role that has MCF_AGENT or MCF_SUPR real-time event notification (REN) permissions, and WEBLIB_MCFLINK web libraries in their role's permission list.

This example illustrates the fields and controls on the CTI Console. You can find definitions for the fields and controls later on this page.

Note: Do not open two consoles for the same user on a single machine. Doing so can result in an error.

Field or Control	***Description***
 (register)	Select to register and unregister with the CTI middleware. The button acts as a toggle switch. When the green check mark appears, you are registered; when the red *X* appears, you are not registered.
Lines	Select the option to the left of the telephone icon to activate the associated line. The icon that you select determines the active line. All call actions that you choose apply only to the active line. The color of the icon reveals the activity on the line. The colors are: • Blue: Inactive. • Red: Ringing. • Green: On call. • Yellow: On hold.

Field or Control	*Description*
Select Call Action	Select actions related to calls, including *Answer, Hold,* and *Transfer.* Which call action options appear depends on your current status. For instance, the Hold call action is valid only while you are on the line with a caller. Call action options are discussed in detail in the following section.
Status	Displays the agent status, as in *Ready* or *Not Ready* (to receive calls). It can also show *Ready/DND* (do not disturb) if the agent is not using a queue.
Q (queue)	If the agent belongs to an automatic call distributor (ACD) group, the group appears here.
Messages	Informational messages appear in the right corner of the CTI Console. Examples of such messages are *Dialing, Connected,* and *Released.* These messages do not persist. **Note:** Error messages appear in separate windows.
Call Duration	The system tracks the amount of time spent on calls for each line.
Incoming Call Information	Displays the string associated with the Descr call variable. For example, *Gold Customer.*

Note: In the applet-based solution, some CTI systems will automatically sign out the CTI agent when the CTI console is closed. For CTI systems in which this is not automatic, the option can be set on the agent Personalization page (select **PeopleTools** > **MultiChannel Framework** > **CTI Configuration** > **Configure CTI Agents** > **Personalization.**)

Selecting Call Actions

The **Select Call Action** drop-down list box contains all of the options that you have for handling calls. Depending on the status of the agent or the telephone line, certain selections from the drop-down list box are not available.

After you select a call action, two buttons, the **Go** performs the selected call action.

The following list describes the call actions:

Field or Control	*Description*
Dial	When you are in Agent Ready or Agent Not Ready mode, you can call another party.

Field or Control	*Description*
Answer Call	This button appears and flashes when an incoming call is waiting to be answered.
Transfer Mute	Transfers the caller to the desired number without speaking to the intended recipient. A dialog box appears enabling you to enter the extension of the person to whom you want to transfer the caller. **Note:** In the applet-based solution, this action is not supported on Aspect switches.
Transfer	Transfers the caller to the desired number. You have the opportunity to speak with the recipient before transferring the caller. A dialog box appears enabling you to enter the extension of the person to whom you want to transfer the caller.
Conference	Enables you to add one or more individuals to your call.
Hold	Places the caller on hold.
Retrieve	Takes the caller out of *Hold* status. This option appears only when the caller is on hold.
Release	Disconnects the caller. This option is available as a selection only after a call is answered.
Change	When an agent status is unknown, the system sets the status to *Change* so that the user can set the status manually to match the status for the telephone.

Note: You can transfer to or initiate a conference call with individuals who are not enabled to access PeopleSoft CTI. Their phone rings, but remember that the pop-up window showing customer data does not appear.

In the applet-based solution, the following call actions are available depending on whether you are using a DN or Queue configuration:

Field or Control	*Description*
Queue	Select from: *Log on:* Enables the agent to log an extension on to a queue. *Ready:* Indicates that the agent is ready to receive incoming calls. This option is available when the extension is associated with a queue and the status bar reads Agent Not Ready. *Not Ready:* Stops incoming calls. This option is available when the extension is associated with a queue and the status bar reads Agent Ready. *Log off:* Enables the agent to log an extension off a queue. When logged off, the agent is no longer participating in the queue.
DN	This option is available when the extension is not associated with a queue. *Ready:* Indicates that the agent is ready to receive incoming calls.

In the adapter solution, the availability and meaning of the call actions are determined by the third-party CTI vendor. Consult your CTI vendor for details.

Answering a Call

After you have signed in to PeopleSoft CTI, you can receive calls. For each incoming ACD call to your extension, the telephone extension icon turns red. After you have accepted a call, the system does not send you more incoming calls until you have completed the current call.

To answer a call:

1. Select the radio button to the left of the telephone icon that has turned red.

   The *Answer* option is automatically selected as the current option in the drop-down list box when an incoming call arrives.

2. Click **Go.**

   The pop-up browser launches with the appropriate PeopleSoft transaction page displayed. The system determines which page to display based on caller information sent by the CTI middleware.

   After you have answered a call, you enter the not available status.

If an agent erroneously cancels a call instead of clicking **Go** when accepting an incoming call, the agent can recover the call by:

1. Selecting a call option.

2. Selecting the line that is receiving the incoming call.

Note: If CTI screen pop-ups are configured, the CTI console attaches a CTI miniconsole to the screen pop-up page. This miniconsole takes a few seconds to initialize before it can be used.
In the Javascript console, you can configure whether the miniconsole appears.

Transferring a Caller

Occasionally, you need to transfer callers to other agents. PeopleSoft CTI supports two types of transfers:

- Transfer Mute: This option enables you to transfer a call without speaking to the target agent before transferring the call.
- Transfer: This option is also known as a consultative transfer, which means that you consult with the target agent before transferring the call.

Note: You can always transfer or invite users to a conference call even if the called party is not CTI-enabled. The non-CTI-enabled users do not get pop-up windows, but their phones still ring.
When you initiate a transfer or conference call on the Cisco system using the applet-based solution and the contacted party does not answer, wait until the system notifies you of the unsuccessful connection before attempting another action. This wait can be 20 to 30 seconds.
In the applet-based solution, Cisco ICM does not notify client applications about some call events during two-step transfers or conferences. As a result, the state of the PeopleSoft CTI Console may not stay synchronized with that of the teleset, especially if the teleset is used to initiate or complete these call actions. The CTI console should automatically resynchronize with the teleset when the call is completed.

To perform a transfer mute:

1. Select the appropriate telephone line.

   The selected line must be green.

2. From the **Select Call Action** drop-down list box, select *TransferMute.*

3. In the **Phone No.** edit box, select the number that you want to dial.

   The drop-down list box contains all of the numbers from the shared phone book and agent phone book. If the number that you want to dial does not appear, click **Dial other number** and manually enter the number.

4. Click **Go.**

   This connects the caller to the new agent and releases your line.

   The system prompts the recipient of the transfer that it is transferring a call from your extension. When the recipient accepts the transfer, the PeopleSoft page connected to the caller’s case opens as it did when you first received the call.

To perform a transfer (consultative):

1. Select the appropriate telephone line.

   The selected line must be green.

2. From the **Select Call Action** drop-down list box, select *Transfer.*

3. In the **Phone No.** edit box, select the number that you want to dial.

   The drop-down list box contains all of the numbers from the shared phone book and agent phone book. If the number that you want to dial does not appear, click **Dial other number** and manually enter the number.

4. Click **Go.**

   When you use one extension with two lines, the outbound call you make to the agent to whom you are transferring the incoming call gets initiated on your second line and the incoming call gets placed on hold. When the outbound call is established, you can consult with the recipient and place that call on hold. To complete the transfer, you need to return to the first line and select *Complete* and click **Go.** This action releases the call on the second line and transfers the call on the first line to the recipient of the transfer.

   When you use two extensions, each with one line, the CTI Console does not have access to the outbound call to the intended recipient of the transfer. When the outbound call is established, you can consult with the recipient. You do not have to toggle between the two lines, and you cannot put the recipient on hold.

5. To complete the transfer, select *Complete* and click **Go.**

Initiating Conference Calls

If you need the assistance of other agents to answer a caller's questions, you can use the conference feature to include the appropriate agents on a call.

To initiate a conference call:

1. Select the appropriate telephone line.

   The selected line must be green.

2. From the **Select Call Action** drop-down list box, select *Conference.*

3. From the drop-down list box of all numbers from the shared and agent phone books, select a number to be dialed.

   If the number is not there, select *Dial other number* to access an edit box and enter the number to be dialed.

4. Click **Go.**

   The system notifies the target agent of the incoming call (conference). The PeopleSoft page associated with the caller’s case opens for the target agent as it did for you when you first received the call.

   This feature depends upon two parameters set up by the administrator:

   - Default Screen Pop-up URL.
   - Personalization: Screen Pop-up Mode.

     If the default URL for screen pop-up is set and the screen pop-up mode is 0 (pop up when incoming) for the second agent, the agent gets the screen pop-up as soon as the call is transferred.

If the screen pop-up is set to 1 (pop up after answer), the screen pops up only after the first agent completes the transfer or conference call. However, if the default URL for the screen pop-up is not set, then whether the mode is 0 or 1 doesn't matter. In that case, the second agent gets the screen pop-up only after the first agent completes the transfer or conference call.

When you use one extension with two lines, the outbound call that you make to the agent to whom you are inviting to the conference gets initiated on your second line and the incoming call gets placed on hold. After the outbound call is established, you can consult with the third party, and place that call on hold. To complete the conference, you need to return to the first line and select *Complete* and click **Go.** This action releases the call on the second line and starts the conference on the first line.

When you use two extensions, each with one line, the CTI Console cannot access the outbound call to the third party. When the outbound call is established, you consult with the target agent. You do not have to toggle between the two lines, and you do cannot put the recipient on hold. To start the conference, select *Complete* and click **Go.**

5. After consulting with the target agent, select *Complete* from the **Select Call Action** drop-down list box, and click **Go.**

Working with the Hold Status

Putting calls on hold and retrieving calls on hold is likely to be the call action that you perform most.

To place a call on hold:

1. Select the appropriate telephone line.

   The selected line must be green.

2. From the **Select Call Action** drop-down list box, select *Hold.*
3. Click **Go.**

To retrieve a call on hold:

1. Select the appropriate telephone line.

   The selected line must be green.

   The retrieve option is automatically selected as the current option in the drop-down list box when a call is on hold.

2. Click **Go.**

Disconnecting a Caller

After you have finished a call, you need to release the call.

To release a call:

1. Select the appropriate telephone line.

The selected line must be green.

2. On the console control bar, select *Release* from the **Lines** drop-down list box.
3. Click **Go.**

   If you are using the applet-based solution, the system automatically places you in wrap-up mode, which enables you to complete any remaining work before accepting more incoming calls. Technically, when you are in wrap-up mode, your status is *Agent Not Ready.*

   If you are using the adapter-based solution and you have configured wrap-up mode in your CTI middleware, the console can automatically place you in a *Not Ready* or *Work Not Ready* state, which enables you to complete any remaining work before accepting more incoming calls.

   When you are ready to accept incoming calls, select *Agent Ready.*

Switching Agent Ready Status

Your agent status determines whether you can receive incoming calls.

In the applet-based solution, agent statuses include:

- *Agent Ready*

  To activate *Agent Ready* status:

  From the **Select Call Action** drop-down list box, select *Agent Ready.*

  When you are ready, the system routes incoming calls to your extensions.

- Do Not Disturb

  To activate Do Not Disturb status:

  From the **Select Call Action** drop-down list box, select *DND.*

  With Do Not Disturb, your extensions do not accept incoming calls.

  Note: This status is not available to agents associated with an ACD queue.

- *Agent Not Ready*

  To activate *Agent Not Ready* status:

  From the **Select Call Action** drop-down list box, select *Agent Not Ready.*

  This status is typically used when agents are at their desks, but temporarily unable to receive calls. While you are not ready, the system routes calls to other available agents.

  Note: This status applies only to agents associated with an ACD queue.

Note: In the adapter-based solution, the adapter provider is responsible for determining what states and statuses are available in the drop-down list boxes, as well as the meaning of those states.

Dialing an Outbound Call

You can use PeopleSoft CTI to place a call while the agent status is either *Agent Ready* or *Agent Not Ready.*

Note: If you have multiple extensions assigned to you, *do not* call one of your extensions from the other.

To place an outbound call:

1. Select your status from *Agent Ready* or *Agent Not Ready.*
2. Select the option next to the telephone icon representing a free line.

   For example, if you had a customer on hold on one line, you would select the icon for the second line.
3. From the **Select Call Action** drop-down list box, select *Dial.*
4. From the drop-down list box showing all numbers from the shared and agent phone books, select a number to be dialed.

   If the number is not there, select **Dial other number** to get an edit box and enter the number to be dialed.
5. Click **Go.**

   As with any other call you receive, you can access all the call actions for calls that you initiate. You can transfer the person that you've called, place the line on hold, or initiate a conference with another party.

   Note: If you are using the applet-based solution, the system places you in *Agent Not Ready* status until you release the call. If you are using the adapter-based solution, you must configure your CTI middleware call actions.

   When you make an outbound call:

   - You can specify a URL to display a page.

     When you display the page, the outbound call data is also attached to the URL so that an application can collect information such as the automatic number identification (ANI), dialed number identification service (DNIS), and so on.
   - A context ID (such as a customer case number or an invoice number) is attached to the call data.

     This allows PeopleSoft application context to be passed on to CTI middleware, where it can be stored. This context can be used to establish a relationship between the outbound call and the application context when the call was made.

Related Links

Creating a List of Frequently Dialed Phone Numbers

Completing a Call

When you disconnect or release a call from the miniconsole, the system disconnects you from the CTI middleware, and the miniconsole becomes disabled (unavailable for entry). However, the PeopleSoft page remains active so that you can finish updating information if needed.

The availability of an agent to accept incoming calls depends on how the system administrators of the agent have set up the agent's CTI middleware.

Using Hot Keys

Hot keys are combinations of keyboard buttons that you can press instead of using a mouse. To help you easily select options, PeopleSoft CTI offers the following hot keys:

Hot Key	*Description*
Alt + R	Registers your phone extensions with the CTI middleware.
Alt + 1	Makes Extension 1 the active line.
Alt + 2	Makes Extension 2 the active line.
Alt + S	Presents a list of applicable call actions for you to select.
Alt + P	Presents a list of frequently called telephone numbers for you to select.
Alt + G	Enables you to perform a call action.
Alt + C	Enables you to cancel a call action.
Alt + Z	Enables you to check agent status.

CTI Components Requirements

This section describes components required for PeopleSoft CTI.

PeopleSoft MultiChannel API

The PeopleSoft MultiChannel API (PSMCAPI) is a Java API and software development kit (SDK) that provides server-side connectivity with the PeopleSoft CTI system. PSMCAPI enables third-party telephony vendors and system integrators to integrate with PeopleSoft applications.

PSMCAPI and JSMCAPI are installed during PeopleTools installation.

This table provides the installed locations of PSMCAPI and JSMCAPI components:

Component	*Location*
PSMCAPI Java archive	<PS_HOME>\sdk\psmcapi\dist\lib
SMACKED configuration files: • psmcapilog.properties • renclient.properties	<PS_HOME>\sdk\psmcapi\dist\config
JSMCAPI JavaScript file	<PIA_HOME>\webserv\<domain>\applications\peoplesoft\PORTAL\ps\pMCF

Working With Older Adapters

For some implementations using older versions of third party adapters, you may require a version of psmcapi.jar built with an older version of JDK. PeopleTools provides a previous version of the PSMCAPI for use with older adapters.

This table provides the installed locations of the PSMCAPI components provided for use with older adapters:

Component	*Location*
PSMCAPI Java Archive	PS_HOME\sdk\psmcapi\dist\archive\lib
PSMCAPI configuration files	PS_HOME\sdk\psmcapi\dist\archive\config

Configuring PSMCAPI

Two files, psmcapilog.properties and renclient.properties, include parameters to configure PSMCAPI. Configure the logging characteristics of PSMCAPI in psmcapi.properties. To have PSMCAPI generate full debug logs, set com.peoplesoft.pt.mcf.level to FINEST.

The following table lists all the parameters in renclient.properties:

Parameter	*Default Value*	*Description*
interval_heartbeat_server	30000	Heartbeat interval in milliseconds from PSMCAPI to console.
interval_heartbeat_client	30000	Heartbeat interval in milliseconds from console to PSMCAPI.

Parameter	*Default Value*	*Description*
maxsize_eventqueue	100000	Maximum number of events in the event queue.
interval_event_expired	300000	Expiration interval in milliseconds for an event in the event queue. An expired event is never published to the client/ console and is discarded from the event queue.
interval_topic_reaper	3600000	Topic reaper interval in milliseconds.
number_of_requests_in_requestschunk	1	The number of requests that JSMCAPI sends at a time during auto-recovery.
waitingtime_between_requestchunks	1	The time elapsed between two consecutive JSMCAPI request queues during auto-recovery.
mtu_size	0	The Maximum Transmission Unit size of your computer or network. Set mtu_size to 0 (zero) for no TCP packet padding, or to the maximum transmission unit (MTU) size for your network or computer to remove TCP acknowledgement delays.
heartbeats_to_miss	2	The number of heartbeat intervals to wait before removing a nonresponsive client
psmcapi_heartbeats_to_miss	5	The number of PSMCAPI heartbeat intervals to wait before removing a nonresponsive client.
tcp_nodelay	True	TCP no delay. Set to True to disable the TCP Nagle algorithm.
disable_session	False	This parameter is used when multiple PSMCAPI implementations subscribe to the same REN server. Set to True to disable the internal session and heartbeat listeners.

Required Security for PSMCAPI

CTI agents require only that the MCF Agent security object be enabled on the REN Permissions page of their associated permission list. The MCF Agent security object includes security for both CTI and other MCF channels.

To enable CTI configuration pages, authorize the CTI Type, CTI Applet, and CTI Console panel items on the PT_CTI page permissions for the appropriate permission list.

Also enable WEBLIB_MCF weblib permissions for the appropriate permission list.

Related Links

Configuring REN Servers

"Security Administration Overview" (Security Administration)

JavaScript MultiChannel API

The JavaScript MultiChannel Application Programming Interface (JSMCAPI) is an interface that application developers can use to generate the CTI Console or to enable CTI functionality on a PeopleSoft Pure Internet Architecture page. The JSMCAPI builds on the real-time event notification (REN) JavaScript client. JSMCAPI uses standard JavaScript.

Related Links

Using and Demonstrating JSMCAPI

Configuring PeopleSoft CTI

This section describes how to install and configure the PeopleSoft CTI system. This information assumes that you already have a functioning CTI system installed and configured at your site.

Installing PeopleSoft CTI

When you run the PeopleSoft Pure Internet Architecture setup program, the PeopleSoft CTI files are installed automatically to your web server.

Note: You do not need to select any additional options from the install program dialogs. The CTI files are installed by default.

After you have run the PeopleSoft Pure Internet Architecture setup program, enable the PeopleSoft CTI Console and configure the system as discussed in the following sections.

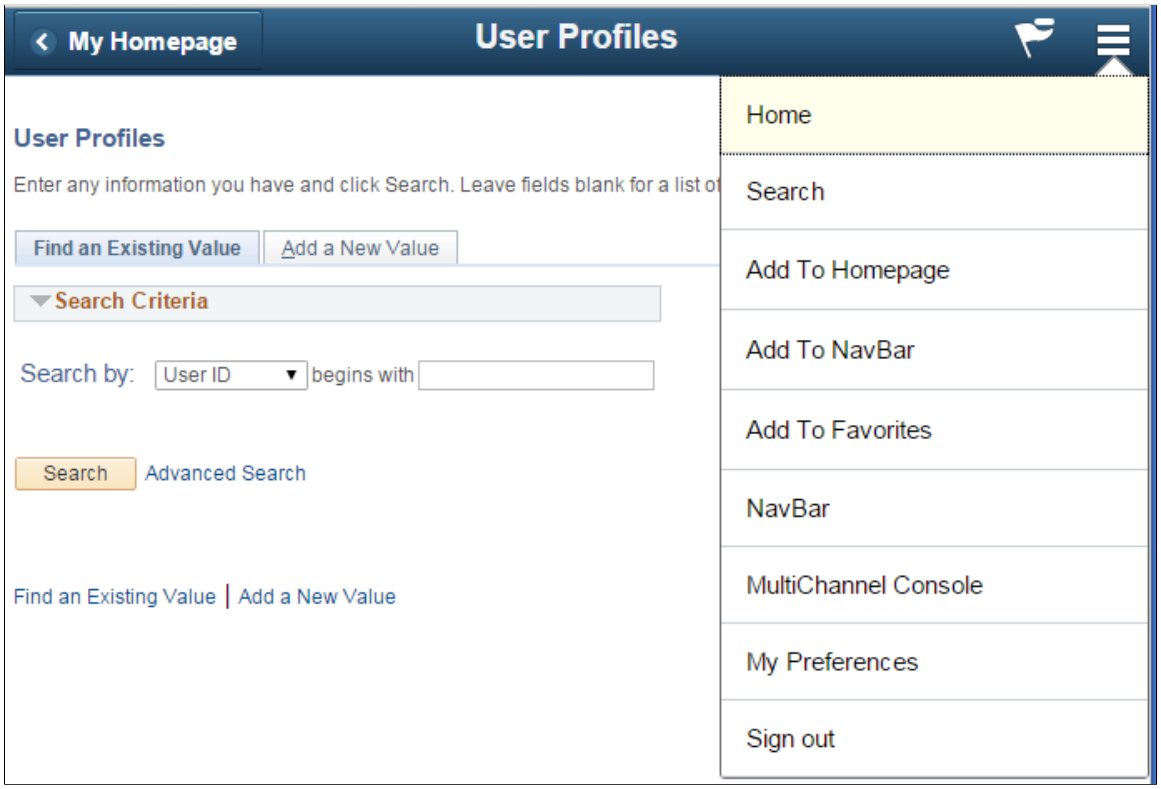
See the product documentation for *PeopleSoft 9.2 Application Installation* for your database platform and *CTI vendor*.

Enabling PeopleSoft CTI

If a user is set up as a CTI agent or a MultiChannel Framework agent and if the user is assigned the PTPT4700 permission list and the PeopleTools MCF Console role, the Multichannel Console link appears

listed on the Action Menu (in the upper-right portion of the screen) of the PeopleSoft Pure Internet Architecture. The following example shows the Multichannel Console link displayed in the PeopleSoft Pure Internet Architecture:

This example illustrates the MultiChannel Console link displayed on the Action Menu on the PeopleSoft Pure Internet Architecture.



Users who do not have the PeopleTools MCF Console role do not see this link.

Note: Blended agents—agents who use both the CTI and MCF consoles—must sign in to queues supported by the same REN server cluster.

See Using the CTI Sample Console.

Related Links

Configuring CTI Agents

Configuring CTI Console Type

Access the CTI Type page using either of the following navigation paths, whichever is appropriate to you:

PeopleTools > **MultiChannel Framework** > **CTI** > **CTI Type** (if you are using the CTI server) or **PeopleTools** > **MultiChannel Framework** > **Third-Party Configuration** > **CTI Configuration** > **CTI Type** (if you are using a third-party routing server).

This example illustrates the fields and controls on the CTI Type page.

Select the **Is Applet** check box if you are using the CTI Java applet console.

Clear the **Is Applet** check box if you are using the CTI JavaScript (JSMCAPI) console.

A CTI configuration contains all the information required for a user to be able to connect to a CTI server.

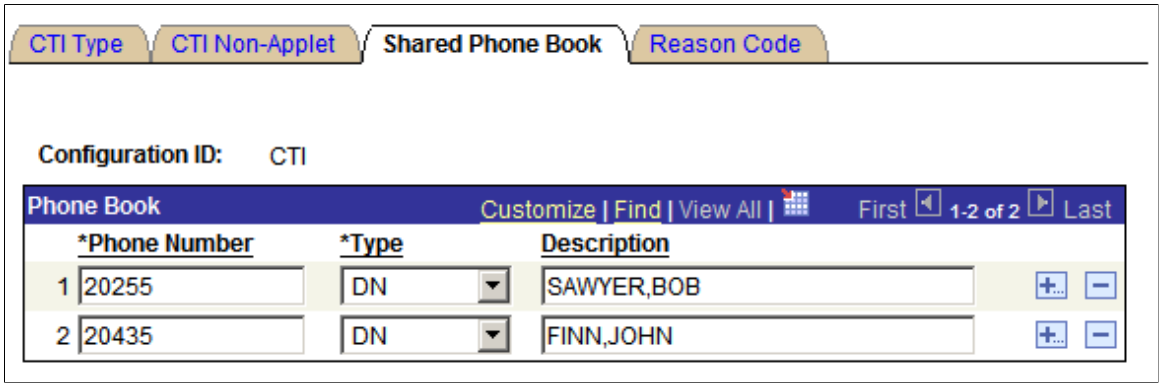
Note: The Configuration ID is created when CTI is configured for the first time.

Creating a List of Frequently Dialed Phone Numbers

Access the Shared Phone Book page using either of the following navigation paths, whichever is appropriate to you:

PeopleTools > **MultiChannel Framework** > **CTI** > **Shared Phone Book** (if you are using the CTI server) or **PeopleTools** > **MultiChannel Framework** > **Third-Party Configuration** > **CTI Configuration** > **Shared Phone Book** (if you are using a third-party routing server).

This example illustrates the fields and controls on the Shared Phone Book page. You can find definitions for the fields and controls later on this page.



On this page, you can manage a list of frequently dialed phone numbers for a specific CTI configuration. These numbers appear when an agent connected to that CTI configuration selects the drop-down list box when dialing a number from the CTI Console. This saves the agents from manually entering any dialed numbers present in that CTI configuration when making outbound calls.

Note: Phone lists are updated on the CTI Console only after the CTI Console launches. To refresh phone lists, refresh the browser and reactivate the console.

Field or Control	*Description*
Phone Number	Enter a frequently dialed phone number to associate with this configuration.
Type	Select one of the following values: • *DN* (directory number): The number that identifies a telephone set on a PBX or in the public network. The caller dials this number to establish a connection to the addressed party. The DN can be a local PBX extension (a local DN) or a public network telephone number. • *Queue* The directory number that identifies an ACD queue or group. Calls to a group are distributed to agents belonging to the group, according to ACD algorithms.
Description	Add a description for the telephone number.

Entering Default Screen Pop-Up URL

The default URL provides a way to enable CTI screenpops if data from the voice task should not be used to construct the base of the pop-up URL. If the default URL is populated, then it will be used for the screenpop, even if sufficient user or call data is provided in a voice task to construct a URL.

Note: User and Call data may be used as parameters to the URL.

Access the Miscellaneous page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **CTI Configurations** > **CTI Popup Screen URL**

Note: This page is available through the CTI server only.

This example illustrates the fields and controls on the Miscellaneous page. You can find definitions for the fields and controls later on this page.

Set default screen pop-up URL parameters on the Miscellaneous page.

Field or Control	*Description*
Use Default Screen Popup URL	Select if you want to set a default URL for the pop-up screen. The system uses the default method to determine what URL to use for the page launched for incoming calls. The system uses that default URL to examine the user data attached to that call. However, for customers who do not want to attach this user data to incoming calls, this option enables you to use the same URL for all pop-up screens. After selecting this option, enter the URL in the **Default Screen Popup URL** edit box.
Default Screen Popup URL	If the **Use Default Screen Popup URL** check box is selected, enter the value for the default URL. **Note:** To ensure that the user does not have to sign in to the pop-up window, the domain of the default URL must exactly match the domain of the sign-in URL. If the sign-in URL has no domain, the machine name of the default URL should be the same as the machine name of the sign-in URL. If either the domain or the machine name do not match, the system prompts the user for the user ID and password for the first pop-up screen. **With Matching Domains:** Within the same domain, such as example.com, the machine names can be different. For example: *Signon URL:* http://ntserver1.example.com/peoplesoft8/signon.html *Default URL:* http://uxserver2.example.com/servlets/iclientservlet/peoplesoft8/?ICType=PANEL&Menu=UTILITIES&Market=GBL&Component=MESSAGE_CATALOG1&Target=Main1&LANGUAGE_CD=ENG&MESSAGE_SET_NBR=1 **Without Matching Domains:** Without the matching domains, the machine name should be the same in both URLs. For example: *Signon URL:* http://ntserver1/peoplesoft8/signon.html *Default URL:* http://ntserver1/servlets/iclientservlet/peoplesoft8/?ICType=PANEL&Menu=UTILITIES&Market=GBL&Component=MESSAGE_CATALOG1&Target=Main1&LANGUAGE_CD=ENG&MESSAGE_SET_NBR=1

Related Links

"Understanding Single Signon" (Security Administration)

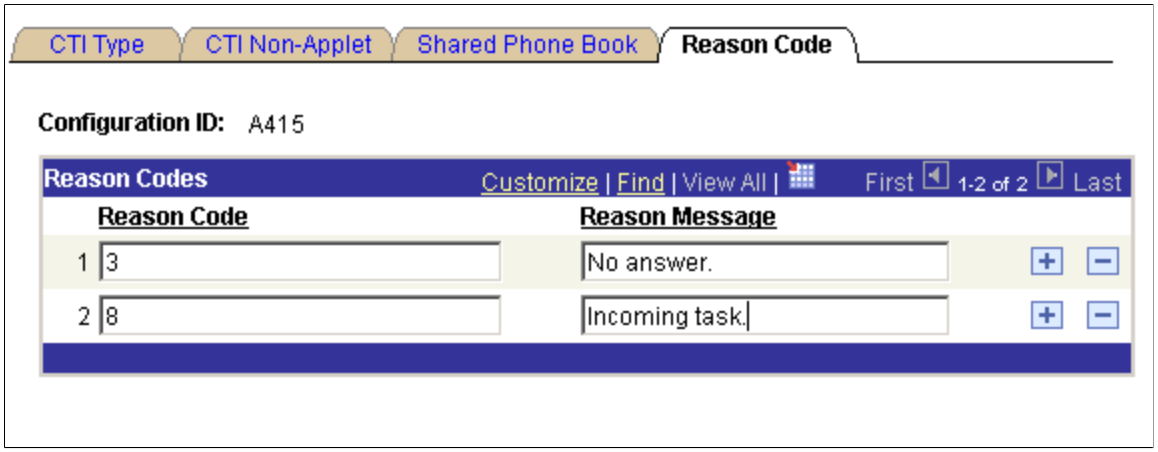
Using the Reason Code Page

Access the Reason Code page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Third-Party Configuration** > **CTI Configuration** > **Reason Code**

Note: This page is available only to third-party vendors.

This example illustrates the fields and controls on the Reason Code page.



This page is used by the third-party vendors to define their customized codes.

Related Links

Understanding JSMCAPI

Configuring PeopleSoft CTI Using Adapters

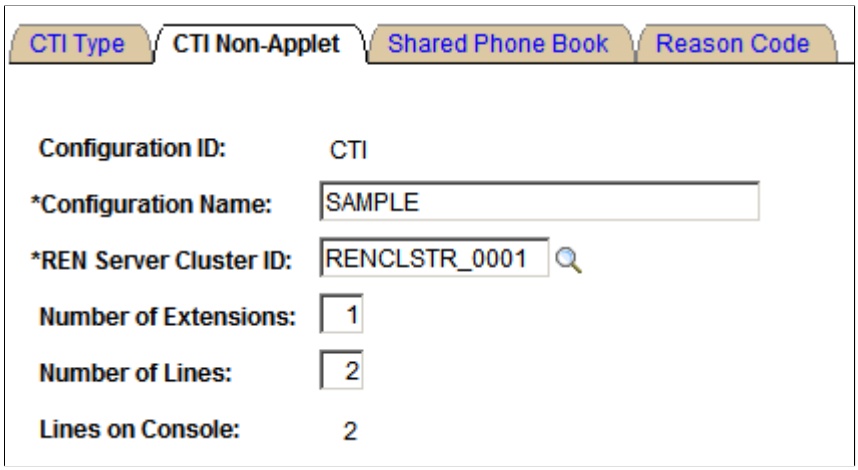
This section discusses how to configure the CTI console.

Configuring the CTI Console

Access the CTI Configuration page using either of the following navigation paths, whichever is appropriate to you, to configure the CTI (JSMCAPI) console:

PeopleTools > **MultiChannel Framework** > **CTI** > **Configuration** (if you are using the CTI server) or **PeopleTools** > **MultiChannel Framework** > **Third-Party Configuration** > **CTI Configuration** > **Configuration** (if you are using a third-party routing server).

This example illustrates the fields and controls on the CTI Non-Applet page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Configuration ID	Displays the name of the CTI configuration. The name cannot be modified after it is created.
Configuration Name	Add a descriptive name to help identify the configuration.
Number of Extensions	Enter the number of extensions or directory numbers associated with the telephone.
Number of Lines	Enter the number of lines associated with each extension. Depending on your configuration, you can enter up to two lines.
Lines on Console	The CTI Console supports up to two lines. The only supported configurations are two extensions with one line each, one extension with two lines, and one extension with one line.

Configuring PeopleSoft CTI Queues and CTI Agents

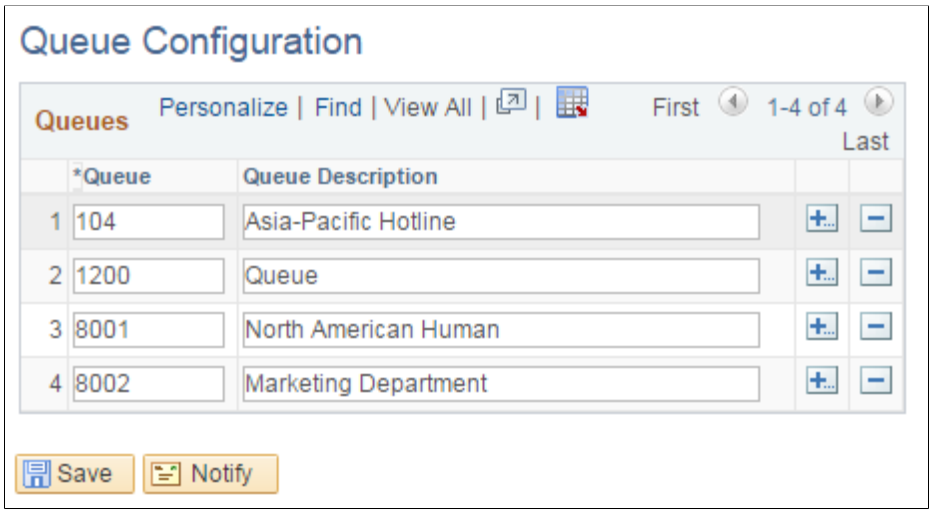
To configure CTI queues and agents, use the Queue Configuration (PT_CTI_QUEUE) and CTI Agent Configuration (PT_CTI_AGENT) components.

Configuring CTI Queues

Access the Queue Configuration page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **CTI Configuration** > **Configure CTI Queues**

This example illustrates the fields and controls on the Queue Configuration page. You can find definitions for the fields and controls later on this page.



Use this page to add queues for agents.

Field or Control	*Description*
Queue	Enter the directory number identifying an ACD group. Calls to a group are distributed to ACD agents belonging to that group, according to ACD algorithms. **Note:** Queue names can be alphanumeric.
Queue Description	Enter a brief description of the queue.

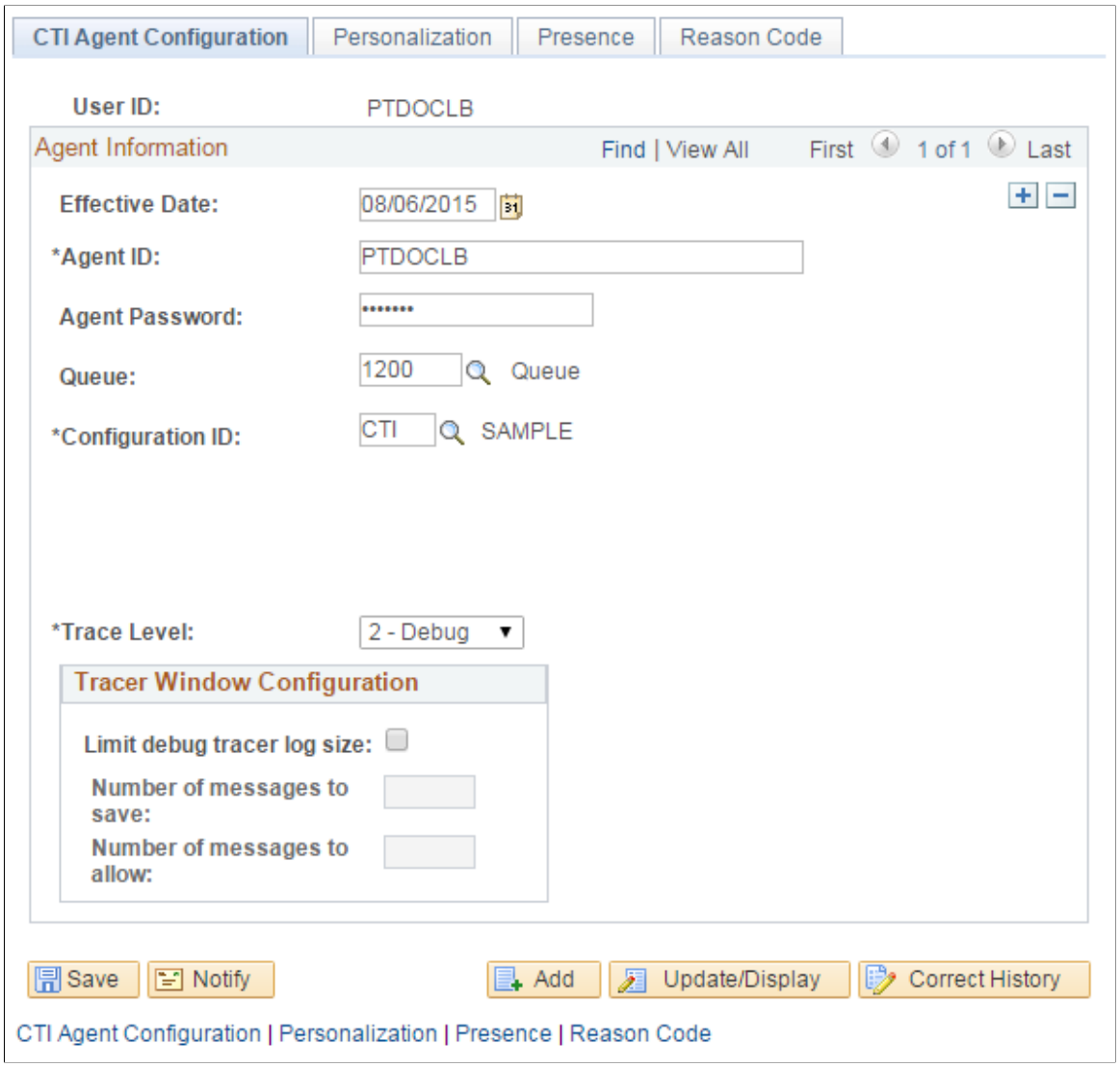
See Defining PeopleSoft MCF Queues for a Third Party.

Configuring CTI Agents

Access the CTI Agent Configuration page using the following navigation path:

PeopleTools > MultiChannel Framework > CTI Configuration > Agent > CTI Agent Configuration

This example illustrates the fields and controls on the CTI Agent Configuration page. You can find definitions for the fields and controls later on this page.



Configuring CTI agents involves the CTI Agent Configuration page, the Phone Book page, and the Personalization page.

Field or Control	*Description*
User ID	Displays the PeopleSoft user ID of the agent.

Agent Information

Field or Control	*Description*
Effective Date	Enter the date on which the current configuration should become active.

Field or Control	*Description*
Agent ID	Enter a user ID for the agent within the switch.
Agent Password	Enter the password that the agent uses to sign in to the phone, if any.
Queue	Enter the name of the queue that you want to assign to an agent. Use the Queue Configuration page to associate a queue with a directory number identifying an ACD group.
Configuration ID	Select the name of the configuration that you want to associate with the agent. The configuration ID is the name of the configuration that you created using the CTI Configuration page.
Application User Name/Password	These fields appear only if they are relevant to your setup. Enter the CTI user name and password of the agent.
CTI Client Signature	**Note:** This option appears only for Cisco configurations. This control shows the signature of a particular agent. The signature uniquely identifies an agent if you have implemented call monitoring. Typically, this value appears as an email address, such as john.doe@example.com.

Field or Control	*Description*
Trace Level	Options are: • *0 - None:* Disables tracing. • *1 - Info* (informational): Traces agent actions, such as dialing out, transfers, and so on. • *2 - Debug:* Used to troubleshoot crashes and other major errors. If you ever need to open a PeopleSoft GSC case about a CTI issue, include a level 2 - Debug trace. If a value other than *0* is selected, a tracer window appears to display activities and events on the chat or MultiChannel Console for debugging purposes. If you are using the JSMCAPI console, two tracers (a JSMCAPI tracer and a console tracer) appear if trace level 2, Debug, is selected. **Note:** Setting the trace level to *2 - Debug* can degrade performance. Unless you are troubleshooting the system, set the trace level to *None.*
Limit debug tracer log size	This check box is enabled when the value entered for **Trace Level** is not *0.* Select this check box to enable the agent to clear the tracer log based on **Number of log messages to save when cleared** and **Maximum number of log messages to allow.** If the check box is cleared, the tracer log will not get cleared and the **Number of log messages to save when cleared** and **Maximum number of log messages to allow**fields will be disabled.
Number of log messages to save when cleared	Specify the minimum number of recent tracer log messages that should be maintained in the tracer window.
Maximum number of log messages to allow	Specify the maximum number of tracer log messages that will be maintained in the tracer window.

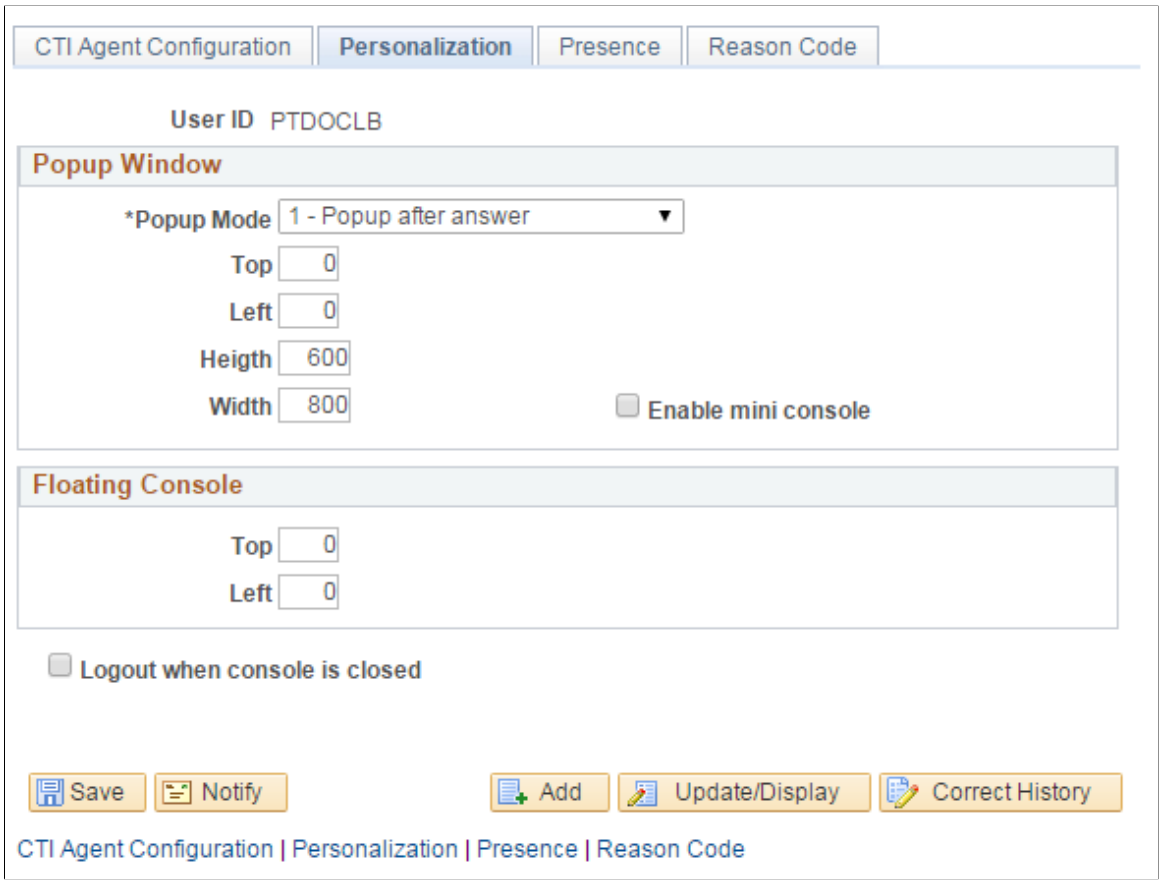
See Creating PeopleSoft MCF Agents for a Third Party.

Personalizing the Agent Console

Access the Personalization page using the following navigation path:

PeopleTools > MultiChannel Framework > CTI Configuration > Configure CTI Agents > Personalization

This example illustrates the fields and controls on the Personalization page. You can find definitions for the fields and controls later on this page.



Use this page to personalize timing, size, and position of the pop-up window on the desktop as well as the position of the floating console.

Popup Window

Field or Control	*Description*
Popup Mode	Enables you to configure when the pop-up window appears. You can have it appear after the call is answered, or you can have it appear when a call comes in. If it appears before the call is answered, the agent can determine whether she or he wants to answer the call based on the information that appears in the pop-up window. The default pop-up mode is *1- Popup after answer.* In some cases, for example accepting a transfer, although you set the value as *0- Popup when incoming,* you still get the pop-up window after you answer. This is because no call data was attached by the CTI vendor to the incoming event, and consequently the PeopleSoft CTI Console couldn't build the URL for the pop-up window. For example, assume party A is transferring a call to party B. Party B gets the incoming event, but party B doesn’t get a pop-up window. This is because the console did not receive the user data to generate the URL despite the fact that the mode is set to *0.* But after B answers this call, A completes the transfer. Then, B gets the event *partychanged,* receives the user data, and generates the URL and pop-up window.
Top	Enter the top position, in pixels. This value is relative to the top of the screen.
Left	Enter the left position, in pixels. This value is relative to the left side of the screen.
Height	Enter the height of the window, in pixels. The minimum value is 100.
Width	Enter the width of the window, in pixels. The minimum value is 100.
Logout when console is closed	This setting depends on CTI vendor support and may not be supported by all vendors. For JSMCAPI consoles with vendor support, select this setting to automatically sign out the agent when the CTI Console is closed.

Field or Control	*Description*
Enable mini console	Select to enable the administrator or user to configure the presence of the CTI mini console in the pop-up window. If this check box is not selected, the mini console does not appear in the popup window. **Note:** The mini console has performance overhead.

Floating Console

Field or Control	*Description*
Top	Enter the top position in pixels. This value is relative to the top of the screen.
Left	Enter the left position in pixels. This value is relative to the left side of the screen.

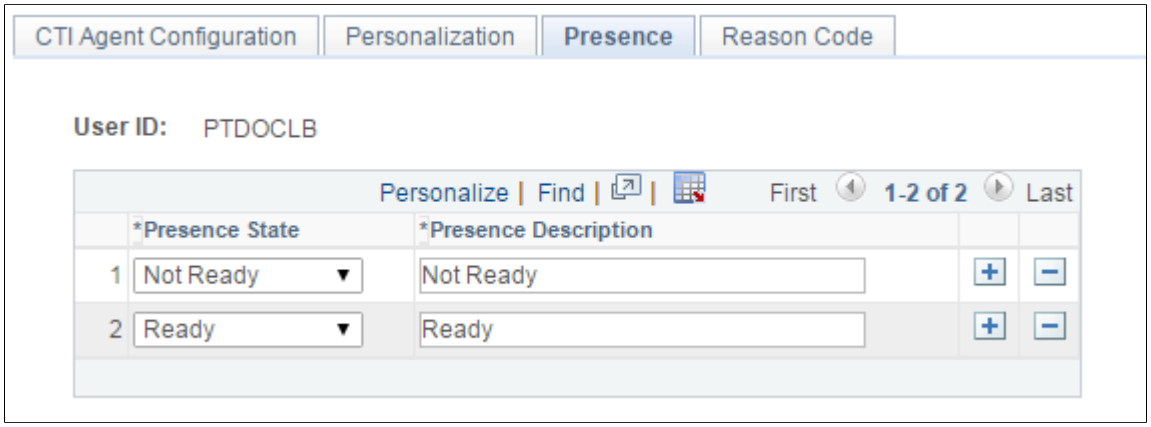
Entering the Presence State

Access the Presence page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **CTI Configuration** > **Agent**.

Search for the user id. The **CTI Agent Configuration** page displays. Select the **Presence** tab.

This example illustrates the fields and controls of the Presence page where you can uodate the presence status.



Field or Control	*Description*
Presence State	Select from *Not Ready* and *Ready*.

Field or Control	*Description*
Presence Description	Enter a description for the state.

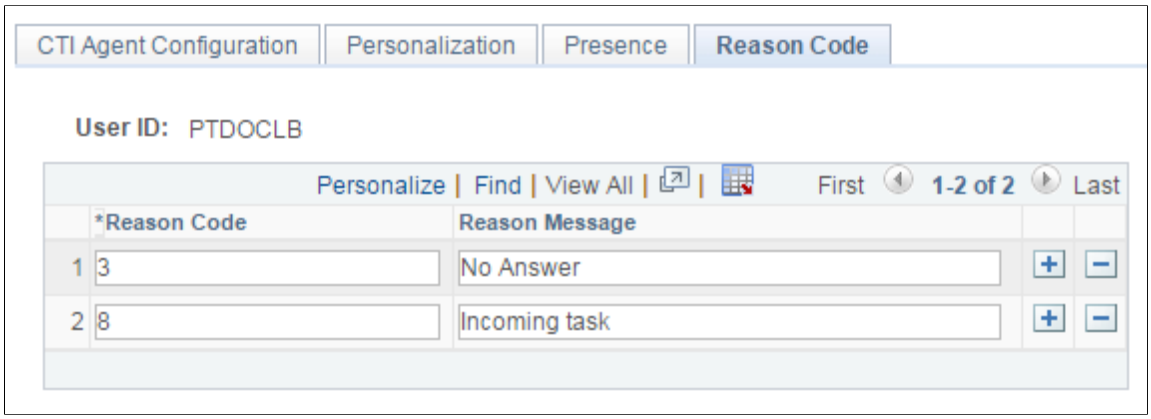
Using the Reason Code Page

Access the Reason Code page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **CTI Configuration** > **Agent**.

Search the user id. The **CTI Agent Configuration** page displays. Select **Reason Code** tab.

This example illustrates the fields and controls on the Reason Code page.



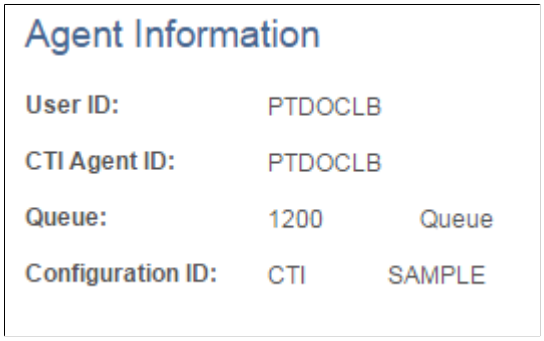
This page is used by the third-party vendors to define their customized codes.

Viewing Information About the Agent Information Page

Access the Agent Information page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **CTI Configuration** > **View CTI Agent Information**

This example illustrates the fields and controls on the Agent Information page. You can find definitions for the fields and controls later on this page.



The Agent Information page is a read-only page that displays the following information about a CTI agent.

Field or Control	*Description*
User ID	Displays the agent's PeopleSoft user ID.
Agent ID	Displays the agent's CTI middleware ID.
Queue	Displays the queue to which an agent is assigned.
Configuration ID	Displays the configuration ID associated with the agent.

Using Other PeopleSoft CTI Options

You can configure other optional CTI parameters.

Configuring Pop-Up Windows

PeopleSoft CTI can launch and populate transaction pages in the following ways:

- **Default URL.**

  The same URL is used for all calls. The Automatic Number Identification (ANI) is passed in as a parameter to that URL. The ANI identifies the telephone number from which the incoming call originated, and the number may be useful in determining what application data to retrieve. For example, it may be the home phone number of a customer.

- **Build the URL from attached Call Data.**

  PeopleSoft CTI formats a URL for the browser with a specific target PeopleSoft menu, market, and component. However, this method cannot be used if you are accessing multiple databases through the PeopleSoft Portal.

- **iScripts.**

  PeopleSoft CTI opens the transaction page using an iScript. This method must be used if you are accessing multiple databases through the PeopleSoft Portal. The iScript communicates with the targeted database to populate the appropriate transaction page with the caller's data.

Note: The call ID is passed in the URL string as a variable named *callID*.

Note: Browser settings and add-on tools that block pop-up windows can prevent CTI inbound call pop-up windows from appearing. If possible, configure such tools to allow pop-up windows from your site or domain.

Related Links

“Understanding Internet Script Classes” (PeopleCode API Reference)

Supporting Single Sign-In

In the applet-based solution, PeopleSoft CTI offers single sign-in. The console connects to the CTI middleware using the CTI user ID and password retrieved from the PeopleSoft database.

Related Links

"Understanding Single Signon" (Security Administration)

Logging CTI Events

CTI events can be logged in the CTI Event Log. You can view the logged CTI events by navigating to **PeopleTools** > **MultiChannel Framework** > **Universal Queue** > **Administration** > **CTI Event Log.**

To implement CTI event logging, navigate to **PeopleTools** > **MultiChannel Framework** > **Universal Queue** > **Configuration** > **Tune MCF Clusters** and set the key *log_cti* to *yes.* Use the Notify Cluster page to notify the affected queue to refresh logging parameters.

The CTI event log logs all events and requests related to a session, user, or call connection. It also logs all events related to call and group information.

Related Links

Tuning Cluster Parameters

Implementing Free Seating

When users sign in, they do not need to reenter telephone extensions and other user information if they have used the workstation before and the relevant information for the telephone associated with that workstation has not changed.

The PeopleSoft system enables free seating by maintaining a cookie on the workstation.

Using the PeopleSoft CTI Sample Pages

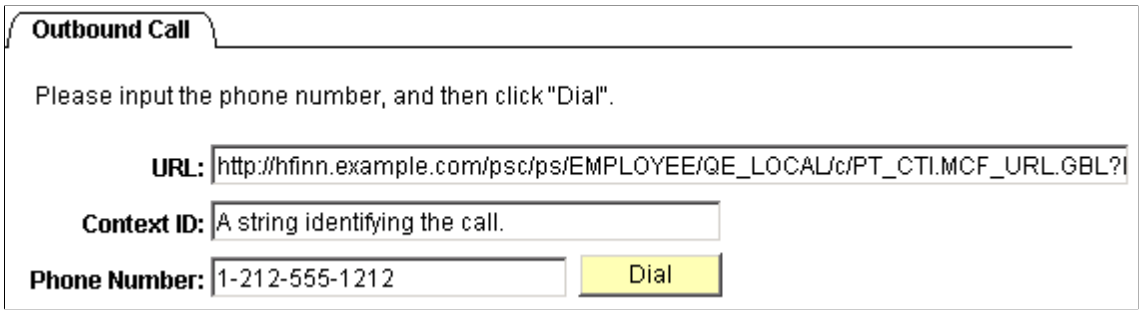
To demonstrate an outbound call, use the Sample Pages component (PT_CTI_DEMOOUTB). The CTI sample pages are intended for demonstration purposes only and should not be used in production.

Using the Outbound Call Page

Access the Outbound Call page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **CTI Configuration** > **Sample CTI Pages** > **Outbound Call**

This example illustrates the fields and controls on the Outbound Call page. You can find definitions for the fields and controls later on this page.



The Outbound Call page is an example of how you can customize an application page to enable a user to direct the CTI Console to dial a telephone number displayed on that page. The outbound calling demonstration works only when the CTI Console is enabled and the user has registered with the CTI vendor.

Field or Control	*Description*
URL	Enter the URL of a page to display when dialing out.
Context ID	Enter a string to attach to the call as outbound context.
Phone Number	Enter the telephone number that you want to dial. This field accepts numeric digits only. Do not enter special characters, such as **-** (hyphen), **.** (period), or other separators.
Dial	Click to dial the telephone number that you entered.

Related Links

Using the CTI Sample Console

Chapter 4

Configuring REN Servers

Understanding REN Servers

This section discusses REN servers.

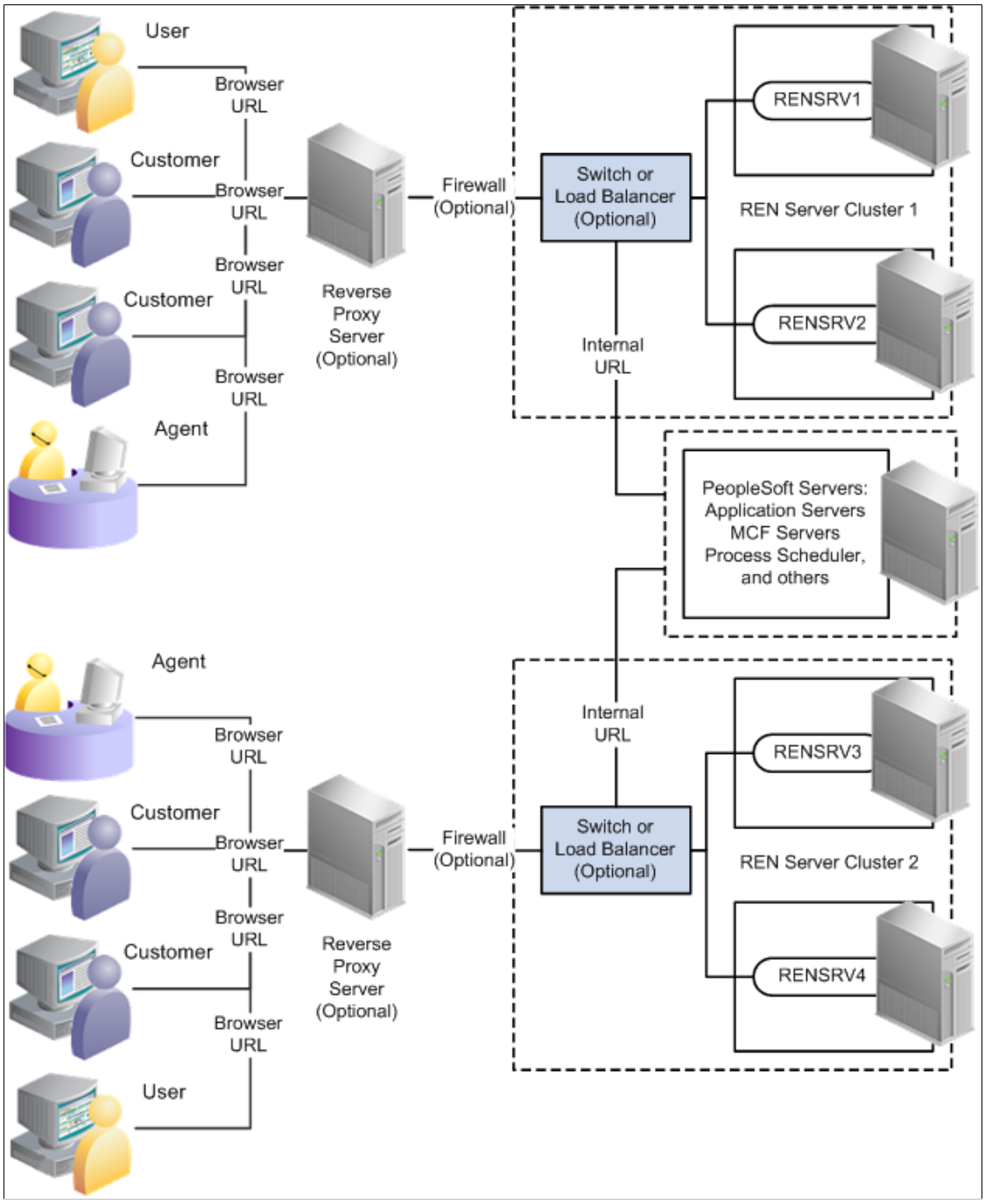
The REN server, an application server domain process, is essential to PeopleSoft MultiChannel Framework (MCF) architecture. MCF events are sent to REN servers, which deliver them to recipients of those topics.

REN servers are also used by other PeopleSoft applications to push event notifications to users, such as the Reporting Window output option and the Optimization Progress Window.

The REN server is a modified web server using the HTTP 1.0 or 1.1 communications protocol. Communication with MCF server processes and browser windows is bidirectional because they maintain persistent connections to the REN server. Events can be sent proactively to browser windows without polling or page refreshes.

REN Server Failover, Scalability, and Security Configuration

REN servers can be configured to support both failover and scalability, and should be protected with firewalls and appropriate security measures, as illustrated in the following diagram.



REN Server Failover

Although the REN server is integrated into an application server domain, it is not a standard PeopleTools server process (it has no database connection) and therefore has a separate failover mechanism. Two scenarios exist for failure recovery:

- For a standalone REN server, Oracle Tuxedo restarts the server if it fails.

  MCF servers and consoles reconnect to the REN server. However, any active browser sessions (such as MCF chat) are interrupted until a connection can be reestablished between the chat console and the restarted REN server.

- For clustered REN servers, each REN server in the cluster is a peer that mirrors the current state.

  This configuration has two advantages over a standalone REN server:

  - Clustered REN servers guard against hardware failure (provided that the clustered REN servers are on different host machines).
  - Active browser sessions are not lost.

REN Server Clusters

You can configure a REN server cluster with only one REN server member. However, a REN server cluster that is configured with two or more REN servers provides failover.

All REN servers in a cluster mirror each other and appear to external processes as a single URL. The REN server cluster must have an HTTP load balancer or switch as its front end. All connections with browsers and application server processes address the front end's URL. The load balancer should use an active standby content-switching rule to route all traffic to a designated REN server in the cluster. The front end selects an alternate member of the cluster only when the designated REN server fails to respond.

The REN server cluster maintains mirrored state in all members by relaying events with HTTP messages. Increasing the number of servers within a REN server cluster therefore does not address scalability issues. Clustering REN servers does not improve performance and may increase processing overhead and internal network traffic. The internal HTTP connections between cluster members should be high speed for best performance. Because of the overhead involved in synchronous cluster members, each member of a cluster can handle less load than a REN server in a cluster with only one REN server.

Note: In an environment in which multiple REN servers exist within a single cluster, the primary REN server sends synchronization data to the other members of the cluster. If any of these synchronization messages fail, then the primary REN server retries up to cluster_retry_count times. The minimum value of this parameter, cluster_retry_count, in psrenconfig.txt is 0, which means that the REN server does not retry.

If a REN server crashes, it does not rejoin the cluster because it would not be synchronized with the other clustered REN servers. The entire cluster must be shut down and rebooted to restore all members back to full participation.

Incoming cluster requests must eventually route to the front end's HTTP address. Queue servers and application servers use the cluster URL, which is typically set to be the URL of the front end. Browser clients make requests using the browser URL, which may be set to the front end, or to a server that proxies to the load balancer. If browser transactions are encrypted with SSL, then the browser URL is an HTTPS address to a reverse proxy server or SSL accelerator.

Note: If you use SSL between the browser and REN server, then you must use a reverse proxy server or SSL accelerator, unless you have configured an SSL-enabled REN server.

Note: When clustering multiple REN servers, typically there is some performance degradation.

Note: Periodically, Javsacript-based clients to the REN server, such as our MCF and CTI consoles, will perform a clean up cycle. This clean up cycle, called a "tunnel refresh" will cause the client to disconnect temporarily from the REN server. Depending on the implementation of the client, this may cause a message to be presented to the user, and reconnection may or may not be automatic.

Understanding SSL-Enabled REN Servers

You can enable a secure channel of communication between the clients and the REN server by enabling SSL on the REN server using openssl. The SSL protocol runs above Transmission Control Protocol/ Internet Protocol (TCP/IP) and below higher-level protocols, such as HTTP and IMAP4. By using TCP/ IP on behalf of higher-level protocols, openssl allows an SSL-enabled server to authenticate itself to an SSL-enabled client, a client to authenticate itself to a server, and both machines to establish an encrypted connection.

Installing Digital Certificates

REN servers require digital certificates to work in SSL mode. The servers pick up the certificates from the keystore in the PeopleTools database or an external keystore. The certificates must be imported into PeopleTools database from **PeopleTools** > **Security** > **Security Objects** > **Manage Digital Certificates.** Certificates that are installed in the database will have a unique combination of certificate type and alias.

The certificate type that is used for the server should be of the type CERT, and the alias is <machine name>.<domain name>. When the certificate is configured with a unique alias name, it should be associated with the REN server that is SSL-enabled. The REN server loads its server certificate from the database at the start-up.

See Installing Digital Certificates.

Authenticating Server and Client

For server authentication, the server sends its certificate to the client as a part of the SSL handshake and the client authenticates by verifying the Certificate Authority (CA) of the certificate against its trusted keystore. When the REN server is configured for SSL, all clients must trust the CA of the server certificate to participate in a successful communication. The keystore can either be in the database or an external keystore.

Client authentication verifies the clients's authenticity to participate in a communication with the server. When the REN server is configured for client authentication, all clients must supply a valid client certificate to participate in a successful communication.

All clients must use the REN cluster's HTTPS URL to communicate in the SSL mode. If the REN server is SSL only, access is denied to any client trying to communicate with a HTTP URL port. The browser-based clients, the application server client, and the REN Java clients should be configured appropriately to communicate with an SSL-enabled REN server.

Performance and Scalability for SSL-Enabled REN Servers

During an SSL transaction, the handshake is an added overhead that occurs. However, for every transaction, the handshake is done once to authenticate the server and the client. After authentication, the data is digitally signed, encrypted, and exchanged on an established session. For each console, authentication establishes a session only once, and no subsequent transactions inherit any overhead of authentication.

Configuring REN Server Security

This section provides an overview of REN server security configuration and discusses how to define permission lists for REN server access.

Understanding REN Server Security Configuration

Protect the REN server behind firewalls. A reverse proxy server can be used between browser clients and the REN server. Browser sessions can be SSL-encrypted by means of a reverse proxy server or hardware SSL accelerator.

Note: The security of your PeopleSoft system and configuration of load balancers, switches, and reverse proxy servers is beyond the scope of this document. Refer to your PeopleBooks for more information.

REN server access from browser clients is restricted to users who are currently signed in to PeopleSoft software with appropriate REN server permissions. You must enable single sign-in security to obtain REN server access. Permission to access REN server applications is granted on permission lists, which are in turn associated with security roles and user IDs. Clients lacking access permission receive a 403 Forbidden page from the REN server.

Note: REN server access requires that single sign-in be enabled.

Related Links

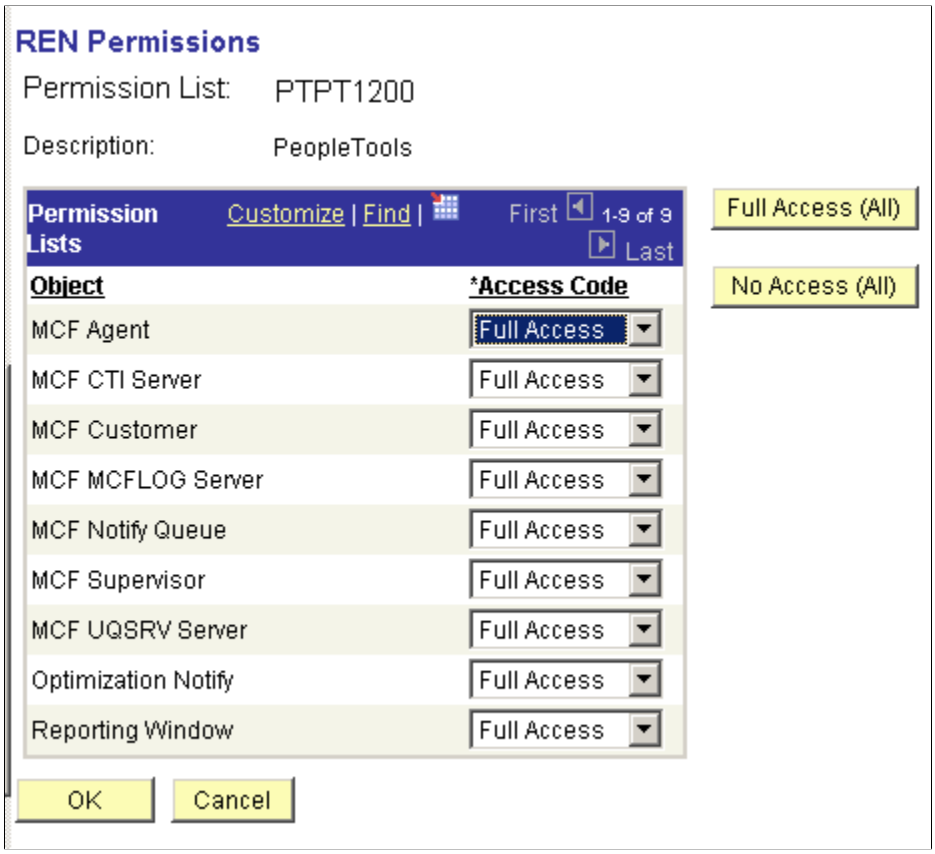
“Security Administration Overview” (Security Administration)

“System and Server Administration Overview” (System and Server Administration)

Defining Permission Lists for REN Server Access

The following REN Permissions page shows the objects and permissions that are defined for permission list PTPT1200. You can create custom permission lists and define access to REN servers.

This example illustrates the objects and permissions that are defined for a permission list on the REN Permissions page.



To define permission lists for REN server access:

1. Select **PeopleTools** > **Security** > **Permissions and Roles** > **Permission Lists.**
2. On the search page, search for and select your permission list.
3. On the Permission List page, select the PeopleTools tab.
4. Click **Realtime Event Notification Permissions.**
5. On the REN Permissions page, select your permissions.

   To enable REN server access for roles that are defined with the current permission list, select *Full Access* for each object that is required by the role. For example, users who require access to the MultiChannel Console must have Full Access defined for the MCF Agent object.

Note: To enable access to the Report-to-Window functionality, add WEBLIB_RPT to the Web Libraries page of the permission list, and set Reporting Window to Full Access on the REN Permissions page. Grant full access to the MCF CTI Server object only on the permission list that is assigned to the CTI server role. No other users should have MCF CTI Server access.
The user ID that is configured to start the Process Scheduler must have full access to the Reporting Window REN permission on at least one permission list for that user ID. If the user ID does not have full access to the Reporting Window, then the pop-up window stays in a status of queued.

Related Links

Required Security for PSMCAPI

Configuring REN Servers

To configure REN servers, use the REN Server (REN_SERVER_CMP) component.

This section provides an overview of REN server configuration options.

Understanding REN Server Configuration Options

Depending on your requirements, choose one of two REN server creation and configuration options:

- To create a single REN server in a particular database using default configuration parameters, create an application server domain using PSADMIN.

  Event Notification may be enabled in the quick-configure menu. An associated REN server cluster is automatically created when Event Notification is enabled.

- To create additional REN servers in a particular database, configure each REN server as required on the REN Server Definition and REN Server Cluster pages.

  Then create the associated application server domains. Event Notification must also be enabled in the quick-configure menu.

When a REN server starts, it looks for configuration information in the database, using the application server domain name and host name as keys. If the associated configuration information exists in the database, the REN server uses it. If no such configuration information exists, the REN server is configured by defaults, which also configure a REN server cluster for each REN server. You can change the default REN server configuration by using the REN Server Configuration pages, but such changes do not take effect until the REN server starts up again.

Note: You can create only one REN server per application server domain.

This section discusses some possible REN server configurations that depend on domain server topology.

Simple Configuration: Mycompany.com

In this configuration, the REN server is on the host machine MachA, the REN server uses the default port number 7180, the domain name server (DNS) addresses the host machine as MachA.mycompany.com, and no SSL or reverse proxy server is involved:

Parameter	*Value*
PeopleSoft Pure Internet Architecture Authentication Token Domain	mycompany.com
Authentication Domain in REN Server Cluster Configuration	mycompany.com

Parameter	*Value*
REN Server Cluster Root Path	/psren
REN Server Cluster URL	http://MachA:7180
REN Server Browser URL	http://MachA.mycompany.com:7180

Simple Configuration with SSL-Enabled REN Server: Mycompany.com

In this configuration, the REN server is on the host machine MachA, the REN server uses the default port number 7143, and DNS addresses the host machine as MachA.mycompany.com. The REN server is SSL-enabled.

Parameter	*Value*
PeopleSoft Pure Internet Architecture Authentication Token Domain	mycompany.com
Authentication Domain in REN Server Cluster Configuration	mycompany.com
REN Server Cluster Root Path	/psren
REN Server Cluster URL	https://MachA:7143
REN Server Browser URL	https://MachA.mycompany.com:7143

Reverse Proxy Server with Non-SSL Configuration

This configuration includes a single REN server and a reverse proxy server. The reverse proxy server could be either a dedicated reverse proxy server or a web server with a proxy plug-in configured to redirect both PeopleSoft Pure Internet Architecture and REN server requests. The application server host machine is MachA, and the REN server uses its default port 7180. The reverse proxy server is on MachRPS using port 8080 for HTTP. The DNS server must recognize MachRPS.mycompany.com.

Parameter	*Value*
PeopleSoft Pure Internet Architecture Authentication Token Domain	mycompany.com
Authentication Domain in REN Server Cluster Configuration	mycompany.com

Parameter	*Value*
REN Server Cluster Root Path	/psren
REN Server Cluster URL	http://MachA:7180
REN Server Cluster Browser URL	http://MachRPS.mycompany.com:8080

Reverse Proxy Server with SSL Configuration and Secure HTTP

For SSL, install certificates on the reverse proxy server, set the server to encrypt all communications, and use HTTPS URLs from the browser. In this example, the reverse proxy server uses port 8443 for SSL:

Parameter	*Value*
PeopleSoft Pure Internet Architecture Authentication Token Domain	mycompany.com
Authentication Domain in REN Server Cluster Configuration	mycompany.com
REN Server Cluster Root Path	/psren
REN Server Cluster URL	http://MachA:7180 **Note:** The cluster URL should not be a secure HTTP address if SSL is handled through a reverse proxy server.
REN Server Browser URL	https://MachRPS.mycompany.com:8443 **Note:** This is a secure HTTP address (HTTPS).

Note: If you use SSL between the browser and REN server, you must use a reverse proxy server or SSL accelerator.

Related Links

“Security Administration Overview” (Security Administration)

Configuring REN Servers and SSL-Enabled REN Servers

Specify REN server configuration parameters based on your network topology and server arrangement.

Define the parameters for REN server configuration in three locations:

- Authentication token domain, set during PeopleSoft Pure Internet Architecture installation or in web profile configuration.
- REN server configuration parameters, specified in an application server domain using PSADMIN.
- REN server parameters, including cluster and browser URLs, set in the PeopleTools REN Server and REN Cluster components.

  Configuration parameters that are set in the REN Server and REN Cluster components override any defaults in PSADMIN.

Authentication Domain

The authentication domain tells PeopleSoft Pure Internet Architecture the internet domain name that browser clients use when accessing PeopleSoft applications across the internet. The token is required to comply with the same-origin security policy that is enforced by most browsers. The domain name that is specified in the REN Server Configuration page must be identical to the domain name that is specified as the authentication token domain during PeopleSoft Pure Internet Architecture installation.

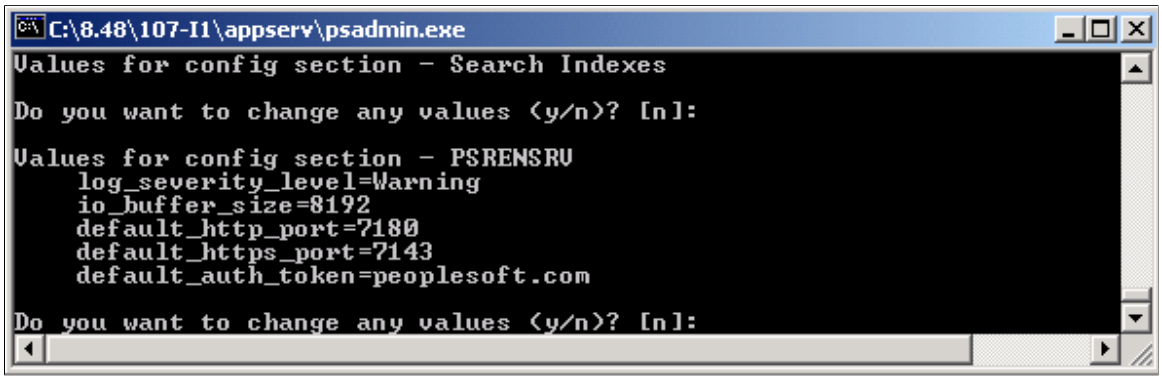
If authentication domain is not set during PeopleSoft Pure Internet Architecture installation, define the authentication domain in web profile configuration to match the REN server configuration.

Note: You must specify the authentication token domain if you access the REN server and the PeopleSoft Pure Internet Architecture web server using different DNS names from the browser client (for example, if they are on different machines).

Configuring a REN Server and SSL-Enabled REN Server with PSADMIN

If necessary, you can specify parameters in the PSRENSRV section of the PSADMIN application server domain configuration, as illustrated in the following example:

This example illustrates the parameters in the PSRENSRV section of the PSADMIN application server domain configuration.



Specify parameters as described in the following table:

Parameter	*Default*	*Description*
log_severity_level	Warning	This is the logging level for the REN server. Select from one of the following log severity levels, from less to more logged data: Error, Warning, Notice, Debug. **Note:** Do not use Debug in a production environment.
io_buffer_size	8192	This is the TCP buffer size in bytes that is used for serving content. Do not exceed a value of 65536. If the REN server is running on Microsoft Windows, change io_buffer_size to a minimum value of 56000.
default_http_port	7180	This is the REN server's HTTP port. The default value is *7180.* The default_http_port parameter takes effect only when a REN server starts up for the first time and the database does not already contain configuration information for the REN server. **Note:** After the HTTP port number that is assigned to the REN server has been established in the database, the only way to change it is on the REN Server Definition page. Editing the port number in the psappsrv.cfg file does not overwrite the value that is stored in the database.

Parameter	*Default*	*Description*
default_https_port	7143	This is the REN server's HTTPS port for SSL-enabled REN server. The default value is *7143*. **Note:** The https port is used only when the REN server is SSL-enabled. The default_https_port parameter is configured in psappsrv.cfg and is used when a SSL-enabled REN server starts up for the first time. **Note:** To change the default https port, use the REN Server Definition page. Changing the SSL Port requires the reboot of the REN server.
default_auth_token	example.com	This is the fully qualified domain name of the application server. This value should match the value of the web server's authentication domain. The default_auth_token parameter takes effect only when a REN server starts up for the first time and the database does not already contain configuration information for the REN server. When configuring the REN server parameters through PSADMIN, do not place a period or dot (.) before the default_auth_token value. For example, the parameter should read default_auth_token=example.com

Access logging is no longer enabled by default. In order to enable access logging, remove the leading #(hash) from this line in the “Modules to load” section of the psrenconfig.txt file.`#ns_param nslog ${bindir}/${prefix}rennslog${ext}`

The configuration for REN server and for any REN clients must be updated to restrict REN server communication to TLS version 1.3.

To update the REN server configuration, set the following parameters in the psrenconfig.txt file:

- ns_param ServerProtocols tlsv1_3
- ns_param SockServerProtocols tlsv1_3

- ns_param SockClientProtocols tlsv1_3

After specifying REN server configuration parameters, be sure to specify Y (Yes) when asked if you want event notification configured and MCF server configured. Boot this domain from the Domain Administration menu.

Note: Use PeopleSoft Pure Internet Architecture REN server definition and configuration pages to modify configuration parameters whenever possible. REN server configuration parameters that you make using PSADMIN are written to the psappsrv.cfg file in the application server directory. REN server configuration values that are found in the database override any values that are found in psappsrv.cfg. Use static IP addresses for your web servers. If you use dynamic IP addresses (DHCP), ensure that the domain name server (DNS) can map fully qualified domain names to the dynamic IP addresses.

Socket Binding

The REN server listens on the port that is defined in the REN Server Definition page, which is by default *7180.* However, the host name to which the REN server binds is determined by information in the psrenconfig.txt file for each application server domain. If the host machine contains multiple network interface cards (NICs), then the REN server binds by default to only one NIC, which is given by `uname()` on Unix, or `GetComputerName()` on Microsoft Windows.

To bind a REN server to a specific NIC, manually edit psrenconfig.txt for the appropriate application server domain, changing both set address and set hostname to the IP address and locally-known host name of the NIC. For example:

```
set address   192.0.2.10
set hostname  myserver.example.com
```

Note: If you enter an invalid IP address in the psreconfig.txt file, the REN server may not start correctly. Check the REN server log for error messages that identify the issue.

Note: To configure a REN server with a virtual machine name, manually enable the #ns_param javascript_kn_server parameter in the psrenconfig.txt by removing # (hash) from the parameter.

Configuring TCP_NODELAY

The parameter, TCP_NODELAY in psrenconfig.txt controls whether to disable the TCP Nagle algorithm on the TCP packets sent by the REN server. Two instances of TCP_NODELAY are available in psrenconfig.txt. TCP_NODELAY in the nssock section is used by non-SSL REN servers, and the instance in the nsopenssl section is used by SSL-enabled REN servers. The TCP Nagle algorithm is generally enabled by default and inserts a short delay before sending small TCP packets. This helps prevent network overload.

If TCP_NODELAY is set to 0, the TCP algorithm acts normally. This is the recommended configuration for most applications. However, for certain CTI applications, this parameter must be set to 1 to improve performance. If TCP_NODELAY is set to 1, the TCP Nagle algorithm is disabled on operating system platforms that support disabling this feature.

Configuring SSL Receive Parameters

In the REN server, the SSLReceiveTimeout and SSLReceiveWaitTime parameters are used to control the time to wait when there is a delay in receiving a TCP packet. To optimize the performance of REN

server while receiving data, set the following values for these parameters in the nsopenssl section of the psrenconfig.txt file:

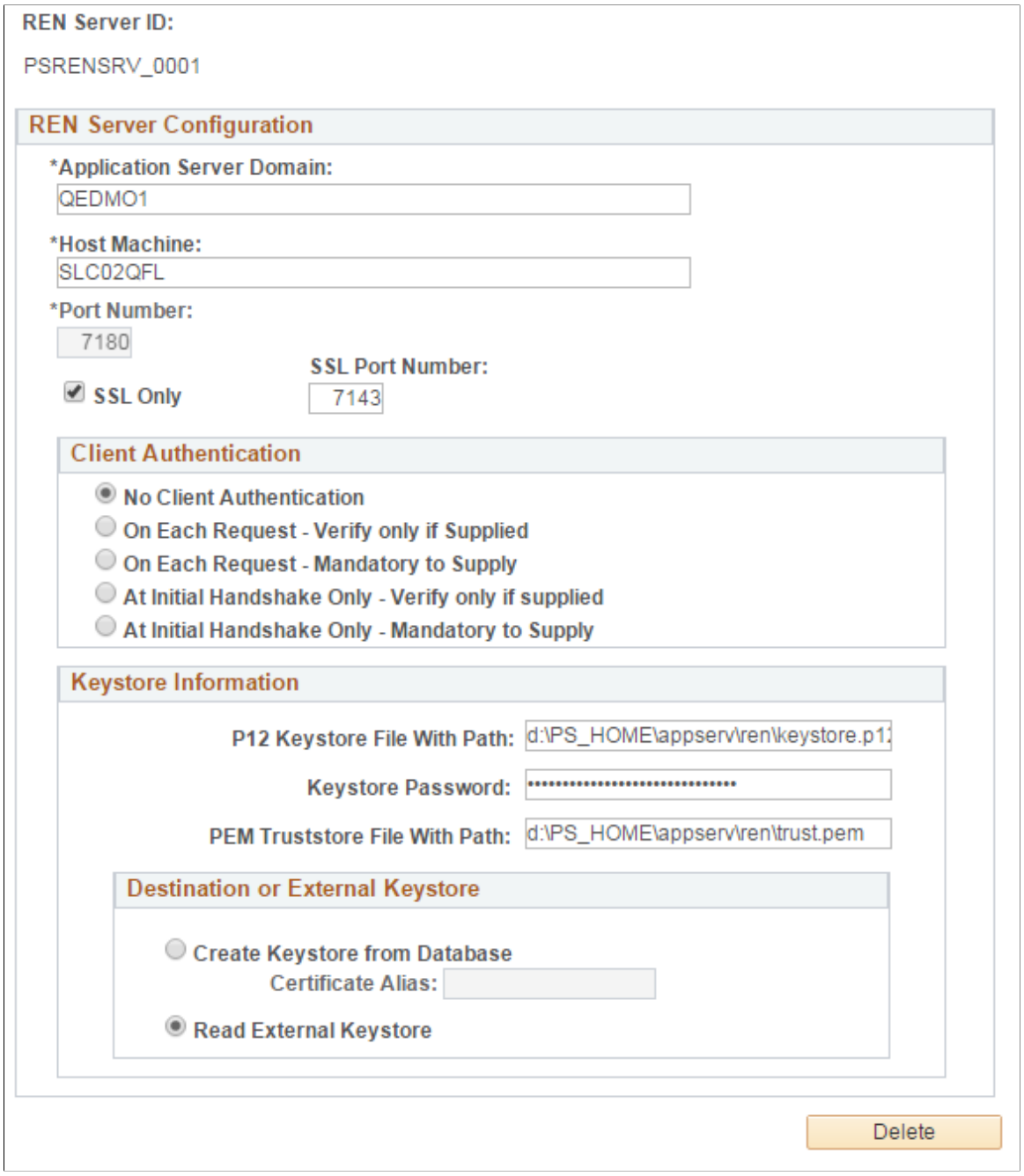
- ns_param SSLReceiveTimeout 300
- ns_param SSLReceiveWaitTime 10

Defining REN Servers

Access the REN Server Configuration page using the following navigation path:

PeopleTools > **REN Server Configuration** > **REN Server Definition**

This example illustrates the fields and controls on the REN Server Configuration page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Application Server Domain	Enter the application server domain that is serving this REN server.

Field or Control	*Description*
Host Machine	Enter the name of the host machine on which the specified application server domain runs. This entry requires the host machine name, not its DNS name. However, the host machine name may need to be fully qualified, for example, machineA.example.com. On a Unix machine, determine the host name by running `uname -a`. On a Microsoft Windows machine, determine the host name by running `hostname` at a command prompt.
Port Number	Enter the HTTP port number on which this REN server is addressed. Change the HTTP port value if multiple REN servers are running on the same host machine to avoid port conflicts.
SSL Only	Select to enable SSL on REN server. **Note:** If this option is selected, you must enter the SSL port. If you want to enable SSL on the REN server, ensure that you create and configure a Renserver certificate before defining a REN server.
SSL Port Number	Enter the HTTPS port number on which this SSL-enabled REN server is addressed.
Client Authentication	Select an option to determine the level of client authentication. **Note:** If the browser is configured for client authentication pop-up or the browser has more than one certificate configured, the SSL session ends if the user fails to provide the certificate within three heartbeats. To avoid such a session time-out, the user must either accept the client certificate within a heartbeat or increase the session time-out value in psrenconfig.txt. The Client Authentication values are described in a subsequent table.
P12 Keystore File With Path	Enter the absolute path where the PKCS #12 keystore will be created or is located.
Keystore Password	Enter a password for the internal or the external PKCS #12 keystore. The password is internally encrypted.
PEM TrustStore File With Path	Enter the absolute path where the TrustStore will be created or is located. **Note:** The trusted certificates are the root certificates, which are different from the certificate chain.

Field or Control	*Description*
Create Keystore from Database	Select to use stored certificates from the Digital Certificates page.
Certificate Alias	Select a certificate alias to be used as a server certificate by the SSL-enabled REN server. **Note:** The certificate alias is stored in the PSKEYDB, PSCERTDB, and PSREN records.
Read External Keystore	Select to use an external PKCS #12 keystore to read the Private Key, the Certificate Chain, and the Trusted Certificates. If you want to use an external Java keystore, you must configure the external Java keystore before you define the REN server on this page. See Configuring External Keystore in REN Server **Note:** By default, the **Read External Keystore** radio button is not selected, that is, the application server, at boot up, reads the Private Key, the Certificate Chain, and the Trusted Certificates from the database and creates a PKCS #12 keystore at the location entered in the **P12 Keystore File With Path** field. **Note:** When you select the **Read External Keystore** radio button, the application server, at boot up, streams the path and password of the external PKCS #12 keystore to the REN server. The path of the external keystore is entered in the **P12 Keystore File With Path** field and the password is entered in the **Keystore Password** field. Subsequently, the REN server, at boot up, loads the external PKCS #12 keystore by using the password.

The following table shows the client authentication values:

Parameter	*Flag Value*	*Description*
No Client Authentication	0	Client authentication is disabled.
On each Request-Verify only if Supplied	1	Client authentication is enabled The server sends a client certificate request to the client. Verification happens only if the certificate is provided. If the verification process fails, the TLS/SSL handshake is immediately terminated. If the client does not return any certificate, SSL communication still continues

Parameter	*Flag Value*	*Description*
On Each Request-Mandatory to Supply	3	Client authentication is enabled and mandates that the client provide the certificate. If the client does not return a certificate, the TLS/SSL handshake is immediately terminated with a handshake failure alert. If the client returns a certificate, it is verified. The communication fails if the verification fails.
At Initial handshake Only-Verify only if Supplied	5	Client authentication is enabled and requests a client certificate on the initial TLS/SSL handshake only. Verification happens only if the certificate is provided. If the client does not provide any certificate, SSL communication still continues. If verification fails, the TLS/SSL handshake is immediately terminated.
At Initial handshake Only-Mandatory to Supply	7	Client authentication is enabled and mandates that the client provide the certificate only in initial TLS/SSL handshake.

Configuring REN Server and SSL-Enabled REN Server Clusters

A cluster is typically a collection of REN servers among which the session information is replicated. You cannot add both SSL and non-SSL servers in a single cluster.

To configure REN server clusters, use the REN Cluster (REN_CLUSTER_CMP) component.

REN server clusters address failover and scalability.

Defining a REN Server Cluster

A REN server serves requests only if it is a part of the cluster. If the REN server is SSL-enabled:

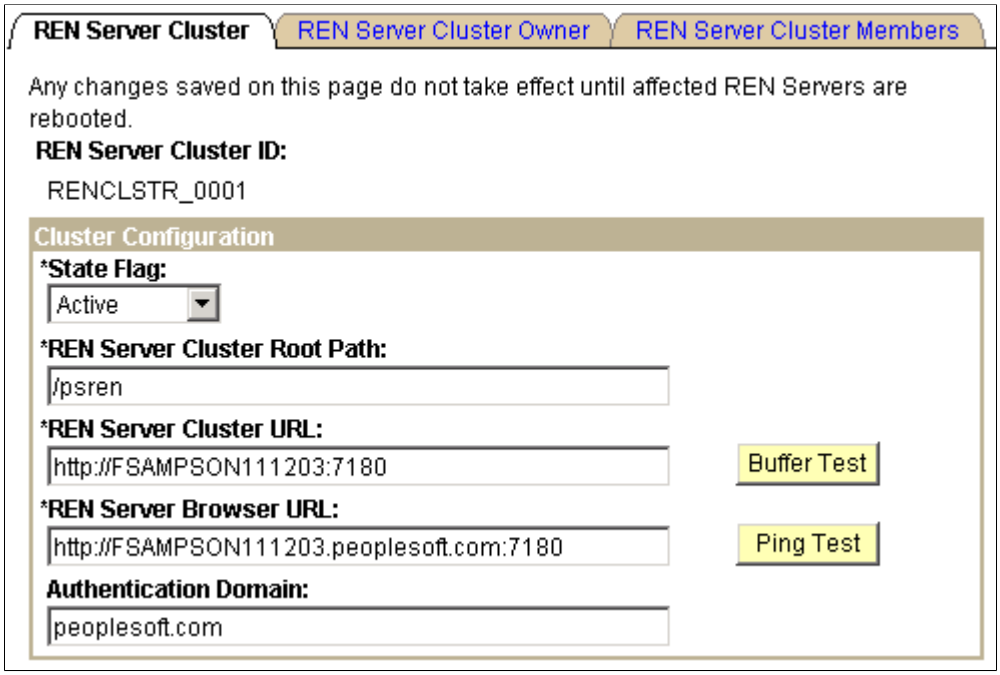
- All the member REN servers must be SSL-enabled REN servers.
- All the member REN servers must use the same server certificate.
- Both **REN Server Cluster URL** and **REN Server Browser URL** must start with HTTPS and use the HTTPS port.

Note: When the administrator changes the REN server to be in SSL mode, he or she must also ensure that the REN server is a member of SSL clusters only. In any given REN cluster, all REN servers that are members must be either SSL-only servers or non–SSL servers. For SSL-enabled REN servers, use SSL-enabled PeopleSoft Pure Internet Architecture.

Access the REN Server Cluster page using the following navigation path:

PeopleTools > **REN Server Configuration** > **REN Server Cluster** > **REN Server Cluster**

This example illustrates the fields and controls on the REN Server Cluster page. You can find definitions for the fields and controls later on this page.



By default, if you start a REN server from PSADMIN without configuring a REN server cluster, a cluster is created with a cluster ID RENCLSTR_000n

Field or Control	*Description*
State Flag	Select *Active* or *Inactive.* This field determines whether the cluster can receive new client requests. For scalability, configure multiple REN server clusters with the same ownership and set them to active status. Then the reporting window and customer chat applications will direct new client requests to a randomly chosen active REN server cluster. If all clusters are inactive, the client receives an error message. If the cluster supports MCF servers, current chat sessions continue even after a cluster is inactive. But the MCF system does not route any additional requests to an inactive cluster. Inactivate a cluster before deleting the cluster, or before removing a member REN server from the cluster. You can inactivate a REN server cluster without deleting the cluster.
REN Server Cluster Root Path	The default REN server cluster root path is /psren. Change this as required so that multiple REN server clusters are addressable through a single reverse proxy server. Changes to the root path should also be reflected in the URL mapping of any reverse proxy server.

Field or Control	*Description*
REN Server Cluster URL	The REN server cluster URL is the address that is used to reach the REN server cluster internally. This is the URL that is used by internal processes. If the MCF cluster is served by a REN server cluster, the cluster URL is that of the switch or load balancer in front of the clustered REN servers. The cluster URL must be unique for each cluster. No two clusters can address the same cluster URL. Specify the cluster URL in the form <http://<DNS_machine_name>:<port>, where: • <DNS_machine_name> is the server machine name that is recognized by your DNS. • <port> is the REN server port number; the default value is 7180. This port number is the REN server port number or the port number of a proxy server, load balancer, or other front end. • The protocol must be HTTP if the REN server is non-SSL; for an SSL-enabled REN server, the protocol must be HTTPS. **Note:** Use the limit_http_responses parameter in the psrenconfig.txt configuration file to control the number of HTTP responses that will be sent to the browser following an HTTP POST. The default value of this parameter is *1*, meaning that each POST receives only one response. If limit _http_responses is set to *0*, the number of responses will not be limited, and each POST may receive more than 1 HTTP response from the REN server. Multiple responses can cause unpredictable browser, load balancer, and ping test behavior.
Buffer Test	Click **Buffer Test** to initiate a test of the REN servers' ability to break up and send a large file using multiple internal buffers. The buffer test bypasses REN server security, and does not depend on specified domain names (authentication domain), so you can use it to verify that the REN server is running on the network.

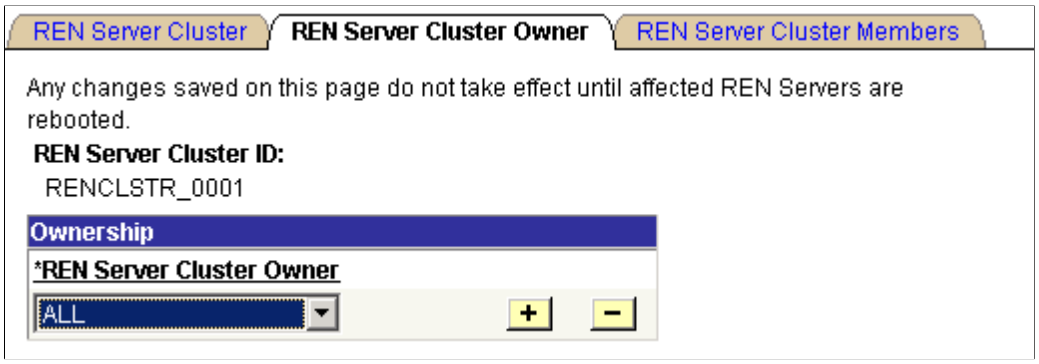
Field or Control	*Description*
REN Server Browser URL	The REN server browser URL is the address that is used by external clients and by agent chat to reach the application that is served by this REN server cluster. The browser URL may be different from the cluster URL, which should not have to go through any firewall, reverse proxy server, or other outward-facing security barrier. If the REN server is reached through a load balancer, switch, or reverse proxy server, specify the fully qualified URL of that device as accessed from the user's browser. The URL must be the address of the gateway machine (proxy server, load balancer, or SSL accelerator). Specify the address in the form http: or https://<DNS_machine_ name>.<domain_ name>:<port>, where: • <DNS_machine_name> is the server machine name that is recognized by your DNS. • <domain_name> is the fully qualified domain name that is recognized by your DNS. • <port> is the REN server port number; the default value is 7180. This port number is the REN server port number or the port number of a proxy server, load balancer, or other front end. **Note:** If the REN server is SSL-enabled, the browser URL must be HTTPS. Also, if you use an HTTPS address for PIA, you must use an HTTPS address for the browser URL.
Ping Test	Click to initiate a test of the REN server that is specified in the browser URL fields. Failure may indicate that a URL or authentication domain is incorrectly specified, the REN server is not running, or single sign-in is not implemented.
Authentication Domain	Enter the authentication domain. This must be the same as the authentication domain that is specified in the PeopleSoft Pure Internet Architecture installation or in the web profile configuration.

Specifying REN Server Ownership

Access the REN Server Cluster Owner page using the following navigation path:

PeopleTools > **REN Server Configuration** > **REN Server Cluster** > **REN Server Cluster Owner**

This example illustrates the fields and controls on the REN Server Cluster Owner page. You can find definitions for the fields and controls later on this page.



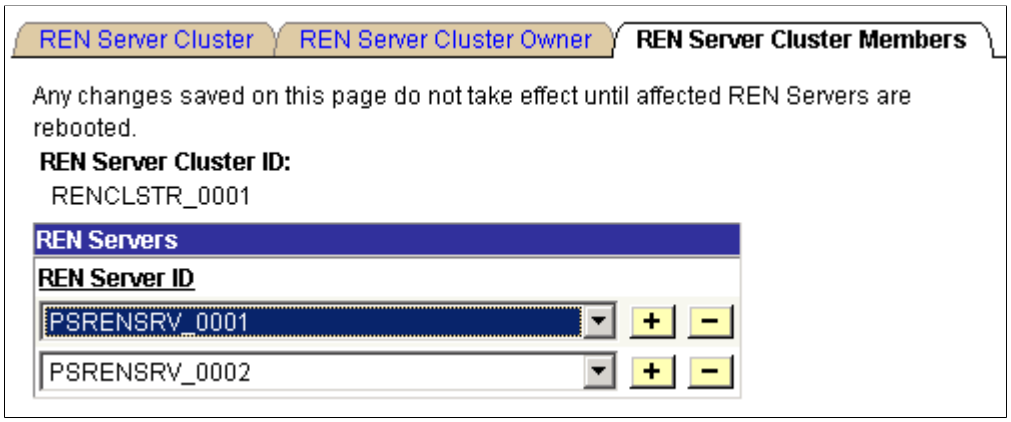
Field or Control	*Description*
REN Server Cluster Owner	Select the owner of this REN server cluster from the drop-down list box. Select from the following values: • *All* • *MCF* • *Optimization* • *Reporting* Specifying an owner for a REN server cluster limits client access to that cluster. This is useful to ensure performance under load. Specifying an owner for a REN server cluster also supports security. For example, an MCF cluster can be created only on a REN server cluster that is owned by MCF or ALL.

Specifying REN Server Cluster Members

Access the REN Server Cluster Members page using the following navigation path:

PeopleTools > REN Server Configuration > REN Server Cluster > REN Server Cluster Members

This example illustrates the fields and controls on the REN Server Cluster Members page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
REN Server ID	Select a REN server from the drop-down list box.

Each REN server can belong to only one REN server cluster.

Configuring a Reverse Proxy Server with a REN Server

This section provides an overview of reverse proxy server (RPS) configuration and provides examples.

Understanding RPS Configuration

Production PeopleSoft installations may configure the REN server behind an RPS. The RPS isolates the REN server and other web servers from the open internet, provides SSL session handling, and presents a single-server origin to outside clients. PeopleSoft customers may put REN servers and PeopleSoft Pure Internet Architecture web servers behind one RPS, or just REN servers.

These examples assume that:

- You have installed the current PeopleTools release on both host machines.
- You have configured a web server using the default parameters on the first host machine.
- You have configured a REN server using the default parameters on the first host machine.

See Understanding REN Server Configuration Options.

Example: Configuring a WebLogic RPS for a REN Server on Another Host Machine

This example presents one possible configuration for a REN server running on one host machine and installing an RPS to run on a second host machine, using Oracle WebLogic. The RPS redirects clients to both a REN server and to the PeopleSoft Pure Internet Architecture web server.

To configure an RPS for a REN server on another host machine:

1. Install a new web server domain on the second machine.

   Name the domain *rps*.

   Configure the following values:

   - AppServer Name: <*application_server_machine_name*>
   - JSL Port: *9999*

     The RPS will not make Jolt connections.
   - HTTP Port: *8080*
   - HTTPS Port: *8443*

2. Start the new web server.

   Navigate to *PS_CFG_HOME*\webserv\rps\bin, and run `startPIA.cmd`.

3. Access the Admin Console using either the desktop or hosted version of the WebLogic Remote Console.

   See “Using WebLogic Remote Console” (System and Server Administration).

4. Under Providers, select **Add Admin Server Connection Provider**, and supply the required connection information for the rps server.

   See the Oracle GitHub documentation, https://oracle.github.io/weblogic-remote-console/administration-server/domain-configuration.

5. Click the Edit Tree tile, and then select **Deployment** > **App Deployment** > **rps**.
6. Select the Targets tab, and use the arrows to move PIA from Chosen to Available.
7. Click Save.
8. Click the Edit Tree tile, and then select **Deployment** > **App Deployment** > **HttpProxyServlet**.
9. Select the Targets tab, and use the arrows to move PIA from Available to Chosen.
10. Click Save.
11. For better web server performance, in the Edit Tree section, select **Servers** > **PIA**.
12. Select the Protocols tab, select the HTTP tab, and set both **Duration** and **HTTPS Duration** to *120* secs.
13. Stop the *rps* web server.

    Navigate to *PS_CFG_HOME*\webserv\rps\bin and run `stopPIA.cmd`.

14. Configure RPS parameters for the *rps* server.

    Locate the file web.xml at *PS_CFG_HOME*/webserv/rps/applications/HttpProxyServlet/WEB-INF.

Edit web.xml in a text editor, changing the WebLogic port and WebLogic host from *8080* to *80* (the value *8080* is a default value that is derived during installation of the domain *rps*). For example:

```
<init-param>
 <param-name>WebLogicPort</param-name>
 <param-value>80</param-value>
 <description>HTTP listen port of WebLogic PIA/PORTAL server.</description>
</init-param>
```

To specify the associated REN server, (which is on another machine), edit web.xml, changing the REN server host machine, port, and root URL from their default RPS values. For example:

```
<init-param>
 <param-name>WebLogicHost</param-name>
 <param-value>MACHINE_2</param-value>
 <description>Hostname of REN server.</description>
</init-param>
<init-param>
 <param-name>WebLogicPort</param-name>
 <param-value>7180</param-value>
 <description>Listen port of REN server.</description>
</init-param>
```

Another example is:

```
<servlet-mapping>
 <servlet-name>RENHttpProxyServlet</servlet-name>
 <url-pattern>/psren/*</url-pattern>
</servlet-mapping>
```

15. Reboot the RPS web server.

    Navigate to *PS_CFG_HOME*\webserv\rps\bin, and run `startPIA.cmd`.

16. (Optional) Configure and enable SSL on the RPS machine.

Note: When using an Apache RPS, you must configure the kn_response_flush_override and the flush_rps_buffer_size_for_knjs parameters in the psrenconfig.txt file. Set both parameters according to the instructions within that file. Apache needs both parameters present with the same buffer size. The kn_response_flush_override parameter flushes a message, while the flush_rps_buffer_size_for_knjs parameter flushes the stay-alive.

Note: Using WebLogic as a reverse proxy server is not recommended for a production system.

Configuring Apache-based Reverse Proxy Servers for a REN Server

Apache-based proxy servers vary widely in configurations; here we present an example configuration. The configuration files for your environment may be quite different.

To proxy for RenServer, find and edit the httpd.conf configuration file. Make the following modifications to the file:

1. Move the line LoadModule proxy_module modules/ApacheProxyModule.dll to the bottom of the file.
2. Comment out the line AddModule mod_proxy.c.

3. Add the following five lines after LoadModule proxy_module:

```
<IfModule mod_proxy.c>
  ProxyRequests Off
  ProxyPass /psren http://machine:7180/psren
  ProxyPassReverse /psren http://machine:7180/psren
</IfModule>
```

4. Reboot your web server and reverse proxy server.

Chapter 5

Configuring PeopleSoft MCF Servers and Clusters

Understanding PeopleSoft MCF Server and Cluster Architecture

This section discusses PeopleSoft MCF server and cluster architecture.

PeopleSoft MCF Server Configuration

PeopleSoft MCF depends on processes that are configured and booted as part of an application server domain. Configure MCF processes (servers) through PSADMIN, along with other processes in each application server domain.

Note: The Real-time Event Notification (REN) server process can be used by applications that are separate from the queue server and MCF log processes. In this case, you can configure the application server domain for event notification without creating the MCF servers.

After considering performance and failover issues, the MCF system administrator provides configuration information that describes the arrangement of queues, domains, queue server processes, REN server processes, MCF processes, and URL addresses.

REN server processes are configured on PeopleSoft Pure Internet Architecture pages before or after being initiated in an application server domain. Each queue server process is uniquely identified in the system by the combination of the machine name, domain subdirectory name, and process identifier. MCF log processes use the same queue-server identification scheme.

Both the queue server and the MCF log server are REN Java clients. If the REN server is configured to accept only SSL connections, then you must configure SSL certificates for the queue server and MCF log server.

See Installing Digital Certificates.

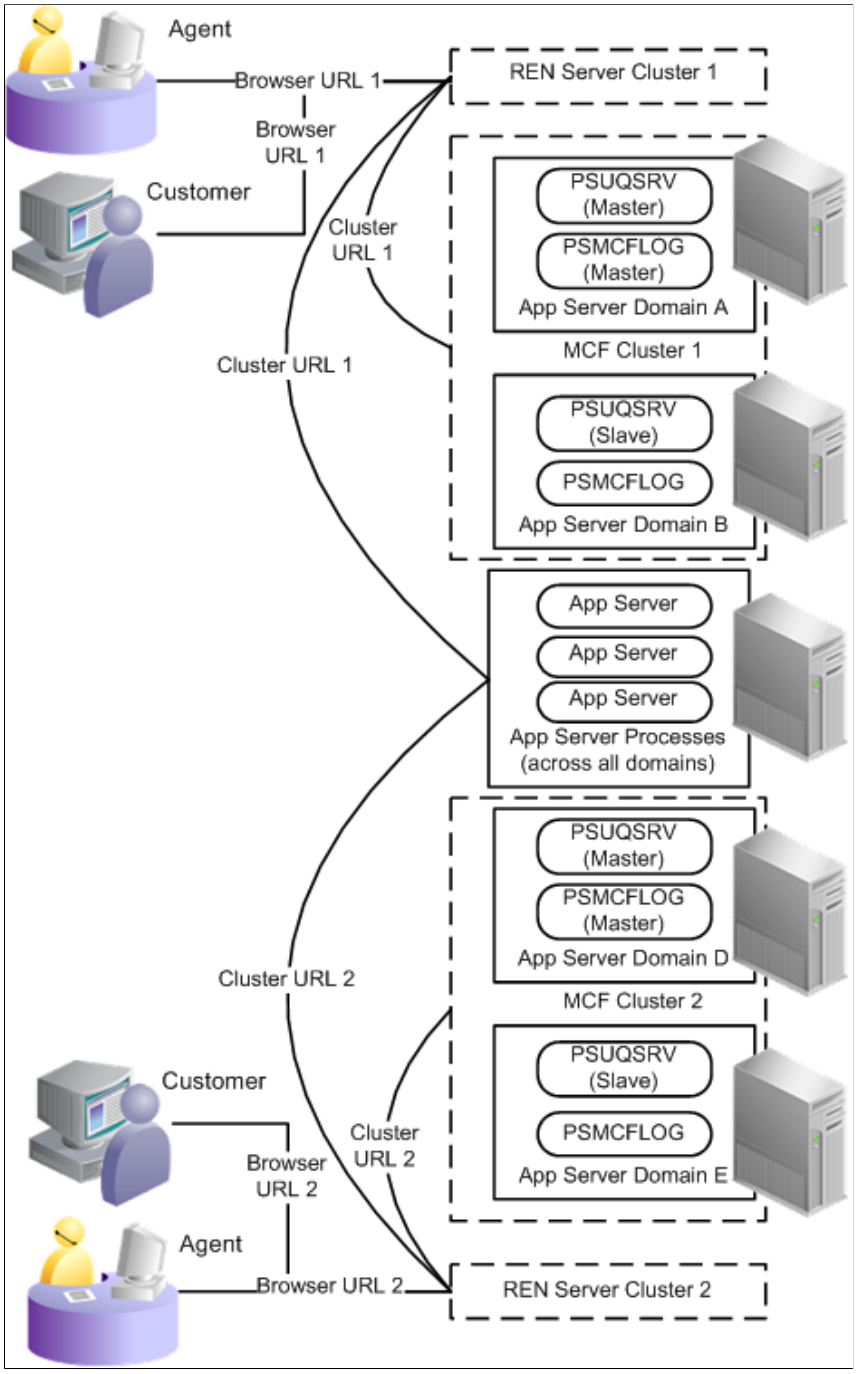
Related Links

Understanding REN Servers

PeopleSoft MCF Cluster Architecture

In a PeopleSoft system, all application server processes, including MCF servers and REN servers, belong to an application server domain. Each domain can have only one REN server process (PSRENSRV), one queue server process (PSUQSRV), and one MCF log server process (PSMCFLOG). Domains can be redundantly clustered to provide failover. Logical queues can be serviced by multiple clusters for scalability. Support for scalability and failover is integrated into the configuration process.

The following diagram illustrates MCF cluster architecture.



Queue Server and Queue Server Failover

The queue server is a server process in the PeopleSoft application domain that routes email, chat, and generic tasks to the agent based on the agent properties, such as state and skill set, and task routing properties, such as priority, language, and cost.

The queue server process (PSUQSRV) is a Tuxedo-managed server with a standard PeopleSoft database connection. Each queue server process is the central routing point for one or more physical queues. The

queue server maintains state information for work requests, work in progress, agent availability, and agent workload. Queue server state is written to database records, except for the assignment of chat to agents.

The queue server can recover from a crash because most of its state is written to the database. When a queue server reboots, it checks the database and loads state information for open work tasks. The queue server connects to its REN server and issues restart queries to each console so that it can rebuild agent assignment information, which may have changed while the queue server was down.

Although a single queue server can recover state after recovering from software failure, this does not guard against hardware failure. Multiple queue server processes running on multiple host machines and configured in a cluster provide failover for hardware failure. Unlike the REN server cluster, the clustered queue servers operate as one primary and many secondaries. The primary handles all routing decisions, while secondary processes monitor the primary and step in only if it fails. Any rebooted queue server rejoins the cluster in a secondary role. Any secondary that is promoted to primary loads state from the database and issues queries to consoles as if it were the only process in the cluster.

Each queue server process follows a fixed procedure to ensure that the cluster has at most one functioning primary. Database locks eliminate possible race conditions, and the primary periodically writes a timestamp to indicate its health. The masterinterval parameter controls the frequency at which the primary process must update the timestamp in the cluster table. The masterinterval parameter corresponds to the maximum time after a primary queue server fails before another queue server process takes over. Minimizing this value provides rapid failover response time but also requires frequent database updates.

See Tuning Cluster Parameters.

Each queue server must be part of an MCF cluster, and each MCF cluster must include at least one queue server. An MCF cluster of only one queue server provides no redundancy against hardware failure.

Create a queue server that starts when an application domain is started by selecting MCF servers from the quick-configure menu during application domain configuration.

In summary, configure queue servers to provide hardware failover. Each queue server is part of an MCF cluster. To support hardware failover, distribute primary and secondary queue servers over multiple hosts. Every queue server in an MCF cluster communicates with the same REN server ID. Therefore, REN server failover is also crucial.

The first queue server that places a valid primary entry for itself in the cluster table becomes the primary queue server. In most cases, the primary queue server is the first queue server started. No configuration parameter exists to designate primary or secondary queue server within a cluster.

After an MCF cluster's primary queue server is established, all other cluster members become secondary queue servers. If the secondary queue servers within a cluster detect a failure of the primary queue server, the remaining secondary servers compete to become the primary queue server. If the primary queue server reboots before a secondary takes over, the primary queue server also competes. No configuration parameter exists to designate priority among secondary servers.

See Understanding REN Servers.

Logical Queues and Physical Queues

PeopleSoft MultiChannel Framework enables the configuration of both logical and physical queues.

A logical queue is an application-level queue that receives work requests (tasks) relating to an application area, such as chat requests regarding sales information, and routes them to agents that are capable of

handling the work. For example, you might configure a logical queue called SALES for sales inquiries and another called SUPPORT for support issues.

Logical queues can be partitioned into physical queues for scalability. A physical queue is managed by a single MCF cluster. For scalability, the tasks that are enqueued on a logical queue are distributed by the framework among all available physical queues. For example, the SALES queue could be serviced by four MCF clusters across four physical queues: SALES1, SALES2, SALES3, and SALES4.

Each agent can be assigned to only one physical queue within each logical queue. Each agent can be assigned to multiple logical queues.

PeopleSoft MCF Log Server and Log Server Failover

The MCF log server (PSMCFLOG) is a Tuxedo-managed server that is similar to the queue server. Each MCF log server receives events that are sent by a REN server and is responsible for writing MCF events to the database.

The MCF log server logs events to PS_MCFUQEVENTLOG. By default, the log server does not log periodic state information broadcasts from the queue server to the MultiChannel Console. If you need to log these events, configure logging on the Cluster Tuning page. You can also configure the log server to log the contents of chat sessions. Chat session logging is deactivated by default. Logged chats are stored in PS_MCFCHATLOG.

If the MCF log server crashes, it resumes functioning immediately after restarting. When the first secondary log server detects a failed primary, it takes over as the primary log server for the cluster. The new primary log server again receives all base topics, but it does not log chat sessions that started or continued during the time that the original primary log server failed. The new primary log server does not log per-agent events for agents that were signed in at, or during, the time of the failure.

An MCF log server is created along with a queue server when you enable MCF servers during application server domain configuration. No specific log server configuration is available during domain configuration.

Queue Server Scalability

PeopleSoft MultiChannel Framework is scalable to support large-capacity call centers or other large organizations. The basic strategy is to divide the workload by spreading it over several MCF clusters. This is accomplished by creating multiple physical queues for each logical queue and spreading the management responsibility for each physical queue to separate queue server processes, preferably on multiple host machines. This technique should not be confused with failover protection, which also adds processes and machines. In failover, the added processes are clustered together and do not provide performance improvement.

Organize applications using PeopleSoft MultiChannel Framework around logical queues (for example, SALES queue and SUPPORT queue). Incoming work tasks are sent to a logical queue. PeopleSoft MultiChannel Framework then assigns the task to one of the corresponding physical queues. This assignment is random across the queues. The load across the servers is balanced by servicing only one physical queue per logical queue by single MCF cluster.

For example, a logical SUPPORT queue might be split into physical SUPPORT1 and SUPPORT2 queues such that work requests are randomly distributed between the two physical queues. Half the agents

receive from one queue and half from the other. This splits the workload evenly between the two queue server processes, while still presenting one logical SUPPORT queue to the application.

Recommended Configurations

Consider the following configuration options to ensure maximum reliability and scalability of your PeopleSoft MultiChannel Framework installation:

- Configure multiple MCF servers in a cluster across multiple host machines.

  This provides protection against single-point failures.

  Each MCF cluster requires a REN server cluster. Configuring multiple REN server clusters is functionally the same as configuring multiple MCF clusters for scalability. Inside a REN server cluster, configuring multiple REN servers is functionally the same as multiple queue servers for failover.

  See Configuring PeopleSoft MCF Clusters.

- Use REN server clusters only for failover.

  REN server clusters do not enhance performance.

- Split logical queues into more than one physical queue if more work is required on that queue than a single process or machine can handle.

- If an application server domain is likely to be restarted regularly for reasons that are not related to PeopleSoft MultiChannel Framework, configure PeopleSoft MultiChannel Framework in a separate domain.

  Regular restarting of MCF servers affects performance because the MCF servers must recover state when they are recycled or when a secondary takes over from a primary server.

Configuring PeopleSoft MCF Clusters

An MCF cluster is a group of multiple MCF-enabled application server domains in which all queue servers and log servers communicate with the same REN server cluster. Only one queue server and one log server in an MCF cluster are active at any one time. These are called the primaries. The rest are dormant and redundant, and are called the secondaries. If a primary drops out of the cluster for any reason, the secondary elect a new primary to take its place.

This section provides an overview of MCF cluster configuration and discusses how to configure MCF clusters.

Understanding PeopleSoft MCF Cluster Configuration

Each MCF cluster includes a minimum of one queue server and one log server communicating with one REN server. An MCF cluster is typically identified by the ID of the REN server cluster serving it. No configuration limit is placed on the maximum number of queue servers in an MCF cluster.

In MCF architecture, a chat client initiates contact with an agent through the designated external (browser) URL for a REN server. This URL can point to a load balancer, switch, reverse proxy server, or other hardware or software directing requests through a firewall. The external URL is also known as the browser URL because it supports the MCF browser windows (agent chat, customer chat, and MultiChannel Console).

For communication of queue servers, log servers, and application servers with REN servers, for example, handling email requests behind a firewall, you can use an internal URL. Specify both internal (cluster) and external (browser) REN server URLs during MCF cluster configuration.

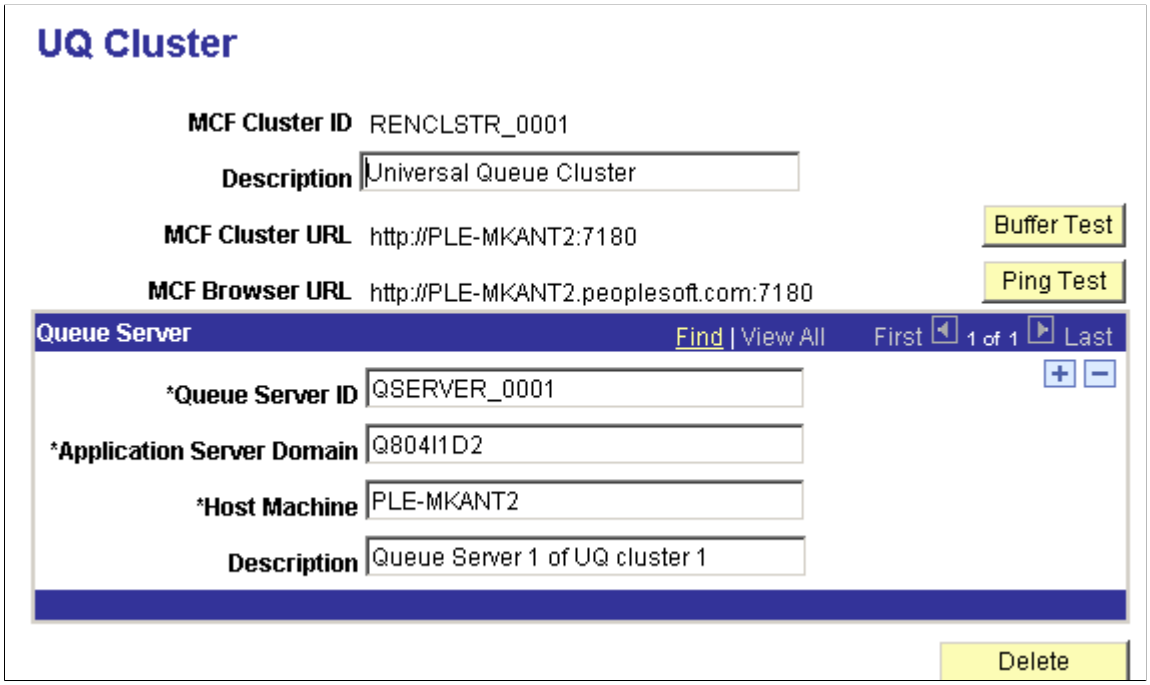
Note: If no security is implemented, the browser and cluster URLs may be the same.

Configuring PeopleSoft MCF Clusters

Access the UQ Cluster page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Configuration** > **Define MCF Clusters** (MCF_UQCLUSTER)

This example illustrates the fields and controls on the UQ Cluster page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
MCF Cluster ID	Displays the MCF cluster ID.

Field or Control	*Description*
MCF Cluster URL	Displays the URL for the REN server that serves this cluster. This is the URL that is used by internal processes. If the MCF cluster is served by a REN server cluster, the cluster URL is that of the switch or load balancer in front of the clustered REN servers.
Buffer Test	Click to initiate a test of the REN server's ability to break up and send a large file using multiple internal buffers. The buffer test does not depend on specified domain names, so you can use it to verify that RENSRV is running on the network.
MCF Browser URL	Displays the URL for a REN server cluster that serves this MCF cluster for external clients and for agent chat. The browser URL may be different from the cluster URL, which should not have to go through any firewall, reverse proxy server, or other outward-facing security barrier.
Ping Test	Click to initiate a test of the REN server that is specified in the URL fields. Failure may indicate that a URL is incorrectly specified.
Delete	Click to delete the entire MCF cluster. No active agents or tasks should be on the cluster.

Note: If the cluster's queue server configuration is changed, changes to the actual application server domains must be made manually using PSADMIN. For example, if a cluster member (queue server) is removed, the affected application server domain must be shut down and reconfigured (set the **MCF Servers** field to *No*) using PSADMIN. If the cluster URL is changed, all associated queue server domains must be shut down and rebooted.

Queue Server

An MCF cluster can consist of a primary queue server and any number of backup servers.

Each cluster requires a minimum of one queue server. The primary queue server is the first queue server started, and the remaining queue servers are backups. If the primary queue server fails, the system determines the subsequent primary queue server among the backups.

You can add a queue server to a cluster by adding a new row. Before removing a queue server, ensure that it is not the primary, and then shut down its domain. Then click **Delete** (the minus sign).

If a domain is started with a queue server that does not belong to a cluster, the universal queue server and MCF log server poll the MCF configuration tables indefinitely until the queue server is assigned to a cluster.

Field or Control	*Description*
Queue Server ID	Enter a unique identifier for each queue server to identify its entries in the database control tables. The log server process that is paired with this queue server uses this same ID to identify its entry in the log cluster table.
Application Server Domain	Enter the application server domain of which this queue server is a member.
Host Machine	Enter the name of the application server host machine. **Note:** In some cases, the name of the server may be the fully qualified host name.

Chapter 6

Configuring PeopleSoft MCF Queues and Tasks

Defining Queues

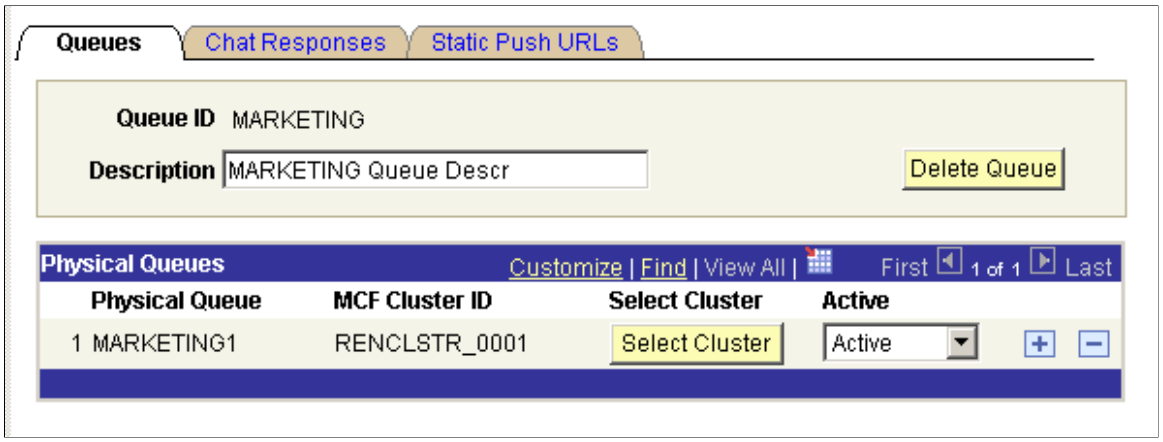
To define queues, use the MCF Queue (MCF_Q_CONFIG_CMP) component.

Defining Queues

Access the Queues page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Configuration** > **Define MCF Queues** > **Queues**

This example illustrates the fields and controls on the Queues page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Queue ID	Queue IDs must be alphanumeric, cannot end in a numeral, but can include underscore characters.

Field or Control	*Description*
Delete Queue	Click to remove this queue. Deleting a logical queue means that no work or agents can be assigned to the queue, and the queue is removed from all agents' available queues. You can delete a logical queue only if all of its constituent physical queues are inactive and have no tasks. Verify that no application code assigns tasks to a queue before deleting the queue. All agents that are assigned to the child physical queue will receive a message notifying them to sign out of their MultiChannel Consoles when the logical queue is deleted.
Physical Queue	Identifies that part of a logical queue that is serviced by the selected MCF cluster. Physical queue IDs always end in a number, for example, SALES2. The physical queue identifier automatically increments by one for each physical queue that is added. Physical queue IDs are automatically generated. The maximum number of physical queues is nine.
MCF Cluster ID	Identifies the MCF cluster that services this physical queue. Click **Select Cluster** to select from configured MCF clusters. Each MCF cluster can service only one physical queue per logical queue. For example, an MCF cluster could service physical queue SALES1 or SALES2, but not both. An MCF cluster can service multiple physical queues belonging to different logical queues. For example, an MCF cluster could service physical queues SALES1, MARKETING2, and COBOL1.
Select Cluster	Click to select a cluster from a list of available MCF clusters. The selected MCF cluster services this physical queue. The primary queue server in this cluster manages tasks and agents that are assigned to this physical queue.

Field or Control	*Description*
Active	Select *Active* or *Inactive* from the drop-down list box. The queue server does not send new tasks to an inactive physical queue. Agents and existing tasks remain on an inactive physical queue. The physical queue must be active to receive new queued tasks. Only inactive physical queues can be deleted from a logical queue. Inactivate a physical queue and complete or transfer all assigned tasks before deleting the physical queue. Active and inactive status support *follow-the-sun* practices. For example, an organization could support SALES1 in the London office, and SALES2 in the San Francisco office when the London office is closed by activating the appropriate queues.

Defining Chat Responses

Access the Chat Responses page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Configuration** > **Define MCF Queues** > **Chat Responses**

This example illustrates the fields and controls on the Chat Responses page. You can find definitions for the fields and controls later on this page.

Chat responses that are specified on this page are available to all agents who are signed in to this queue.

Field or Control	*Description*
Contact Type	Select one of the following contact types: • *Chat* • *Email* (not currently supported) • *Generic* (not currently supported)

Field or Control	*Description*
Response Name	The response name appears in the agent's **Template Messages** drop-down list box for all agents who belong to a physical queue on this logical queue All chat responses are downloaded when the agent launches the agent chat console by accepting a customer chat. Template messages are not available in collaborative chat.
Response Text	The specified message appears in the client chat window when selected by the agent.

Defining Static Push URLs

Access the Static Push URLs page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Configuration** > **Define MCF Queues** > **Static Push URLs**

This example illustrates the fields and controls on the Static Push URLs page. You can find definitions for the fields and controls later on this page.

Static URLs enable the agent to push a predefined URL to the customer, which appears in a pop-up window for the customer. The agent can edit these URLs in the chat window after selecting them.

Field or Control	*Description*
URL Name	Enter a name to identify the URL. The URL name appears in the agent's Static URL drop-down list box for all agents who belong to this logical queue
URL Description	Enter a description of the URL. This description appears only on this page to further describe this URL or, for example, its reason for inclusion.

Field or Control	*Description*
URL	Enter the queue push URL. The URL must include the opening http:// and any required parameters. All static URLs that are defined for the queue are downloaded when the agent launches the agent chat console by accepting a customer chat. Static URLs are not available in collaborative chat. If you send a static push URL that is a PeopleSoft Pure Internet Architecture URL, ensure that the recipient has permissions to access that portal, node, or page.

Configuring Tasks

To configure tasks, use the MCF Task (MCF_TASKCFG_CMP) component.

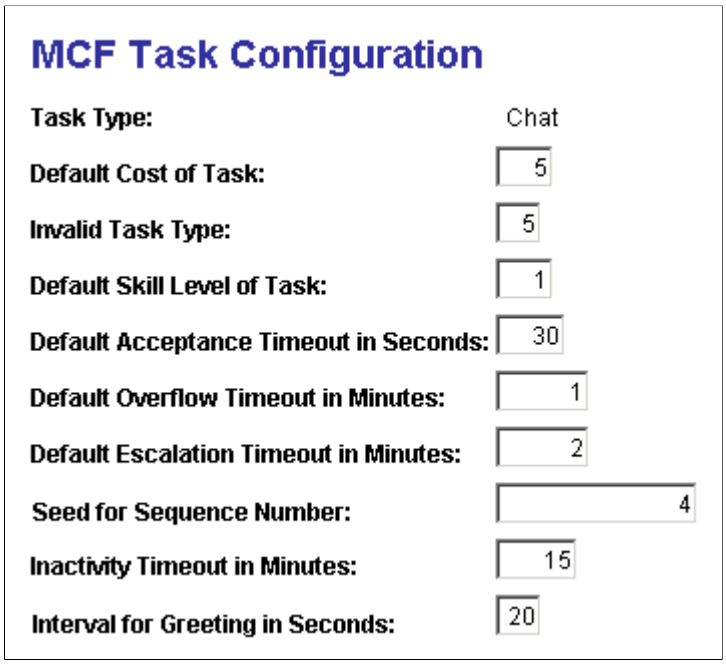
This section discusses how to configure tasks.

Configuring Tasks

Access the MCF Task Configuration page using either of the following navigation paths whichever is appropriate to you:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Configuration** > **Define MCF Tasks**(if you are using the CTI server) or **PeopleTools** > **Multichannel Framework** > **Third-Party Configuration** > **MCF Third-Party Tasks** (if you are using a third-party routing server).

This example illustrates the fields and controls on the MCF Task Configuration page. You can find definitions for the fields and controls later on this page.



Define values for different types of tasks that the queue server uses to assign tasks to appropriate agents and to manage tasks that are not accepted or closed within configurable time limits.

After you define or change values for a task, you must use the **Refresh Task Properties** button on the Cluster Notify page to propagate the changes to clusters. If you are working with MCF tasks, use the Cluster Notify page (MCF_CL_NOTIFY_PG) by navigating to **PeopleTools** > **Multichannel Framework** > **Universal Queue** > **Configuration** > **Notify MCF Cluster.** If you are configuring third-party tasks, use the third-party Cluster Notify page (MCFTP_CL_NOTIFY_PG) by navigating to **PeopleTools** > **Multichannel Framework** > **Third-Party Configuration** > **Notify MCF Third-Party Cluster.**

Field or Control	*Description*
Default Cost of Task	Enter the cost of the task. Cost is a measure of the workload that each task places on an agent. The cost of a task is an estimate of the task's expected complexity and of the time that is required to resolve the task. The minimum value is *0,* and no maximum value exists. The costs of tasks that are assigned to an agent are added up and evaluated against the maximum workload for each agent to determine whether the agent can receive additional tasks. For example, if an agent has a maximum workload of 100, and the default cost of a chat is 20, the agent can manage five concurrent chat sessions, assuming that the default cost is not overridden in the InitChat() built-in function call and that no other task types have been assigned. **Note:** Although priority has no effect on voice tasks (which are not queued), voice task cost is included in calculating an agent's workload. Default costs are: • Chat: *5* • Email: *2* • Generic: *1* • Voice: *10*

Field or Control	*Description*
Default Priority of Task	Enter the priority of this task. A higher value means a higher priority. Tasks are ordered on a physical queue based on their assigned priority. The minimum value is *0,* and no maximum value exists. A queue server gives precedence to a task of higher-priority value over a task of lower-priority value when looking for an agent to assign the task to. This means that the queue server always assigns a task of priority 100 to a qualified available agent before it looks for an agent for a task of priority 10. If two tasks have the same priority, they are assigned in the order of their enqueue time. The value that is specified here can be overridden in the EnQueue() or InitChat() built-in function call. **Note:** Priority has no effect on voice tasks, which are not queued; however, voice task cost is included in calculating an agent's workload. Default priorities are: • Chat: *5* • Email: *2* • Generic: *1* • Voice: *10*
Default Skill Level of Task	Enter the minimum agent skill that is required to handle this task. The queue server assigns this task type to an available agent with the lowest skill level on that queue that is greater than or equal to the skill level that is required by the task. The minimum value is *0,* and no maximum value exists. The value that is specified here can be overridden in the EnQueue() or InitChat() built-in function call. Default skill levels are: • Chat: *1* • Email: *1* • Generic:*1* • Voice : *2* **Note:** Only the third-party routing server supports voice channel. The queue server does not route voice tasks.

Field or Control	*Description*
Default Acceptance Timeout in Seconds	Specify the period of time that an agent has to accept an assigned task (to click the flashing icon on the MultiChannel Console). If the task is not accepted within this time, the task is reenqueued for assignment to another agent. The queue server uses an algorithm to minimize reassignment of tasks that previously timed out to the same agent. The value that is specified here can be overridden in the EnQueue() or InitChat() built-in function call. Default acceptance timeouts are: • Chat: *30* • Email: *30* • Generic: *30* • Voice : *10* **Note:** Only the third-party routing server supports voice channel. The queue server does not route voice tasks.
Default Overflow Timeout in Minutes	Specify the overflow time-out. The overflow time-out is the time period that a queue server has to find an agent who accepts a task (click the flashing icon on the MultiChannel console). If the task is not accepted within this time, the task is removed from the queue and placed in the overflow table. This table can be managed from the Overflow Administration page. The value that is specified here can be overridden in the EnQueue() or InitChat() built-in function call. Default overflow time-outs are: • Chat: *2* • Email: *120* • Generic: *120* • Voice: *1* **Note:** Only the third-party routing server supports voice channel. The queue server uses CTI to route voice tasks.

Field or Control	*Description*
Default Escalation Timeout in Minutes	Specify the default escalation time-out. The escalation time-out is the time period within which a task must be closed. If the task is not closed within this time, the task is removed from the queue and from the agent's accepted task list (that is, the task is unassigned) and the task is placed in the escalation table. This table can be managed from the Escalation Administration page. Escalation time-out is valid on a chat session only after it is accepted by an agent, and has no effect on voice tasks (CTI). The value that is specified here can be overridden in the EnQueue() or InitChat() built-in function call. Default escalation time-outs are: • Chat: *10* • Email: *480* • Generic: *480* • Voice: *2* **Note:** Only the third-party routing server supports voice channel. The queue server uses CTI to route voice tasks.
Seed for Sequence Number	Displays the current cumulative count of tasks of this type that are enqueued. Do not modify this value until it has reached its upper limit of 2,147,483,647. This value does not apply for the voice task type.
Inactivity Timeout in Minutes	Specify the inactivity time-out Inactivity time-out applies to chat only. The inactivity timeout is the time period within which an agent or customer must participate in a chat. If the chat session is dormant for more than this time, the chat is terminated. The default value is *15* minutes.
Interval for Greeting in Seconds	Specify the interval, in seconds, over which the initial greeting appears in a customer chat window while the system searches for an available agent.

Note: All task parameters are delivered with sample values. Determine a range for these values that is appropriate to your business requirements. For example, task cost could vary over a range of 1 to 100 instead of 1 to 10.

Related Links

Notifying Clusters of Changed Parameters

Viewing the Cluster Summary

To view the Cluster Summary page, use the Cluster Summary (MCF_RSERV_CFG_CMP) component.

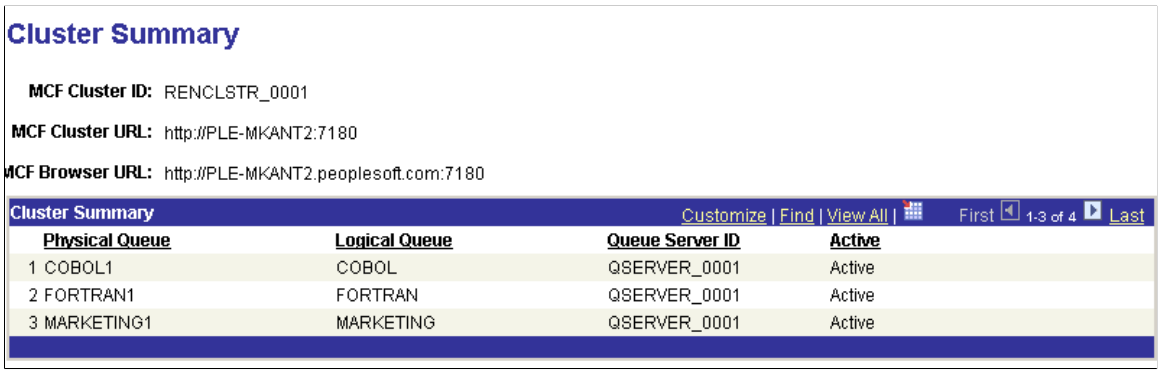
This section discusses how to view the Cluster Summary page.

Viewing the Cluster Summary

Access the Cluster Summary page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > Configuration > MCF Cluster Summary

The Cluster Summary page displays details of the selected MCF cluster.



The Cluster Summary page displays the associated MCF cluster URLs and queue details for the selected MCF cluster. The information cannot be changed from this page.

Tuning Cluster Parameters

To tune cluster parameters, use the Cluster Tuning (MCF_SYSTEM_NV_CMP) component.

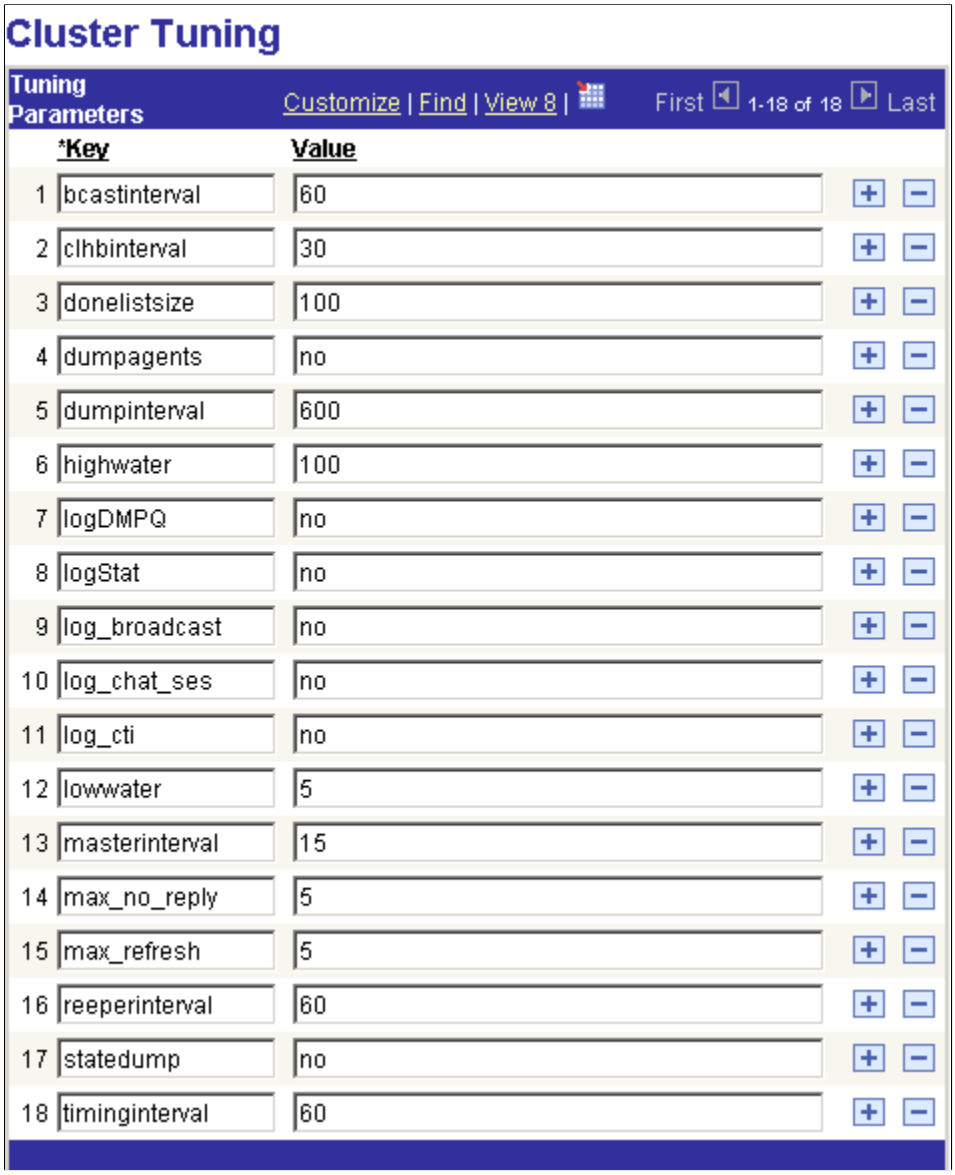
This section discusses how to tune cluster parameters.

Tuning Cluster Parameters

Access the Cluster Tuning page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > Configuration > Tune MCF Clusters

This example illustrates the tuning parameters on the Cluster Tuning page.



Cluster Tuning

Tuning Parameters | Customize | Find | View 8 | First 1-18 of 18 Last

	*Key	Value
1	bcastinterval	60
2	clhbinterval	30
3	donelistsize	100
4	dumpagents	no
5	dumpinterval	600
6	highwater	100
7	logDMPQ	no
8	logStat	no
9	log_broadcast	no
10	log_chat_ses	no
11	log_cti	no
12	lowwater	5
13	masterinterval	15
14	max_no_reply	5
15	max_refresh	5
16	reeperinterval	60
17	statedump	no
18	timinginterval	60

Use the Cluster Tuning page to set MCF cluster parameters to optimize performance or enable logging.

If you make changes to a cluster parameter, you must use the Notify Cluster page to propagate the changes.

See Notifying Clusters of Changed Parameters.

The following table lists the cluster tuning parameters you can modify and describes the default values and usage of each.

Key	*Default value*	*Usage*
bcastinterval	*60*	The interval, in seconds, after which the number of unassigned tasks per physical queue and the number of agents that are logged into each physical queue are broadcast to the MultiChannel Consoles for display next to the queue names. A smaller value provides more accurate queue statistics, but increases the load on the queue server and REN server. A larger value decreases queue server and REN server load, but also decreases statistical accuracy. The bcastinterval value also determines how frequently onStat2 event statistics are calculated. A smaller value provides updated statistics more frequently. This is a timing parameter. If you change the value of this parameter, you must use the **Refresh Threshold/Timing Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters, Using the Monitor Queues Page and Sample Monitor - Queue Statistics Page, Using the Monitor Agents Page and Sample Monitor - Agent States Page.

Key	*Default value*	*Usage*
clhbinterval	*30*	The interval, in seconds, in which the queue server expects to receive a heartbeat from a connected JSMCAPI client. If the queue server receives no heartbeat during this interval, the queue server stops the client session. For a queue server, to avoid a session time-out for the client when CPU usage of the machine on which the client is running is high, increase the heartbeat interval of the client. **Note:** For third-party server, increase the heartbeat interval on the third-party server side to avoid a session time-out. This is a timing parameter. If you change the value of this parameter, you must use the **Refresh Threshold/Timing Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters, Using and Demonstrating JSMCAPI.
donelistsize	*100*	The number of completed tasks that are stored in the list that is used to calculate average task duration. Configure the donelistsize value depending on the task volume that is encountered and the interval over which you want to monitor tasks. This is a timing parameter. If you change the value of this parameter you must use the **Refresh Threshold/Timing Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters, Using and Demonstrating JSMCAPI.

Key	*Default value*	*Usage*
dumpagents	*No*	Enter *Yes* if the status of agent activity should be written to the database during the periodic state dumps. Logging agent status increases queue server load, but provides information about agent performance. This is a task parameter. If you change the value of this parameter, you must use the **Refresh Task Properties** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters, Viewing the Agent State Summary.
dumpinterval	*600*	The interval, in seconds, after which the queue state is written to the database. A smaller value increases load on the queue server, but provides more frequent statistics. A larger value decreases load on the queue server, but provides less frequent statistics. A value of less than one minute will significantly reduce queue server performance. This is a timing parameter. If you change the value of this parameter, you must use the **Refresh Threshold/Timing Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters, Viewing the Queue Server State, Viewing the Queue State Summary.

Key	*Default value*	*Usage*
highwater	*100*	The maximum number of persistent tasks that are retrieved from the database and cached in memory in the queue server. The highwater and lowwater mark values determine how often, and how many, persistent tasks should be read into memory. A higher value causes the queue server to retrieve more persistent tasks at one time, which results in less frequent access to the database, but also more tasks for the queue server to manage, slowing performance. A lower value speeds performance, but requires more frequent access to the database. If a large number of enqueued tasks cannot be routed by the queue server (for example, a call center handling a specific physical queue is offline), increase the highwater mark so that tasks that can be routed can fit into memory cache. This is a threshold parameter. If you change the value of this parameter, you must use the **Refresh Threshold/Timing Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters.
logDMPQ	*No*	Enter *Yes* if you want PSMCFLOG to log REN server event notifications resulting from bcastinterval broadcasts. Only the event is logged, not its contents. This is a logging parameter. If you change the value of this parameter, you must use the **Refresh Logging Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters, Viewing Event Logs.

Key	*Default value*	*Usage*
logStat	*No*	Enter *Yes* to log the statistics that are returned by the queue server for the onStat1 user and group events to the database. This is a logging parameter. If you change the value of this parameter, you must use the **Refresh Logging Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters, Using and Demonstrating JSMCAPI.
log_broadcast	*No*	Enter *Yes* to activate logging of the broadcast messages that are sent. This is a logging parameter. If you change the value of this parameter, you must use the **Refresh Logging Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters, Viewing Broadcast Logs.
log_chat_ses	*No*	Enter *Yes* to activate logging of the contents of chat sessions. This is a logging parameter. If you change the value of this parameter, you must use the **Refresh Logging Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters, Viewing Chat Logs.
log_cti	*No*	Select *Yes* to activate logging of CTI events. This is a logging parameter. If you change the value of this parameter, you must use the **Refresh Logging Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters, Logging CTI Events.

Key	*Default value*	*Usage*
lowwater	*5*	The minimum number of persistent tasks that are cached in memory in the queue server. When the lowwater value is reached, the queue server retrieves another batch of persistent tasks, up to the highwater value. The highwater and lowwater mark values determine when, and how many, persistent tasks should be read into memory. A higher value requires the queue server to access the database more frequently. A lower value can cause the queue server to run out of persistent tasks before refreshing its queue. The lowwater value should be greater than or equal to the maximum number of agents that are logged onto any physical queue at one time. This is a threshold parameter. If you change the value of this parameter, you must use the **Refresh Threshold/Timing Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters.

Key	*Default value*	*Usage*
masterinterval	*15*	The interval, in seconds, after which a cluster primary updates its timestamp in its cluster tables. Secondary clusters check the timestamp to determine whether the primary cluster is still running. A lower value enables rapid discovery of a failed primary server, but increases queue server overhead. A higher value reduces queue server overhead, but delays discovery of a failed primary server. If only one queue server is configured for an MCF cluster, this value can be large. The masterinterval value also acts as a heartbeat interval for the primary queue server connection to user consoles. This is a timing parameter. If you change the value of this parameter, you must use the **Refresh Threshold/Timing Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters.
max_no_reply	*5*	Sets the maximum number of consecutive agent timeouts before the queue server automatically signs out the agent and sets the agent's console status as Assumed Unavailable. This is a timing parameter. If you change the value of this parameter, you must use the **Refresh Threshold/Timing Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters.

Key	*Default value*	*Usage*
max_refresh	*5*	Sets the maximum number of consecutive times that results are discarded when task queue is refreshed from the database if an intervening notification of new persistent tasks exists. This is a timing parameter. If you change the value of this parameter, you must use the **Refresh Threshold/Timing Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters.
reeperinterval	*60*	The interval, in seconds, after which deleted tasks are cleared from memory in the queue server. A lower value increases queue server load but clears memory more frequently. A higher value decreases queue server load but clears memory less frequently. This is a timing parameter. If you change the value of this parameter, you must use the **Refresh Threshold/Timing Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters.
statdump	*No*	Specify *Yes* to write queue server state to the database during the periodic state dumps. The state dump interval is set by the dumpinterval parameter. This is a task parameter. If you change the value of this parameter, you must use the **Refresh Task Properties** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters, Using and Demonstrating JSMCAPI.

Key	*Default value*	*Usage*
timinginterval	*60*	The interval, in seconds, after which the database is checked for expired or overflowed persistent tasks. This parameter does not affect real-time tasks. A lower value increases queue server load but detects timed-out tasks more quickly. A higher value decreases queue server load but detects timed-out tasks less frequently. This is a timing parameter. If you change the value of this parameter, you must use the **Refresh Threshold/Timing Parameters** button on the Cluster Notify page to notify clusters of the changed parameter. See Notifying Clusters of Changed Parameters.

Notifying Clusters of Changed Parameters

To notify clusters of changed parameters, use the Cluster Notify (MCF_AD_NOTIFY_CMP) component.

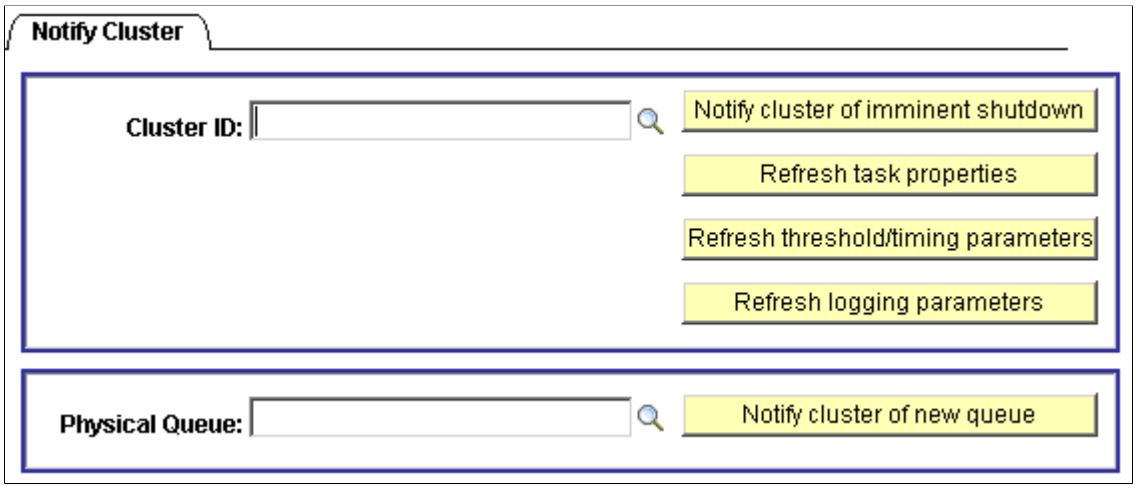
This section discusses how to notify clusters of changed parameters.

Notifying Clusters of Changed Parameters

Access the Notify Cluster page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Configuration** > **Notify MCF Cluster**

This example illustrates the fields and controls on the Notify Cluster page. You can find definitions for the fields and controls later on this page.



Use the Notify Cluster page to notify an MCF cluster of certain changes to its parameters or constituent queues, or that its application servers are being shut down.

For example, after changing MCF cluster parameters on the Cluster Tuning page, use the Notify Cluster page to refresh the tuning parameters.

Field or Control	*Description*
Notify cluster of imminent shutdown	Click to send a message to all agents who are signed in to the selected MCF cluster that they have been signed out. Send this notification if the cluster's application servers are being shut down.
Refresh task properties	Click to load task properties that have been changed on the Tasks page for the selected MCF cluster.
Refresh threshold/timing parameters	Click to reload threshold and timing parameters that have changed on the Cluster Tuning page for the selected MCF cluster.
Refresh logging parameters	Click to reload logging parameters that have changed on the Cluster Tuning page for the selected MCF cluster.
Notify cluster of new queue	Click to notify the selected MCF cluster that the selected physical queue has been added.

Chapter 7

Configuring PeopleSoft MCF Agents

Defining Agents

To define agents, use the MCF Agent (MCF_AGENT_CMP) component.

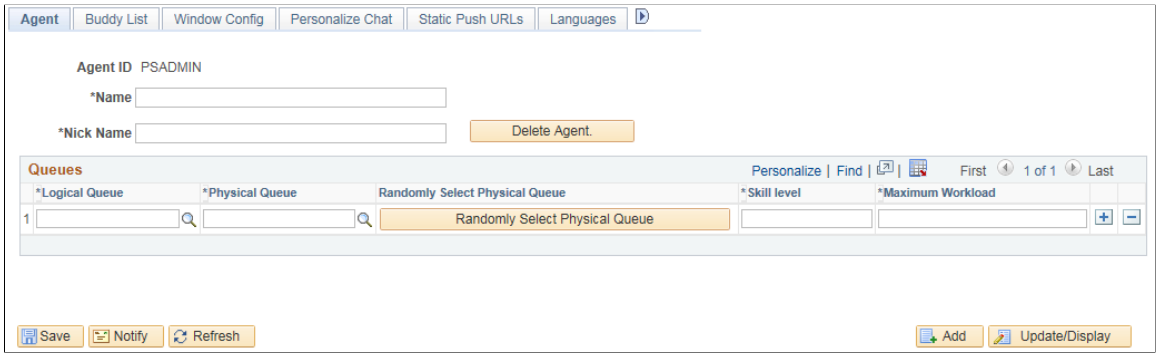
The previous three agent definition pages form the basis of agent configuration. Other parameters are optional.

Creating Agents

Access the Agent page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Administration** > **Define MCF Agents** > **Agent**

This example illustrates the fields and controls on the Agent page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Name	Enter the full name, in (lastname,firstname) format, of this agent. The agent name appears in other agents' buddy lists. **Note:** There is no space in between lastname, firstname.
Nick Name	Enter a short name for this agent. The agent nickname identifies this agent in chat sessions and logs.

Field or Control	*Description*
Delete Agent	An agent cannot be deleted if the agent still has accepted tasks on any queues to which the agent belongs. Before deleting an agent, ensure that the agent is logged off from all queues to which the agent is assigned.
Logical Queue ID	Enter the ID of a logical queue to which this agent is assigned. Each agent can be assigned to more than one logical queue. An agent can log on to only one queue at a time from the MultiChannel Console. **Note:** Do not overwrite the logical queue except when first creating an agent, as the agent's tasks may lose their assignments.
Physical Queue ID	Agents are randomly assigned to a physical queue when the logical queue is associated with the agent. An agent who services a logical queue logs on to a physical queue that is managed by a specific MCF cluster. While an agent can service multiple logical queues, the agent can belong to only one physical queue per logical queue. **Note:** Do not overwrite the physical queue except when first creating an agent, as this may orphan tasks. Use the Physical Queues Move Agent page.
Randomly Select Physical Queue	Click to assign another physical queue (within this logical queue) randomly. This selection will help to spread multiple agents evenly over available physical queues.
Skill Level	Select the skill level of this agent for the tasks that are assigned for this queue. This field is required. The agent is assigned only tasks requiring a skill level that is less than or equal to the skill level specified here. If more than one qualified agent is available to accept the task, the queue server gives preference to the agent with the lowest skill level. Each agent can have a different skill level for each queue to which the agent is assigned.
Maximum Workload	Select the maximum load that this agent can be assigned before tasks are held or assigned to other agents. This field is required. The cost of each accepted task is added to the agent's current workload. A task is not assigned to an agent if its cost pushes the agent's current workload over the maximum.

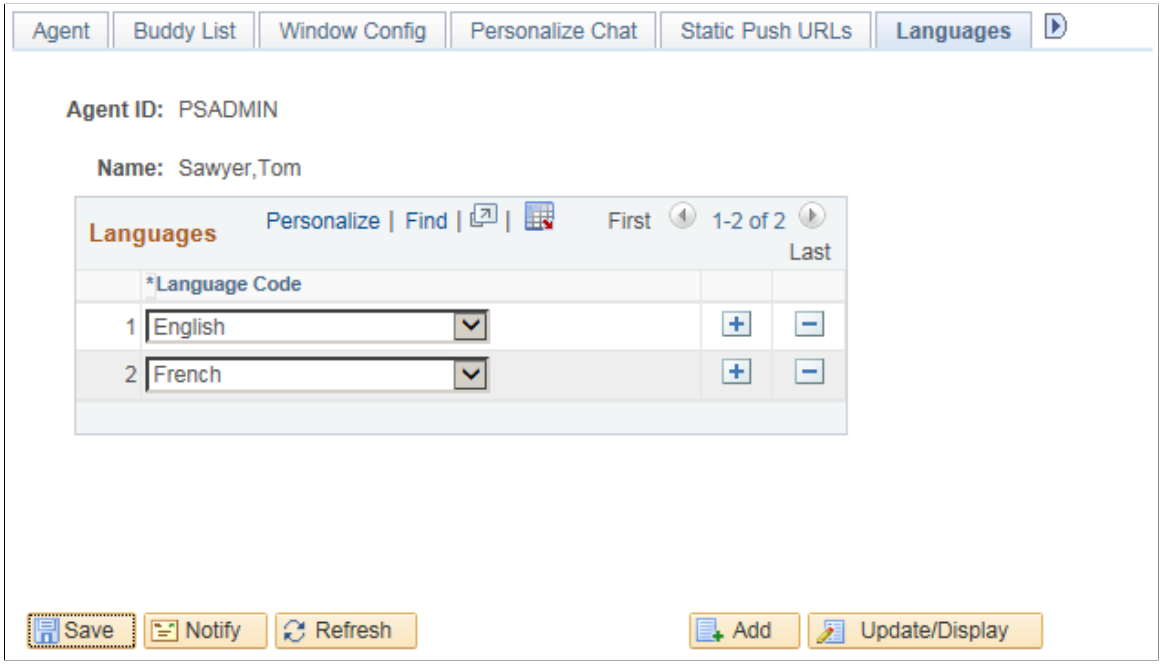
Note: Do not delete a queue from an agent's list unless that agent has no open accepted tasks in that queue.

Specifying Languages That an Agent Supports

Access the Languages page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > Administration > Define MCF Agents > Languages

This example illustrates the fields and controls on the Languages page.



Specify the languages that each agent is qualified to support.

The agent is assigned only tasks that have been enqueued with a language code in the agent's language list. For the EnQueue() built-in function, the language code is specified as a parameter. For the InitChat() built-in function, the language code is determined by the user profile of the initiator.

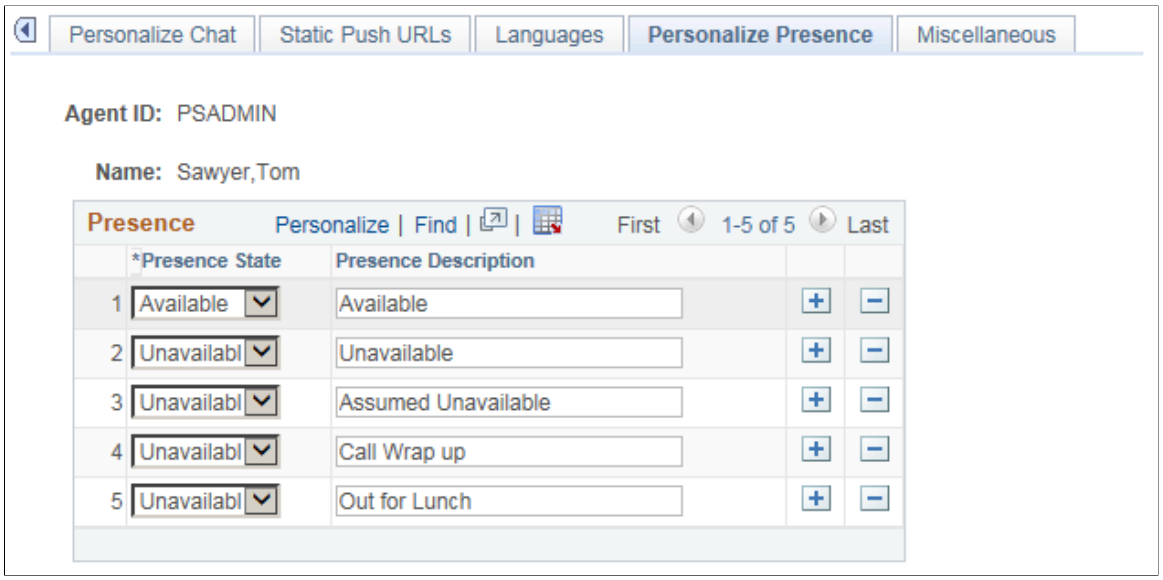
If you do not specify a language code for a new agent, the default value is *English*.

Personalizing an Agent's Presence

Access the Personalize Presence page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > Administration > Define MCF Agents > Personalize Presence

This example illustrates the fields and controls on the Personalize Presence page. You can find definitions for the fields and controls later on this page.



Each agent can configure the presence description that is displayed when the agent is available or unavailable. The queue server understands only the presence state, available or unavailable, but you can specify more specific presence descriptions when displaying or logging an agent's presence. For example, Lunch, Meeting, or Indisposed are unavailable states that can be used for tracking agent time and activity.

If you do not specify presence descriptions, default values are used.

Field or Control	***Description***
Presence State	Select *Available* or *Unavailable.*
Presence Description	Enter a description for each agent state. The description appears in logs of agent activity and when agent presence is displayed. **Note:** The *Available* state has only one description. For the *Unavailable* state, you can enter any presence description, such as tea break, coffee break, lunch, and so on.

Defining Optional Agent Characteristics

To define optional agent characteristics, use the MCF Agent (MCF_AGENT_CMP) component.

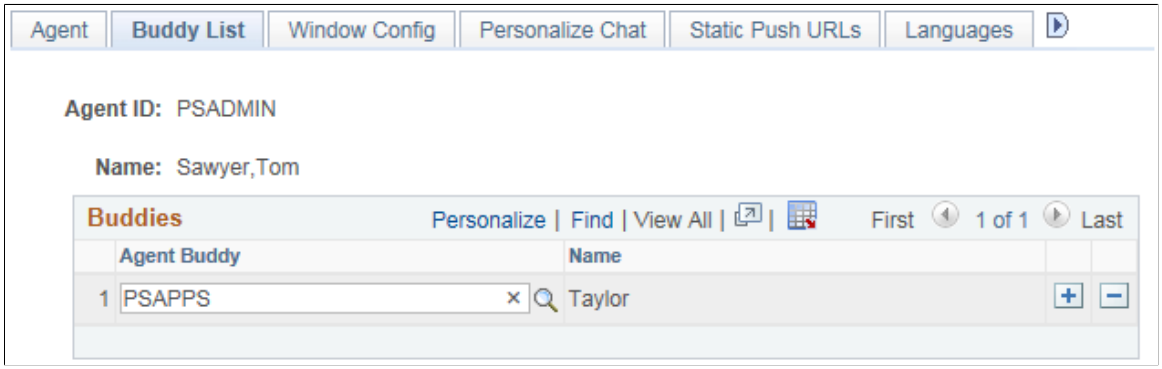
The agent configuration pages are considered optional because most do not have default values and can remain unconfigured without affecting an agent's ability to log on to a queue and accept tasks.

Setting Up Buddy Lists

Access the Buddy List page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > Administration > Define MCF Agents > Buddy List

This example illustrates the fields and controls on the Buddy List page. You can find definitions for the fields and controls later on this page.



The agent's buddy list facilitates collaborative chat and chat conferencing.

Field or Control	*Description*
Agent Buddy	Select another agent with whom this agent can have a chat session or can ask to conference into another chat. Each agent buddy must be logged in a physical queue on the same cluster to be able to chat. If two agents must be able to chat but they do not share a cluster, use the Physical Queue Move Agent page to move the agents into physical queues on the same MCF cluster. Agent buddies are listed with their login status in the buddy list on the multichannel console.
Name	Displays the buddy agent's nickname.

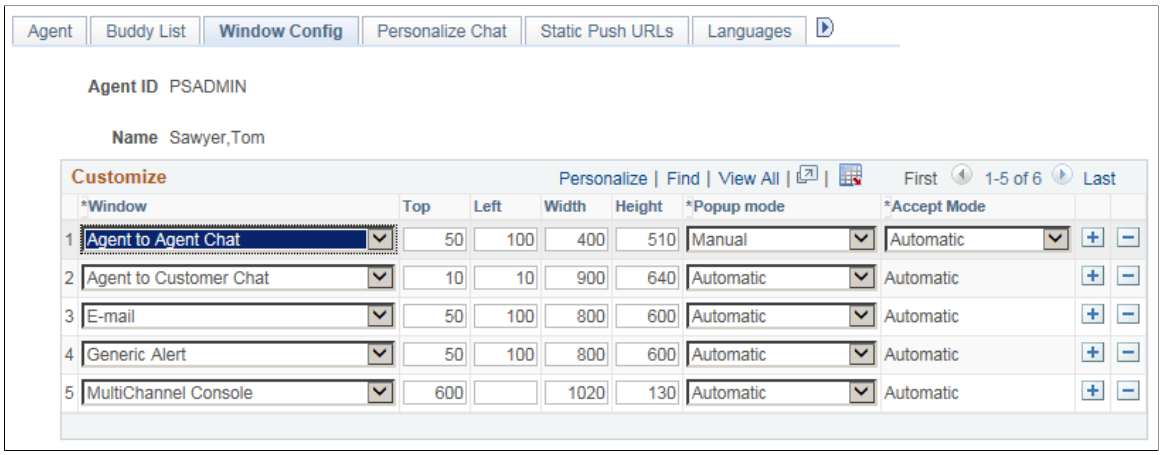
An agent's presence, as shown in the buddy list on the MultiChannel Console or on the Invite Agent list on the chat console, indicates the agent's availability for chat or conference.

Configuring Windows

Access the Window Config page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > Administration > Define MCF Agents > Window Config

This example illustrates the fields and controls on the Window Config page. You can find definitions for the fields and controls later on this page.



Set the initial agent window placement and size by specifying parameters on this page. An agent can resize and move the windows.

Field or Control	***Description***
Window	Select the window to which the specified configuration applies. Select from: • *Agent to Agent Chat* • *Agent to Customer Chat* • *Email* • *Generic Alert* • *Grab URL* • *MultiChannel Console*
Top and **Left**	Enter the distance in pixels from the top and left edges of the screen when the window first appears.
Width and **Height**	Enter the width and height, in pixels, of the window when it first appears.

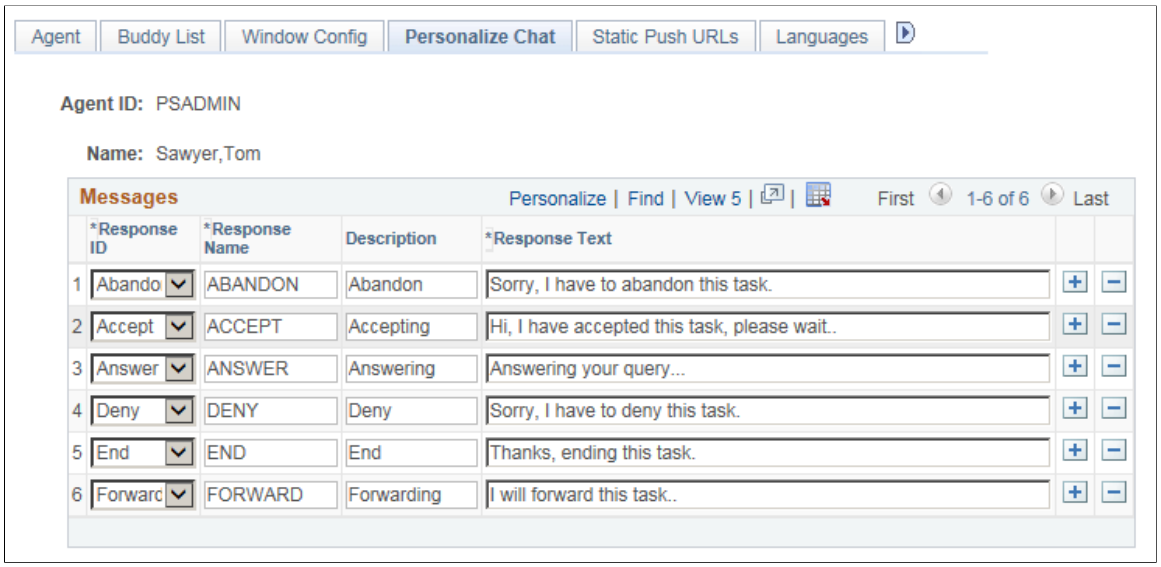
Field or Control	*Description*
Popup Mode	Select from: *Automatic:* The window appears automatically. For customer-initiated chat or tasks that are initiated from the EnQueue() built-in function, the task is automatically accepted as well. For agent-initiated chat, the agent can elect not to accept the task; in effect, the agent can preview the task. If this is the desired behavior, select *Manual* from the **Accept Mode** drop-down list box. If you want agent-to-agent tasks to function like customer-initiated tasks, select *Automatic* from the **Accept Mode** drop-down list box. *Manual:* The window does not appear until the agent clicks the task on the agent MultiChannel Console. For customer-initiated chat or tasks that are initiated from the EnQueue () built-in function, clicking the task means that the task is automatically accepted as well. For agent-initiated chat, the behavior depends on the setting for the **Accept Mode** field.
Accept Mode	Select from: *Automatic:* Agent-to-agent chats are automatically accepted without requiring the agent to click the icon. *Manual:* Agent-to-agent chats require the agent to click the icon. Accept mode affects only collaborative chat.

Personalizing Chat

Access the Personalize Chat page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Administration** > **Define MCF Agents** > **Agent Messages**

This example illustrates the fields and controls on the Personalize Chat page. You can find definitions for the fields and controls later on this page.



An agent can create personalized responses in addition to the system responses that are defined for each queue.

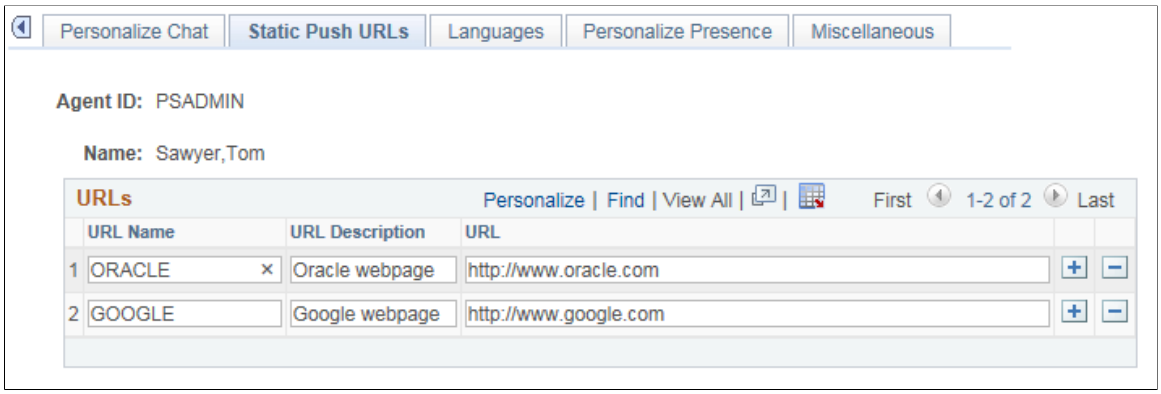
Field or Control	*Description*
Response ID	Responses, except those that are identified by *Other,* are linked to specific events. These responses are always sent on these events from this agent. If an agent does not have a customized response for a specific event, the response is read from a default value that is set in the Message Catalog. The response text that is set here overrides the default text that is set in the Message Catalog. Select from: • *Abandon:* A chat is abandoned when a chat initiator closes the chat window before the chat is accepted by an agent. This message appears when the agent accepts the abandoned chat. • *Accept:* This response is automatically sent to the chat initiator when an agent accepts a chat in response to a chat request that does not include a question. If the chat request includes a question, the agent's *Answer Question* text is sent in response instead of the *Accept* response. • *Answer Question:* This response is automatically sent to the chat initiator when an agent accepts a chat in response to a chat request that includes a question. • *Deny:* This response applies only to collaborative chat. If an agent elects not to accept a chat, this message is automatically sent to the chat initiator. • *End:* If either party quits a chat after the chat is accepted, this message is displayed from the agent. • *Forward:* If the agent forwards a chat session to another queue, this message is sent to the customer. • *Other:* These messages are never automatically sent in a chat session. Their message names appear in the **Template Messages** drop-down list box on the agent chat page. These messages are appended to the template messages (chat responses) that are defined for the queue.
Response Name	This name appears in the agent's template response drop-down list box.
Response Text	Enter the response text to appear in the chat window.

Specifying Agent-Specific URLs

Access the Static Push URLs page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Administration** > **Define MCF Agents** > **Agent Push URLs**

This example illustrates the fields and controls on the Static Push URLs page. You can find definitions for the fields and controls later on this page.



This page defines URLs that this agent can send to a client browser. These URLs are in addition to the URLs that are defined in the queue configuration page.

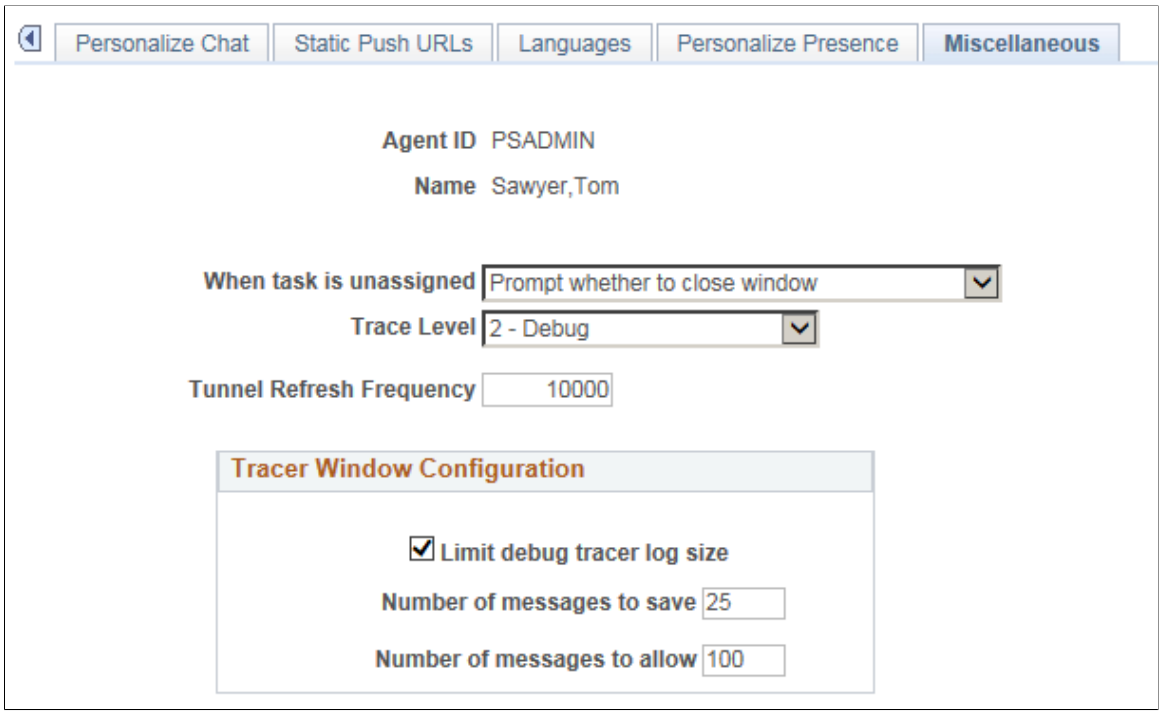
Field or Control	*Description*
URL Name	The URL name appears in the agent's static URL drop-down list box.
URL Description	This description appears only on this page, to further describe this URL or, for example, its reason for inclusion.
URL	Enter the queue push URL. The URL must include the opening http:// and any required parameters. All static URLs that are defined for the agent are downloaded when the agent launches the agent chat console by accepting a customer chat. Static URLs are not available in collaborative chat. If you send a PeopleSoft Pure Internet Architecture URL, be sure that the recipient has permissions to access that portal , node, or page.

Specifying Miscellaneous Parameters

Access the Miscellaneous page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > Administration > Agent > Miscellaneous

This example illustrates the fields and controls on the Miscellaneous page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
When task is unassigned	Select from the following values the action that occurs when a task that is assigned to an agent is unassigned: • *Prompt whether to close window* (default). • *Close the task window.* • *Do not close the task window.*
Trace Level	Select from the following log trace levels: • *0 - None* • *1 - Information* • *2 - Debug* If a value other than *0* is selected, a tracer window appears to display activities and events on the chat or MultiChannel Console for debugging purposes.
Tunnel Refresh Frequency	Specify the number of communications to the Real-time Event Notification (REN) server before a refresh occurs for a browser-based client. Default refresh frequency is 10000. You can configure the field depending on the memory available to the client. A higher value will use more memory, but will allow a longer time between refreshes.

Field or Control	*Description*
Limit debug tracer log size	This check box is enabled when the value entered for **Trace Level** is not 0. Select this check box to enable the agent to clear the tracer log based on **Number of messages to save** and **Number of messages to allow**. If the check box is deselected, the tracer log will not be cleared and the **Number of messages to save** and **Number of messages to allow** will be disabled.
Number of messages to save	Specify the minimum number of recent tracer log messages that should be maintained in the tracer window.
Number of messages to allow	Specify the maximum number of tracer log messages that will be maintained in the tracer window.

Note: Number of messages to save and **Number of messages to allow** fields are required if the **Limit debug tracer log size** check box is selected.

Limit Debug Tracer Log Size Example

This table lists the values entered on the Miscellaneous page:

Field	*Value*
Trace Level	*2 - Debug*
Limit debug tracer log size	Selected
Number of messages to save	*25*
Number of messages to allow	*100*

Based on these values, the first 75 messages will be cleared from the tracer window after 100 messages are logged. It will retain the most recent 25 messages for the agents reference. This process will repeat for every 100 messages that are logged in the tracer. The maximum number of messages is the tracer window at any one point in time is 100.

Chapter 8

Administering Queues, Logs, and Tasks

Administering Physical Queues

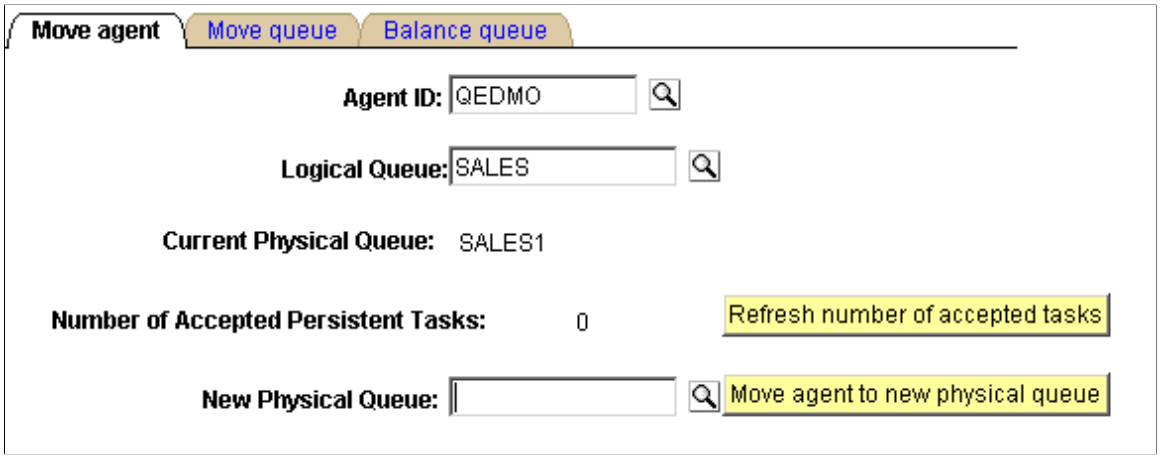
To administer physical queues, use the Physical Queues (MCF_ACCPT_TASK_CMP) component.

Moving Agents Between Physical Queues

Access the Move agent page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Administration** > **Administer MCF Queues** > **Move agent**

This example illustrates the fields and controls on the Move agent page. You can find definitions for the fields and controls later on this page.



You can move an agent, and any open persistent tasks that are associated with that agent, from one physical queue in one cluster to another physical queue in another cluster on the same logical queue.

Field or Control	*Description*
Logical Queue	Select the logical queue within which the agent is to be moved.
Number of Accepted Persistent Tasks	Displays the number of persistent tasks this agent has accepted on this physical queue. This number is updated and displayed when you select the logical queue.

Field or Control	*Description*
Refresh number of accepted tasks	Click to update the number of persistent tasks accepted by this agent on the current physical queue.
New Physical Queue	Select the new physical queue to which this agent and the persistent tasks accepted by this agent will be assigned.
Move agent to new physical queue	Click to perform the action.

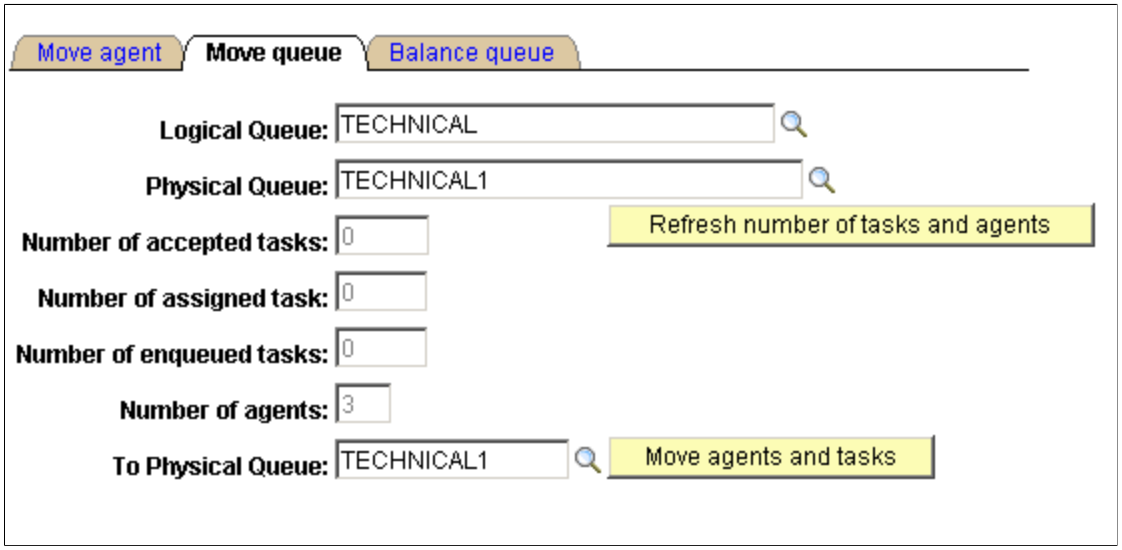
Note: Ongoing chat sessions, which are not persistent tasks, are not affected by the move agent action.

Moving Queues

Access the Move queue page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Administration** > **Administer MCF Queues** > **Move queue**

This example illustrates the fields and controls on the Move queue page. You can find definitions for the fields and controls later on this page.



You can move all agents and their open persistent tasks from one physical queue to another physical queue within the same logical queue. For example, to delete a physical queue, move its agents and persistent tasks to another queue before deleting the first queue. Or, if the cluster serving this physical queue is overloaded, you can create another physical queue on another cluster, mark the first physical queue as inactive, create another physical queue on another cluster and move agents and persistent tasks from the inactive physical queue to the new physical queue.

Field or Control	*Description*
Logical Queue	Select the logical queue within which the selected physical queue's agents and persistent tasks will be moved.
Physical Queue	Select the physical queue from which agents and persistent tasks will be moved.
Number of accepted tasks and **Number of assigned tasks**	Displays the number of accepted and assigned tasks open on this physical queue.
Refresh number of tasks and agents	Click to update the number of agents and persistent tasks assigned to this queue.
Number of enqueued tasks and **Number of agents**	Displays the number of enqueued tasks and assigned agents on this physical queue.
To Physical Queue	Select the physical queue to which the currently assigned agents and tasks will be moved.
Move agents and tasks	Click to move the assigned agents and tasks to the specified physical queue. **Note:** The physical queue must be inactive before moving agents and tasks. Inactivate the physical queue on the Queues page. Agents on the inactive physical queue are automatically logged off before being moved.

Related Links

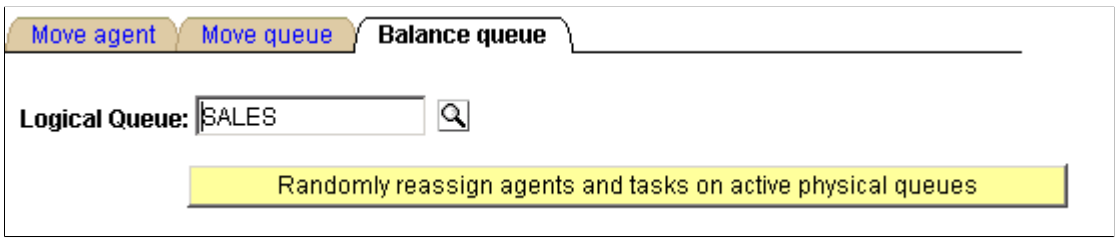
Defining Queues

Balancing Queues

Access the Balance queue page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Administration** > **Administer MCF Queues** > **Balance Queue**

This example illustrates the fields and controls on the Balance queue page. You can find definitions for the fields and controls later on this page.



Over time, the distribution of agents and their associated skill levels, languages, and so on, across the physical queues belonging to a logical queue may change as agents are added or deleted. Rather than manually rebalancing the queue by moving individual agents, you can use the Balance queue page to randomly reassign agents and their open persistent tasks across all the active physical queues belonging to the selected logical queue.

Field or Control	*Description*
Logical Queue	Select the logical queue across which agents and persistent tasks will be balanced.
Randomly reassign agents and tasks on active physical queues	Click to balance agents and tasks across the active physical queues for the selected logical queue. This action redistributes agents and tasks assigned to this logical queue across all active physical queues without regard to previous assignments. Ensure that agents assigned to the affected physical queues have shut down their MultiChannel Consoles. They are *not* logged off automatically.

Viewing Queue Server, Queue, and Agent States

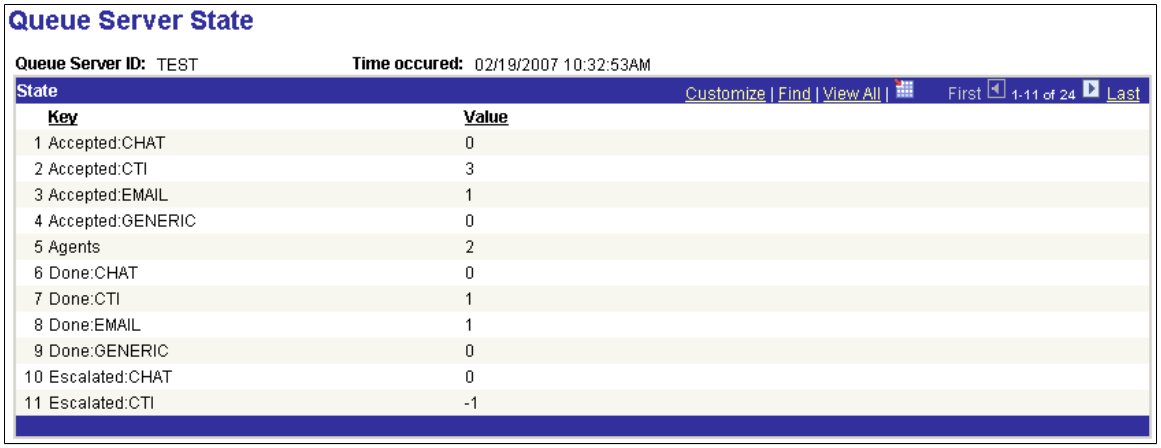
To view queue server state, queue state, and agent state, use the Queue Server State (MCF_QSERVSTATE_CMP), Queue State Summary (MCF_QUEUESTATE_CMP), and Agent State Summary (MCF_AGENTSTATE_CMP) components.

Viewing the Queue Server State

Access the Queue Server State page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Administration** > **MCF Queue Server State**

This example illustrates the cumulative diagnostic totals for the selected queue server on the Queue Server State page.



The Queue Server State page displays cumulative diagnostic totals for the selected queue server, at the selected time, across all physical queues serviced by this queue server. Cumulative totals are reset after each state dump.

This state information comprises statistics to measure load and throughput that can be analyzed and used for tuning the system. For example, if counts are high, consider adding another physical queue to balance the load.

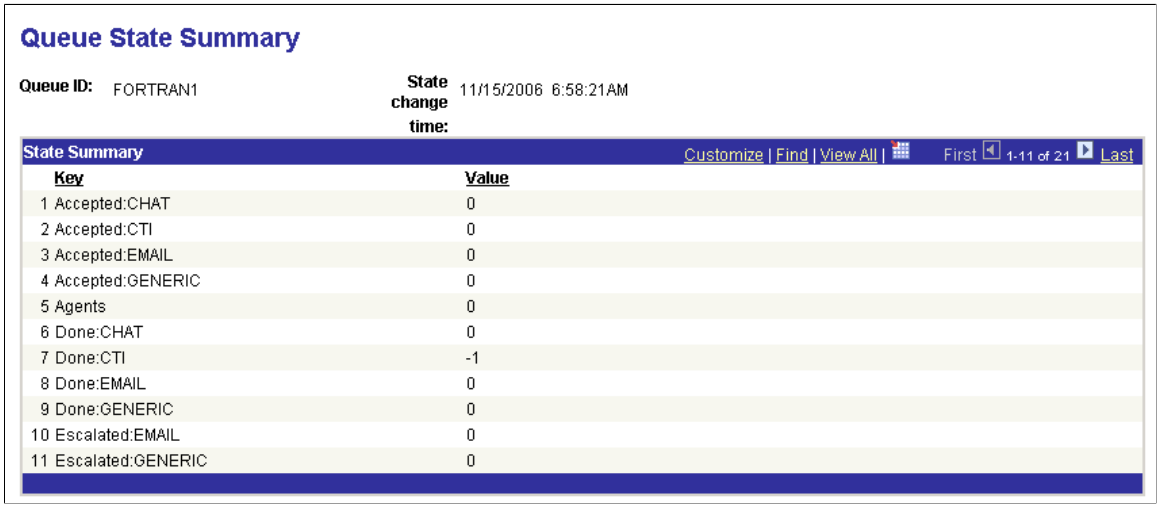
The state stamps are inserted for every primary queue server by cluster at configurable intervals, irrespective of user activity. You configure the intervals on the Cluster Tuning page.

Viewing the Queue State Summary

Access the Queue State Summary page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > Administration > MCF Queue State

This example illustrates the cumulative diagnostic totals for the selected physical queue on the Queue State Summary page.



Queue State Summary

Queue ID: FORTRAN1 State change time: 11/15/2006 6:58:21AM

State Summary — Customize | Find | View All | First 1-11 of 21 Last

	Key	Value
1	Accepted:CHAT	0
2	Accepted:CTI	0
3	Accepted:EMAIL	0
4	Accepted:GENERIC	0
5	Agents	0
6	Done:CHAT	0
7	Done:CTI	-1
8	Done:EMAIL	0
9	Done:GENERIC	0
10	Escalated:EMAIL	0
11	Escalated:GENERIC	0

The Queue State Summary page displays cumulative diagnostic totals for the selected physical queue.

This state information comprises statistics to measure load and throughput that can be analyzed and used for tuning the system. For example, if counts are high for this queue, consider adding another physical queue to balance the load.

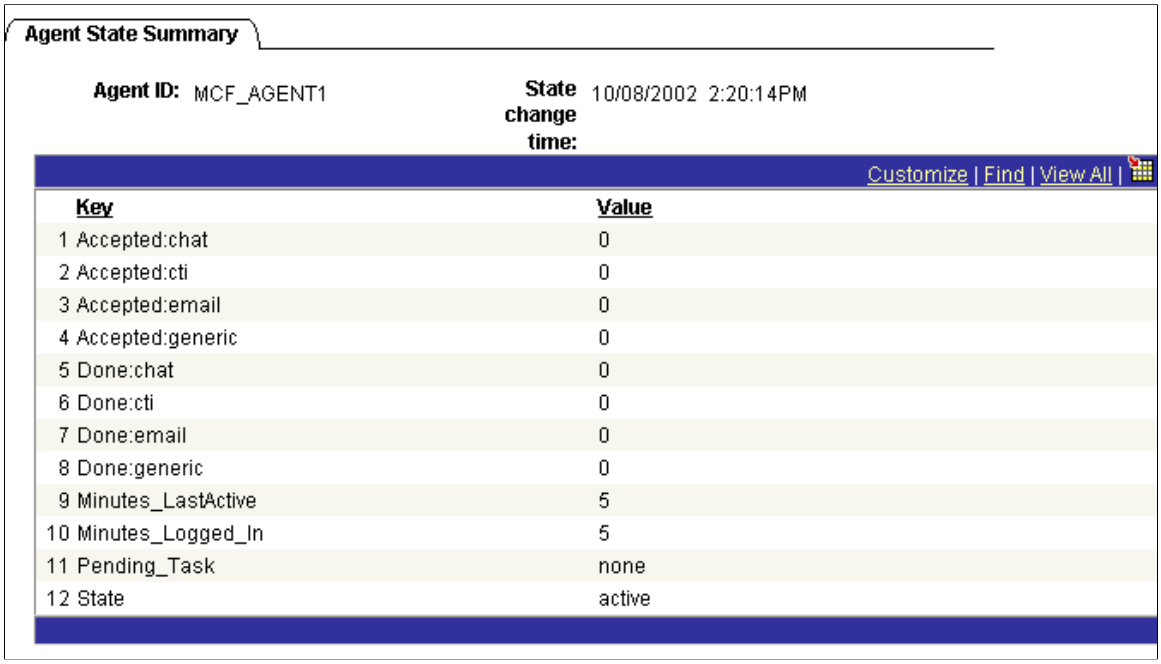
The state stamps are inserted for every physical queue at configurable intervals, irrespective of user activity. The intervals are configured on the Cluster Tuning page.

Viewing the Agent State Summary

Access the Agent State Summary page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > Administration > MCF Agent State Summary

This example illustrates the cumulative totals for the selected agent on the Agent State Summary page.



Agent State Summary

Agent ID: MCF_AGENT1 State change time: 10/08/2002 2:20:14PM

Customize | Find | View All |

	Key	Value
1	Accepted:chat	0
2	Accepted:cti	0
3	Accepted:email	0
4	Accepted:generic	0
5	Done:chat	0
6	Done:cti	0
7	Done:email	0
8	Done:generic	0
9	Minutes_LastActive	5
10	Minutes_Logged_In	5
11	Pending_Task	none
12	State	active

The Agent State Summary page displays cumulative totals for the selected agent.

This state information comprises statistics to measure agent performance and status that can be analyzed and used for performance evaluation.

The state stamps are inserted for every agent at configurable intervals, irrespective of user activity. State is only recorded for agents currently logged on. The intervals are configured on the Cluster Tuning page by setting the **dumpinterval.** Enable agent logging by setting **dumpagent** to *yes*.

An agent state of *Active* indicates the agent is available; *Inactive* indicates the agent is not available.

Note: On a busy system, recording state information frequently may slow performance.

Related Links

Tuning Cluster Parameters

Viewing Broadcast, Chat, and Event Logs

To view broadcast, chat, and event logs, use the Broadcast Log (MCF_BCAST_LOG_CMP), Chat Log (MCF_CHAT_LOG_CMP), and Event Log (MCF_EVENTLOGL_CMP) components.

Viewing Broadcast Logs

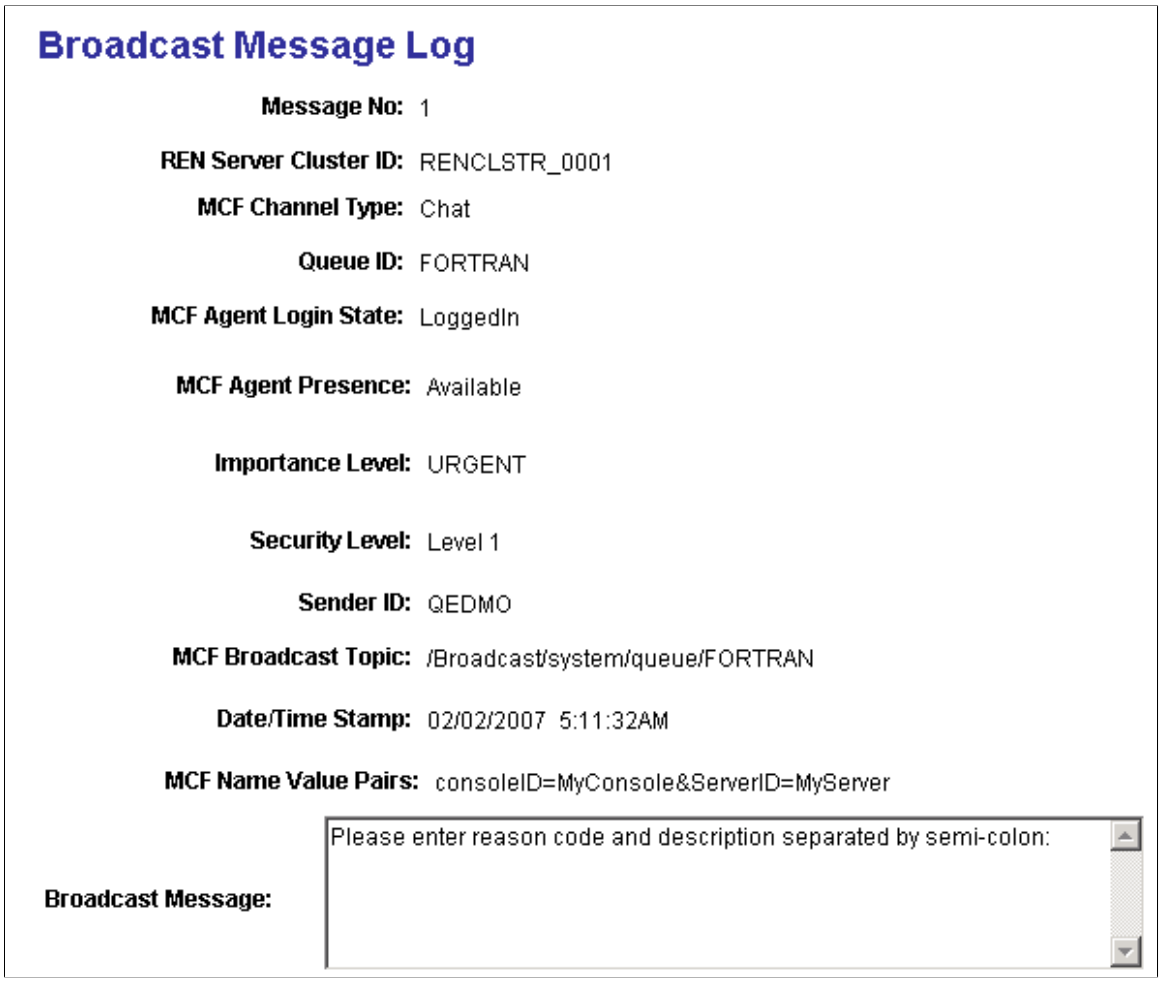
The broadcast log page displays detailed information about any broadcast messages that were sent.

Access the Broadcast Message Log page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Administration** > **MCF Broadcast Log**

You can also access the Broadcast Message Log page by searching by message number, queue ID, or REN cluster ID.

This example illustrates the details of a broadcast message on the Broadcast Message Log page.



Note: To view the broadcast logs, set *log_broadcast* to *Yes* on the Cluster Tuning page. For this parameter to take effect, click **Refresh logging parameters** on the Cluster Notify page.

Related Links

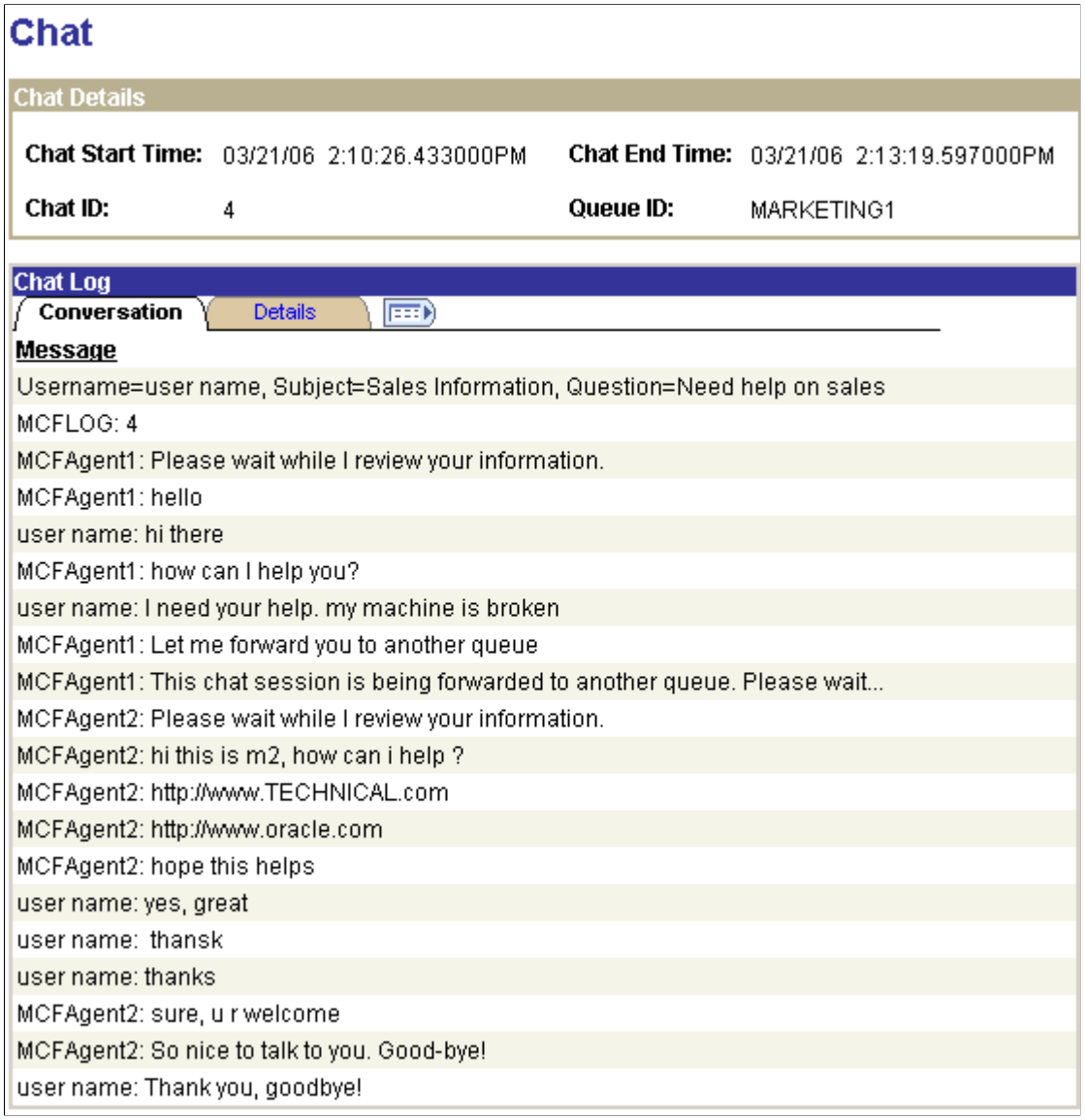
Using PeopleSoft MCF Broadcast

Viewing Chat Logs

Access the Chat page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Administration** > **MCF Chat Log**

This example illustrates a chat log on the Chat page.



If enabled, this log records the contents and events of every chat session.

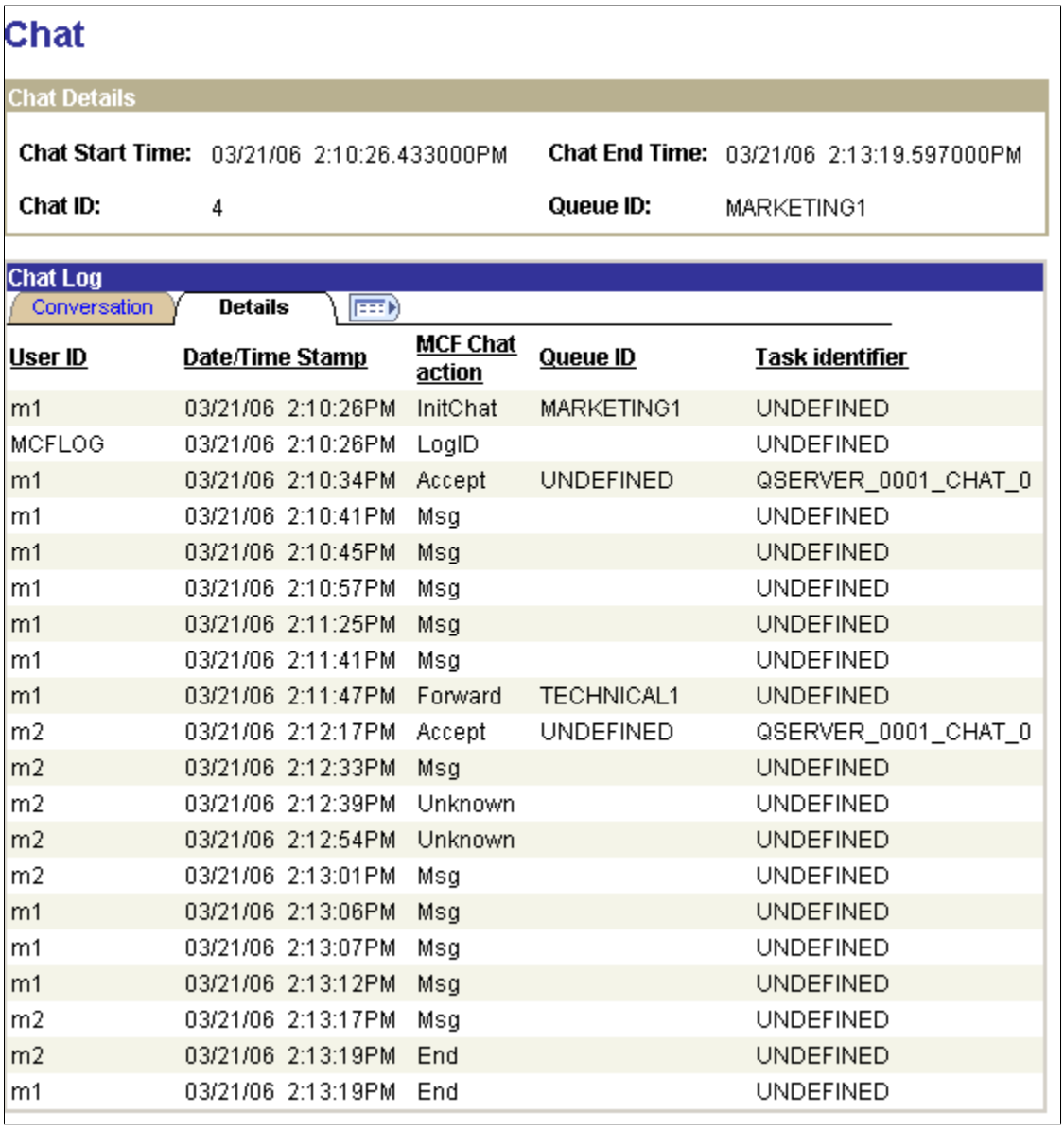
To enable chat logging, set log_chat_ses to *yes* on the Cluster Tuning page. For this parameter to take effect, click **Refresh logging parameters** on the Cluster Notify page.

Details Tab

Access the Details tab of the Chat Page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Administration** > **MCF Chat Log** > **Details**

This example illustrates details of a chat log on the Chat page.



Chat

Chat Details

Chat Start Time:	03/21/06 2:10:26.433000PM	**Chat End Time:**	03/21/06 2:13:19.597000PM
Chat ID:	4	**Queue ID:**	MARKETING1

Chat Log

Conversation | Details

User ID	Date/Time Stamp	MCF Chat action	Queue ID	Task identifier
m1	03/21/06 2:10:26PM	InitChat	MARKETING1	UNDEFINED
MCFLOG	03/21/06 2:10:26PM	LogID		UNDEFINED
m1	03/21/06 2:10:34PM	Accept	UNDEFINED	QSERVER_0001_CHAT_0
m1	03/21/06 2:10:41PM	Msg		UNDEFINED
m1	03/21/06 2:10:45PM	Msg		UNDEFINED
m1	03/21/06 2:10:57PM	Msg		UNDEFINED
m1	03/21/06 2:11:25PM	Msg		UNDEFINED
m1	03/21/06 2:11:41PM	Msg		UNDEFINED
m1	03/21/06 2:11:47PM	Forward	TECHNICAL1	UNDEFINED
m2	03/21/06 2:12:17PM	Accept	UNDEFINED	QSERVER_0001_CHAT_0
m2	03/21/06 2:12:33PM	Msg		UNDEFINED
m2	03/21/06 2:12:39PM	Unknown		UNDEFINED
m2	03/21/06 2:12:54PM	Unknown		UNDEFINED
m2	03/21/06 2:13:01PM	Msg		UNDEFINED
m1	03/21/06 2:13:06PM	Msg		UNDEFINED
m1	03/21/06 2:13:07PM	Msg		UNDEFINED
m1	03/21/06 2:13:12PM	Msg		UNDEFINED
m2	03/21/06 2:13:17PM	Msg		UNDEFINED
m2	03/21/06 2:13:19PM	End		UNDEFINED
m1	03/21/06 2:13:19PM	End		UNDEFINED

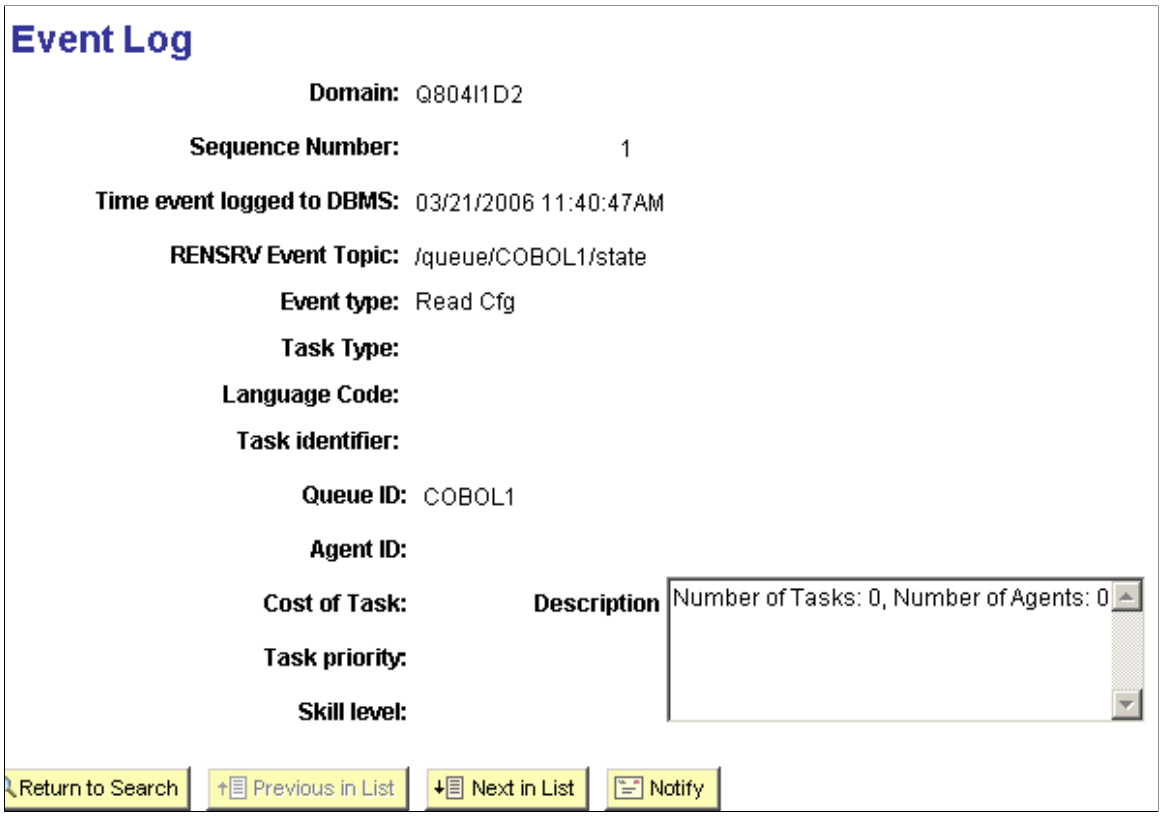
The Details tab displays detailed information about the selected chat conversation.

Viewing Event Logs

Access the Event Log page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > Administration > MCF Event Logs

This example illustrates the fields and controls on the Event Log page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Domain	The application server domain on which this event occurred.
Time event logged to DBMS	The time that this event was recorded in the database.
Event Type	The type of event, as described in the table that follows.
Task Type	The type of task for this event: chat, email, or generic. CTI events are logged in the CTI event log.

The event log records PeopleSoft MultiChannel Framework events sent to the real-time event notification (REN) server, excluding chat content (which is logged in the chat log). The event log can be used for debugging as well as for system monitoring. For example, you can determine when agents log in and log out, or when the queue server was first notified of newly enqueued events.

Data displayed in the event log depends on the event type. Not all fields apply to every event.

To enable logging of state broadcast events, set *logDMPQ* to *yes* on the Cluster Tuning page. For this parameter to take effect, click **Refresh logging parameters** on the Cluster Notify page.

Time is displayed and searched on in the format MM/DD/YYYY HH:MM:SSA/PM.

The following table lists possible event types:

Name	*Translate Table Value*	*REN MultiChannel Framework Topic*	*Description*
Accepted	ACPD	/agent/<agentID>/accepted Agent 's list of accepted tasks	Agent's list of accepted tasks
Accept	ACPT	/queue/agents/accept	Agent accepts an assigned task.
Bcst Admin	BCST	/queue/admin/statedump	Broadcast universal queue information
Contact	CNCT	/queue/contact	Real-time contact (for example, chat)
DB Cntct	DBCT	/queue/dbcontact	Database management system contact (such as email or generic)
Dump Q	DMPQ	/queue/<queueID>/state	Dump queue state information to log
Done	DONE	/queue/agents/dequeue	Done (dequeue)
Forward	FWD	/queue/agents/forward	Forward
Notify	NTFY	/agent/<agentID>/notify	Notify agent of assigned task.
Presence	PRES	/queue/agents/presence	Agent's presence change
Restrt Ack	RACK	/queue/agents/restartack	Restart acknowledgement
Read Cfg	READ	/uqsrv/reread/defaults	Reread defaults
Restart	RSRT	/queue/<queueID>/restart	Restart
Unknown	UNKN	UNKNOWN	Unknown REN server event
Unassign	USGN	/agent/<agentID>/unassign	Unassign

The following table lists event logs:

Name	*Translate Table Value*	*REN MultiChannel Framework Topic*	*Required Argument Value*	*Meaning*
Abandon	ABAN	/chat/<userID>/<chatID>	ps_type=abandon	Abandoned chat session
End	END	/chat/<userID>/<chatID>	ps_type=end	End chat session
Login	LGIN	/queue/agents/loginstate	ps_state=login	Log on
Logout	LGOT	/queue/agents/loginstate	ps_state=logout	Log out
Message	MSG	/chat/<userID>/<chatID>	ps_type=msg	Message
Push URL	PUSH	/chat/<userID>/<chatID>	ps_type=pushurl	Push URL
Timeout	TOUT	/chat/<userID>/<chatID>	ps_type=timeout	Timeout chat session

Viewing PeopleSoft MCF Logs

Diagnostic PSUQSRV and PSMCFLOG traces are written to the log directory of each application server domain. The trace level is determined by the LogFence setting in the domain configuration. You can set the LogFence parameter with PSADMIN configuration of the application server domain. The following table lists LogFence setting values:

LogFence Setting	*Tracing Level*
1	Fatal errors
2	Errors
3	Warnings
4	Level 1 Diagnostic: Logs the queues that the universal queue is servicing, logs new queues that are added, and logs agent logon and logout.
5	Level 2 Diagnostic: Logs most debugging information, except periodic events (such as timer check and heartbeat).

LogFence Setting	*Tracing Level*
6	Level 3 Diagnostic: Logs everything, including periodic events.

Administering Overflow and Escalated Tasks

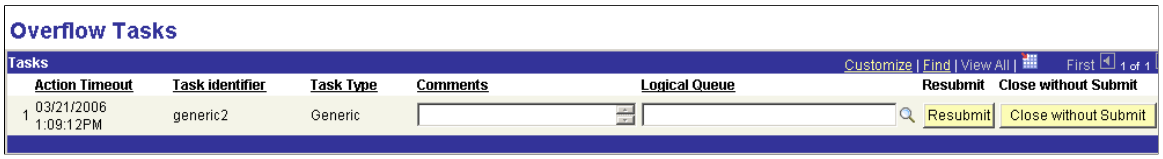
To administer overflow and escalated tasks, use the Overflow Administration (MCF_OVERFLOWL_CMP) and Escalation Administration (MCF_ESCAL_CMP) components.

Administering Overflow Tasks

Access the Overflow Tasks page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **Administration** > **Overflow MCF Tasks**

This example illustrates the fields and controls on the Overflow Tasks page. You can find definitions for the fields and controls later on this page.



Use this page to manage tasks that could not be assigned to an agent within the specified overflow timeout.

Field or Control	*Description*
Action Timeout	Displays the time at which the overflow occurred.
Task Identifier	Displays the task identifier.
Task Type	Types include chat, email, generic, and voice (not supported).
Comments	Enter optional text commentary about the resolution of the task. For example, note that the customer sent follow-up email that was answered by an agent.
Logical Queue	Select a logical queue to which to resubmit this task.

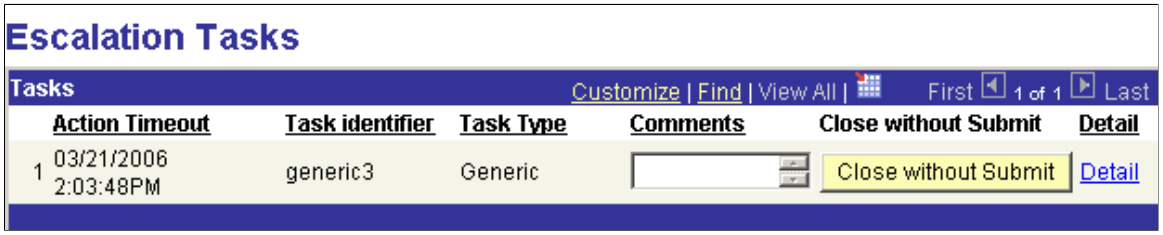
Field or Control	*Description*
Resubmit	Click to send the task to the specified logical queue to retry assignment. Only persistent tasks can be resubmitted; chat cannot be resubmitted.
Close without Submit	Click to close this task without sending it to retry assignment.
Detail	Click to display additional information about the task.

Administering Escalated Tasks

Access the Escalation Tasks page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > Administration > Escalated MCF Tasks (escalation administration)

This example illustrates the fields and controls on the Escalation Tasks page. You can find definitions for the fields and controls later on this page.



Use this page to manage accepted persistent tasks that were automatically unassigned from agents because they were not closed within the specified escalation timeout. Tasks can be closed without submitting them.

Field or Control	*Description*
Action Timeout	Displays when the escalation occurred.
Task Identifier	Displays the task identifier.
Task Type	Types include chat, email, generic, and voice (not currently supported).
Comments	Enter optional text commentary about the task's resolution. For example, note that the customer sent follow-up email that was answered by an agent.

Field or Control	*Description*
Close without Submit	Click to close this task without sending it back to retry assignment.
Detail	Click to view additional information about this escalated task.

Chapter 9

Managing Tasks and Using Chat in PeopleSoft MultiChannel Framework

Managing Tasks with the MultiChannel Console

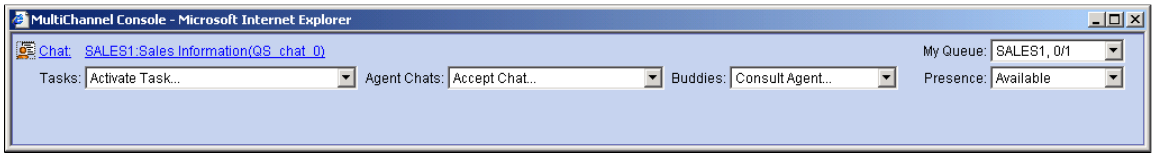
Use the MultiChannel Console to accept, respond to, transfer, and complete tasks and to initiate or join chat sessions.

This section discusses how to use the MultiChannel Console to work with tasks.

Using the MultiChannel Console to Work with Tasks

If your user ID includes security permissions for PeopleSoft MultiChannel Framework, you can access the MultiChannel Console. You can access the MultiChannel Console from the Actions menu.

This example illustrates the fields and controls on the MultiChannel Console. You can find definitions for the fields and controls later on this page.



You cannot launch two MultiChannel Consoles on the same workstation.

You can launch consoles on different workstations using the same user ID, but you cannot log on to physical queues served by the same MCF cluster.

When the MultiChannel Console is launched, it attempts to connect to the real-time event notification (REN) server associated with the last physical queue listed in the **My Queue** drop-down list box. If that REN server is not running, the agent receives an error message. If one or more of the agent's physical queues are associated with a running REN server, the agent can log on by selecting another queue.

Newly assigned tasks appear as a link above the **Tasks** drop-down list box, and display a flashing icon. Click the link to accept the task. Accepted tasks appear in the **Tasks** drop-down list box.

Collaborative chat requests (agent-to-agent chats) may appear as a link above the **Agent Chats** drop-down list box, but all collaborative chat requests (accepted or not) appear in the **Agent Chats** drop-down list box.

Configure the size and initial location of the MultiChannel Console on the agents Window Config (window configuration) page.

Note: If you use Secure Sockets Layer (SSL) security, you may receive a security warning message when your console first opens. You can accept the warning message without compromising SSL security.

Field or Control	*Description*
Tasks	After you accept an assigned task, the task appears here. Activate a task to work on by selecting it. Activating a task brings the window associated with that task to the foreground. If the associated task window is not running, it is launched by the console. A task is removed from the task list when you mark the task as done using one of the following methods: • For email and generic tasks, the application page of the task may include a **Done** button; refer to the application's documentation. • For chat, the task is complete when the chat dialog ends.
Agent Chats	Displays a list of your collaborative chats, both accepted and requested, including a label indicating if the chat is inbound or outbound. Select a chat to activate it. If you are invited to join in a chat session (conference), the conference chat is added to the list. After you accept the conference, the conference is added to the **Tasks** drop-down list box and removed from the **Agent Chats** drop-down list box. The chat is removed from the list when the chat dialog ends.
Buddies	Includes agents that are identified as buddies in your agent configuration. Add buddies on the Buddy List page of the Agent component. To initiate a collaborative chat, select an agent. A chat dialog box displays. **Note:** You can only initiate a collaborative chat with an agent who appears on your buddy list. However, if you appear on another agent's buddy list, that agent can initiate a collaborative chat with you, even if that agent is not one of your buddies.
My Queue	Select a queue to log on to that queue. The drop-down list box includes the physical queues that you can access. To change queues, select a new queue. You are logged off from the old queue and logged on to the selected queue. Add or delete queues on the Queues page of the Agents component.

Field or Control	*Description*
Presence	Displays your current presence and enables selection of a new presence. Select from: • *Active* • *Inactive* Customize presence labels to more closely track agent activity. Edit your presence options on the Presence page of the Agent component. If you select an inactive presence, an icon appears in the upper-left corner of the MultiChannel Console. Changes in agent presence are logged and include the customized presence description. Presence status persists for eight hours from the time of its last change.

You can use hot key combinations, described in the following table, to navigate between MultiChannel Console fields:

Hot Key Combination	*Opens*
ALT + A	Agent chats
ALT + B	Buddies
ALT + P	Presence
ALT + Q	My Queues
ALT + T	Tasks

Communicating with Customers and Agents Using Chat

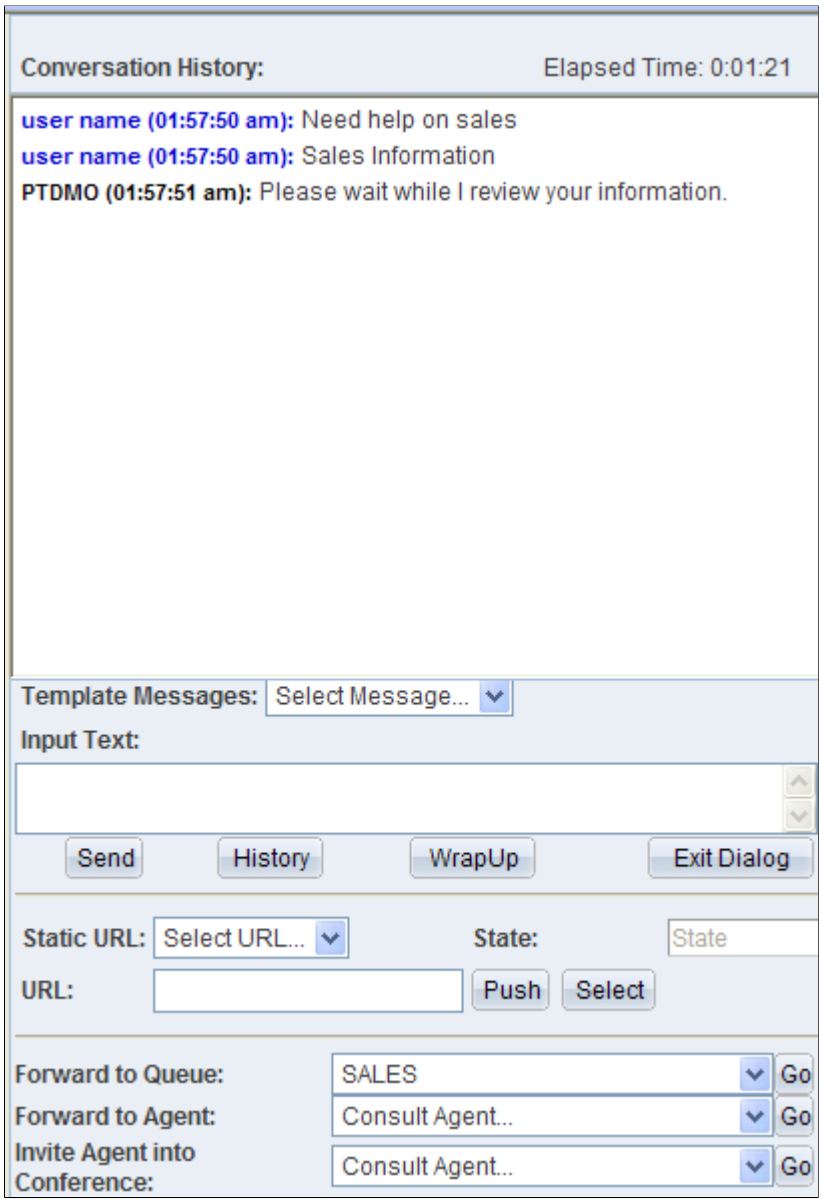
This section describes the chat windows used to communicate with agents and customers.

Using the Agent Chat Window

When you accept a chat task, a chat window opens.

The format of the agent chat window and the contents of its right pane are delivered as part of PeopleSoft MultiChannel Framework. The content of the left pane is determined by application developers, and is passed as a parameter to the InitChat() built-in function.

This example illustrates the fields and controls on the Agent Chat window. You can find definitions for the fields and controls later on this page.



Configure the size and initial location of the agent chat window on the agent's Window Config (window configuration) page.

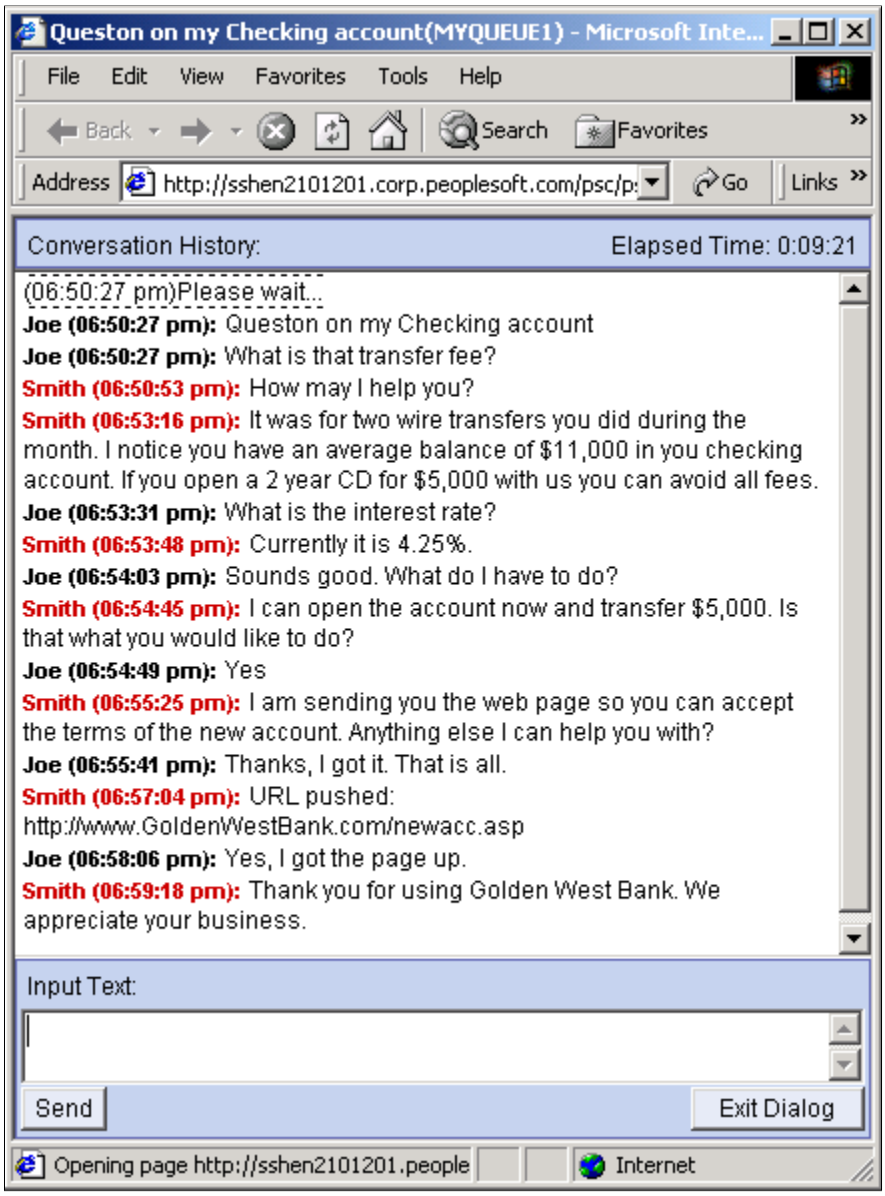
Field or Control	*Description*
Conversation History	Lists progress of the chat, line by line. If chat logging is enabled, the conversation is recorded in the database chat log by the MCF log server. View the chat log on the Chat Log page. If accessibility features are not turned off in the My Personalizations component, an additional text box appears below the conversation history. The most recent customer input appears in the secondary text box, which can be read by screen reading software.
Template Messages	Send the customer a standard message by selecting one from the list. The message text appears in the **Input Text** text box. Click **Send** to send the message. Edit template messages for each queue on the queue Chat Responses page.
Input Text	To respond to the customer, enter text, and then click **Send** or press **Enter.** The maximum size of the text that can be sent at one time is 4096 bytes (4 kilobytes).
Send	Click to send the contents of the **Input Text** text box.
Exit Dialog	Click to end the chat. The chat window remains open, enabling you to return to the page for follow-up work after the customer exits the dialog.
Static URL and Push	To send a static URL to the customer, select a URL name, then click **Push.** When a URL is pushed to a customer, a browser window for that URL is launched on the customer's workstation.
URL	Displays static and grabbed URLs.
Select	Click to launch an application page, from which a URL can be returned to populate the URL field. The format of the URL wizard page that appears when you click **Grab** is defined by application developers. PeopleSoft supplies a sample URL wizard page. See Using the URL Wizard.

Field or Control	*Description*
Forward to Queue	To forward the current chat to another queue, select the queue. You can only forward to another physical queue, and that physical queue must be served by the same MCF cluster as the physical queue that assigned the chat.
Invite Buddy	To request that another agent join the chat, select the agent and click **Go.** The buddy must be logged on to a physical queue that is served by the same MCF cluster as the physical queue that sent you this chat. Chat conferencing is only supported on customer-initiated chat sessions.

Using the Customer Chat Window

When a customer initiates a chat, a chat window appears:

This example illustrates the fields and controls on customer chat window. You can find definitions for the fields and controls later on this page.



The format and content of the customer chat window is delivered as part of PeopleSoft MultiChannel Framework. The customer chat window includes a conversation history text box, input text box, and **Send** and **Exit Dialog** buttons.

If accessibility features are not turned off in the My Personalizations component, an additional text box appears below the conversation history. The most recent agent input appears in the secondary text box, which can be read by screen reading software.

Field or Control	*Description*
Input Text	The customer enters text here. The maximum size of the text that can be sent at one time is 4096 bytes (4 kilobytes).
Send	Click to send the input text to the agent.
Exit Dialog	Click to end the chat and close the chat window.

Note: The agent collaborative chat window is substantially the same as the customer chat window.

Chapter 10

Using PeopleSoft MCF Broadcast and Working with Sample Pages

Using PeopleSoft MCF Broadcast

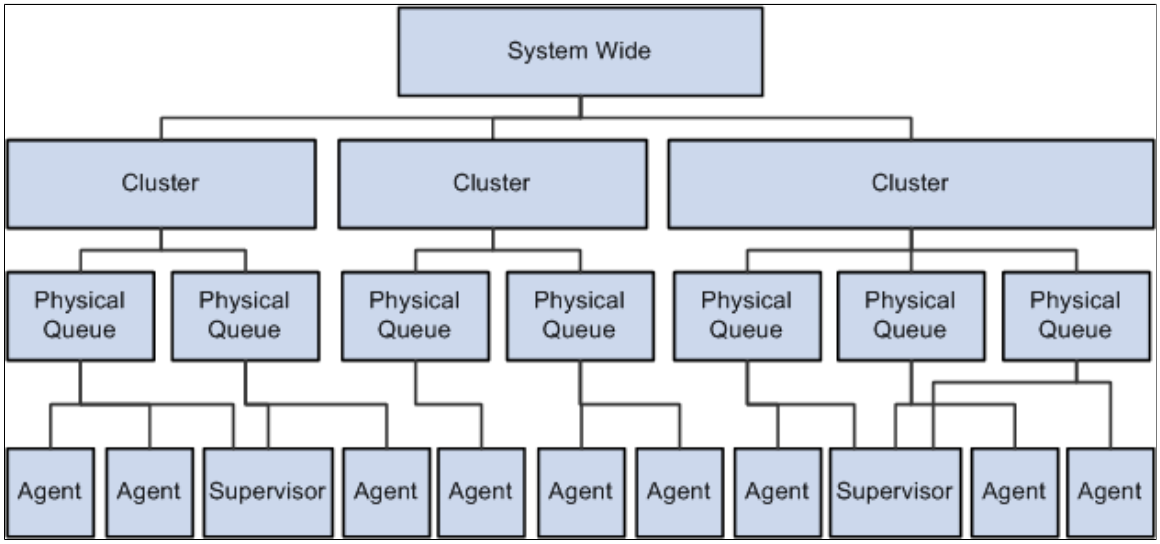
Use the broadcast function to broadcast a notification message. This function is typically used by a supervisor to send a notification message to specific recipients based on the parameters that are provided by the sender. Broadcast notifications can be sent system-wide, cluster-wide, queue-based, task-based, or activity-based.

System-wide broadcast notifications can be sent only using the PeopleCode API to all logical queues on the system. The same notification using JSMCAPI is sent to all the physical queues on the cluster.

Cluster-wide broadcast enables a user to broadcast to an entire audience that is logged on or subscribed to this broadcast topic on a particular cluster. A cluster broadcast can be configured to target a specific audience by specifying a particular queue, a channel, or an agent log state in the broadcast call.

A queue-based broadcast can send a notification to all agents on a particular queue. A channel-type broadcast is received by agents serving a particular channel. An agent working on multiple channels can receive broadcast that is meant for all those channels. An agent login state or activity-based broadcast allows a broadcast notification to be sent to all agents in a particular state.

The following diagram illustrates a cluster-wide broadcast of notifications.



Understanding JSMCAPI Broadcast

A JSMCAPI console primarily operates on physical queues. Whenever an agent or a supervisor uses a JSMCAPI console, the agent chooses a physical queue and logs onto it. To broadcast a message, a console

typically uses this physical queue as a parameter to determine the target cluster. JSMCAPI broadcast works on the following restrictions:

- JSMCAPI broadcast function does not provide any routing logic because it operates within a limited set of physical queues that are assigned to the agent or supervisor.

  The agent can choose any physical queue from the assigned queue list without logging onto any particular physical queue.

- The supervisors or the users of JSMCAPI broadcast can use only the physical queues on their list to send a broadcast notification.

- JSMCAPI works within the confine of a single cluster and does not have any knowledge or access to other clusters in the system.

Implementing MCF Broadcast

To enable you to implement broadcast, MultiChannel Framework provides a PeopleCode built-in function and a JSMCAPI interface for JSMCAPI users (agents and supervisors) to send broadcast notifications.

Note: The broadcast notifications are displayed only in the third-party sample pages.

Subscribing and Publishing Broadcast

The JSMCAPI broadcast function is a specialized publish call to the REN server. Both JSMCAPI and PeopleCode publish to physical-queue broadcast topic (such as /Broadcast/SALES1), and the subscribers always subscribe to broadcast topic for the physical queues to which they were assigned.

To subscribe, use the following function:

```
BroadcastSubscribe(cluster,queue,task,state,presence,method)
```

To publish, use the following function:

```
MCFBroadcast(cluster,queue,task,state,presence,message,securitylevel,
importancelevel, senderid, NameValuePairString
```

To unsubscribe, use the following function:

```
void broadcastUnsubscribe(type)
```

where <type> determines the type of broadcast, system-wide, queue-wide, or agent-wide.

Related Links

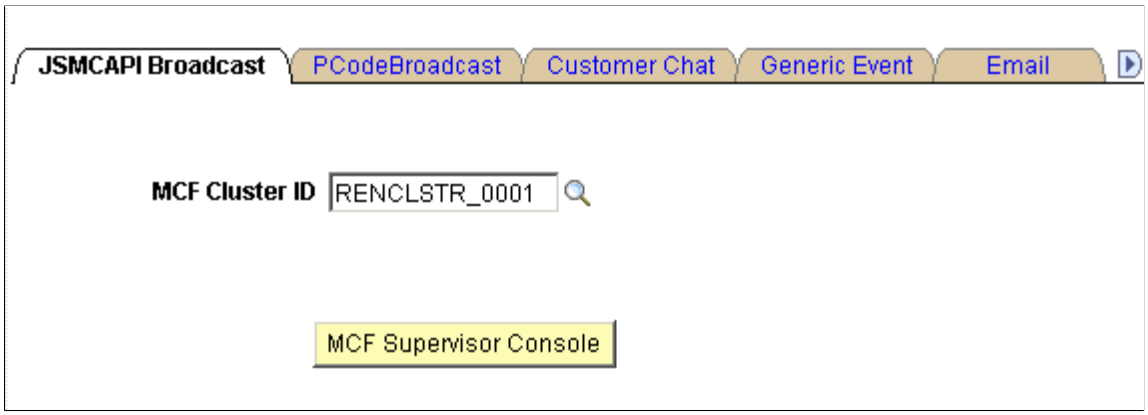
broadcastSubscribe
broadcastUnsubscribe

Using JSMCAPI Broadcast with MCF Supervisor Console

Access the JSMCAPI Broadcast page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **MCF Sample Pages** > **JSMCAPI Broadcast**

This example illustrates the fields and controls on the JSMCAPI Broadcast page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
REN Server Cluster ID	Select the REN server cluster on which to test the MCF Supervisor console. **Note:** The MCF Supervisor console is specific to the REN cluster Id selected.
MCF Supervisor Console	Click to initiate the supervisor console. **Note:** To demonstrate the MCF Supervisor console, you must have MCF servers, both queue and log servers, running and communicating with the specified REN server cluster.

After you click **MCF Supervisor console,** a new browser window appears that displays the sample supervisor console.

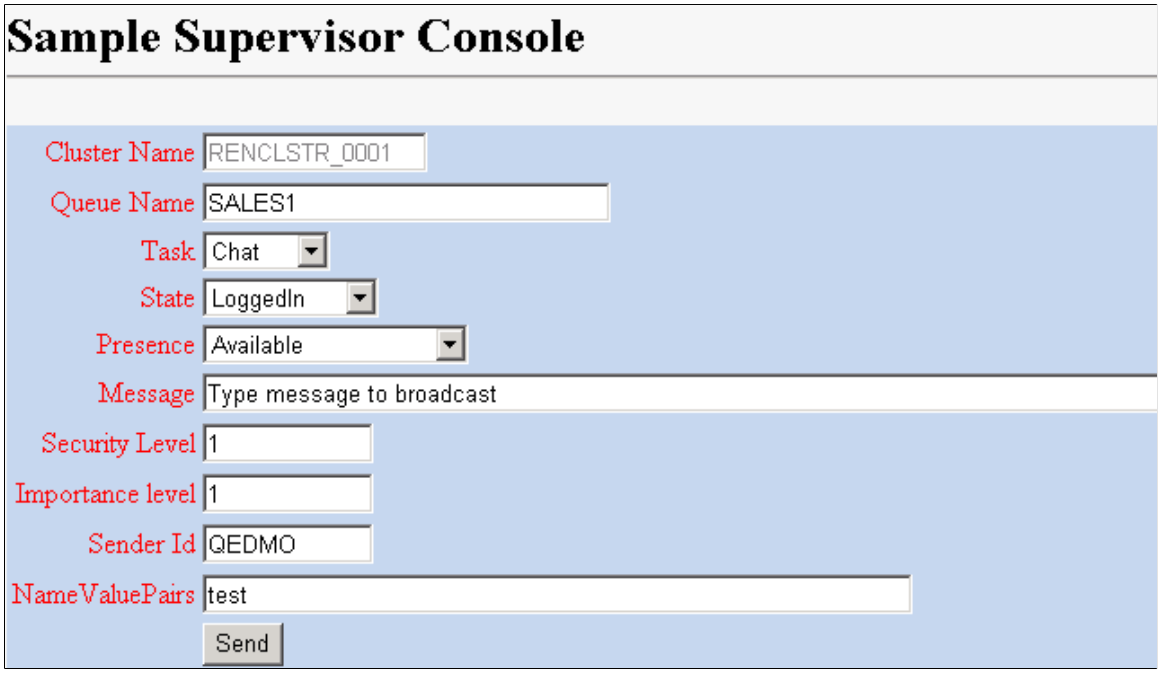
Note: To enable the new browser window to appear, disable any pop-up blocking software for your browser.

MCF Supervisor Console

Access the MCF Supervisor console using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > MCF Sample Pages. Choose an MCF Cluster ID and Click on the MCF Supervisor Console to open the MCF Supervisor Console.

This example illustrates the fields and controls on Sample Supervisor Console. You can find definitions for the fields and controls later on this page.

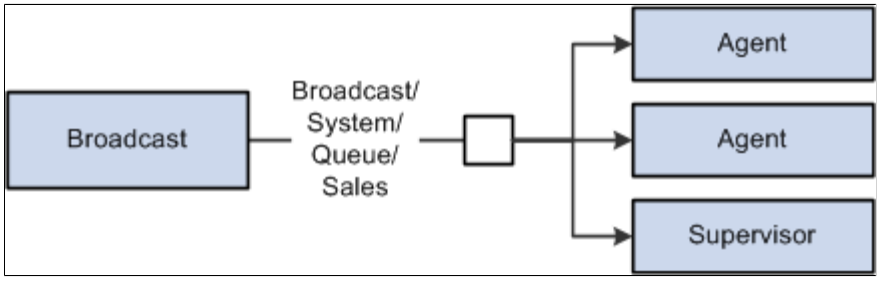


Field or Control	*Description*
Cluster Name	Displays the REN server cluster on which to test broadcast.
Queue Name	Enter the name of the queue that is in the specified cluster. **Note:** Enter the queue name only when you want to broadcast the message to a particular queue.
Task	Select the task. Values are *email, chat, voice, generic,* and *none.*
State	Select the state of the agent. **Note:** If the state of the agent is *LoggedIn* when the broadcast message is being sent by the supervisor, the agent receives the broadcast message using Agent console.
Presence	Select the presence. **Note:** If the agent is logged out, the presence should be inActive. The broadcast message is received only when the agent is currently logged in to the queue on which broadcast is sent.

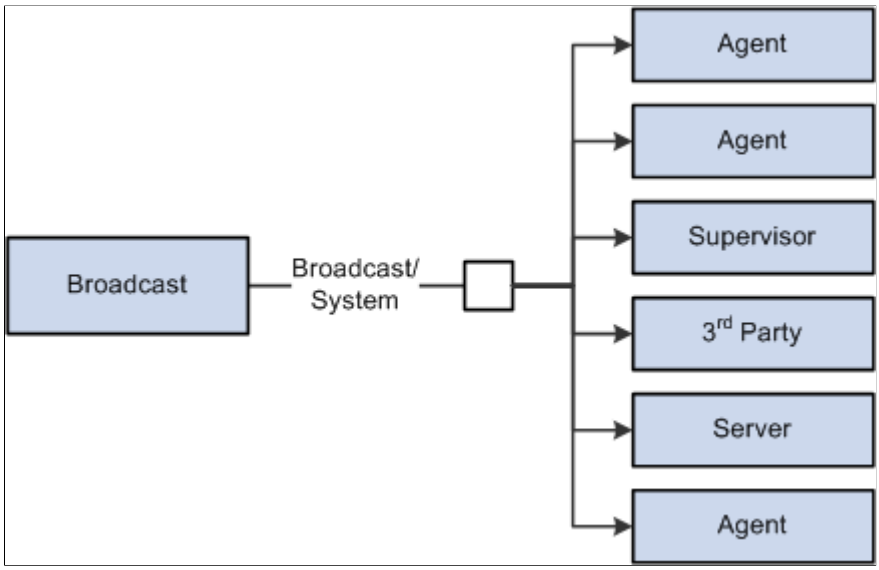
Field or Control	*Description*
Message	Enter the message that you want to broadcast. **Note:** To view the broadcast message that is sent by the supervisor, use the Agent console sample page. Select the same REN cluster that is used by the supervisor to broadcast, and the broadcast message is displayed on the Agent console. See Using the Agent Console Page.
Security Level	(Optional) Enter the security level.
Importance Level	(Optional) Enter the importance level.
Sender Id	(Optional) Enter the sender ID. Use this option only when you want the sender ID to appear as something other than the ID that actually sent the message.
NameValue Pairs	Enter the name and value pairs to configure your data. **Note:** These name-value pairs are concatenated as a string. You can define any name-value parameters for your application and send them in the name-value pair string. The JSMCAPI passes the same as a string to the console. Your particular application needs to process that string accordingly.

Note: For a system-wide or a cluster-wide broadcast, the broadcast message from the supervisor is sent to all agents, as well as the supervisors who are logged in to the cluster that is specified by the supervisor. The system-wide broadcast is same as cluster-wide broadcast because a JSMCAPI supervisor knows only the cluster it is operating on.
For a queue-wide broadcast, the broadcast message from the supervisor is sent to all agents who are currently logged in to the queue that is used by the supervisor to broadcast the message. This assumes that the queue is in the same cluster as the supervisor who is sending the broadcast message.
For agent-wide broadcast, the broadcast message will be sent to all agents who are currently logged to any queue in the same cluster as the supervisor who is sending the message.

The following diagram illustrates queue-wide broadcast of notifications.



The following diagram illustrates cluster-wide broadcast of notifications.



The following table describes the combinations:

Type of Broadcast	***Cluster***	***Queue***	***Agent Login State***	***Agent Activity State***	***Broadcast Audience and Description***
JSMCAPI	Not specified	Not specified	Not specified	Not specified	This is a cluster-wide broadcast. All agents on Cluster _1 will receive the broadcast.
JSMCAPI	Cluster_1	Not specified	Not specified	Not specified	Same as the preceding. All agents on Cluster _1 will receive the broadcast.
JSMCAPI	Cluster_1	SALES	Not specified	Not specified	All agents subscribing to SALES will receive the broadcast. This assumes that SALES is on Cluster_1.

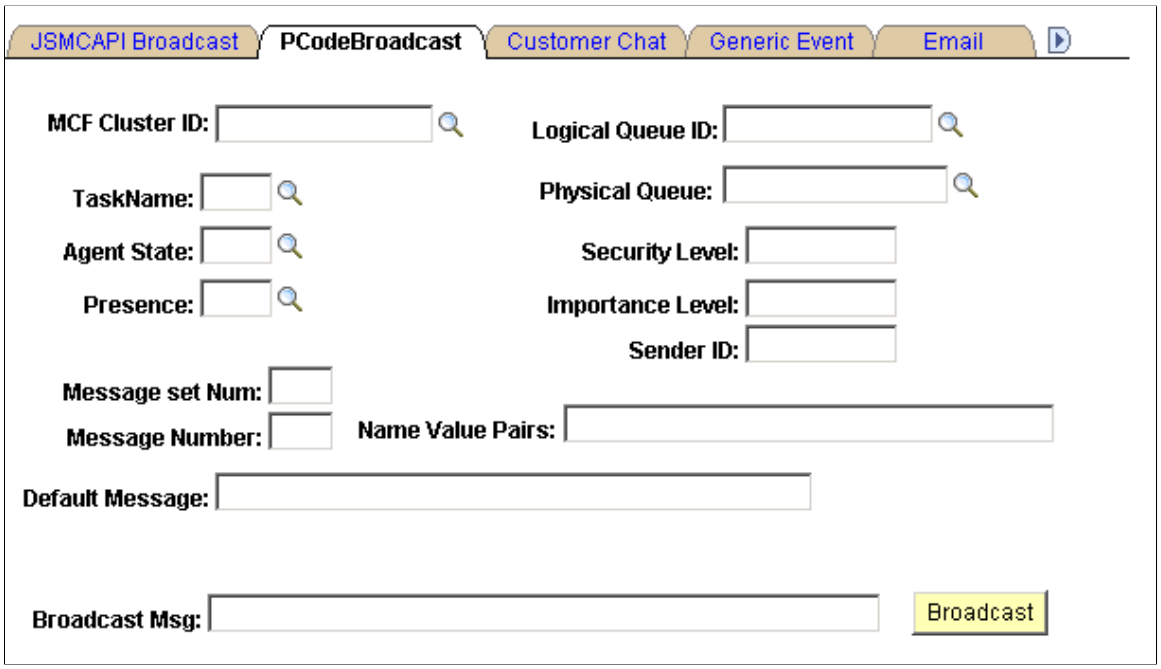
Type of Broadcast	*Cluster*	*Queue*	*Agent Login State*	*Agent Activity State*	*Broadcast Audience and Description*
JSMCAPI	Cluster_1	SALES	Logged	Not specified	All agents who are logged (active and inactive) to sales will receive the broadcast. This assumes that SALES is on Cluster_1.

Using PeopleCode Broadcast

Access the PCodeBroadcast page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > MCF Sample Pages > PCodeBroadcast

This example illustrates the fields and controls on the PCodeBroadcast page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
MCF Cluster ID	Select the target cluster receiving the broadcast message.
Logical Queue ID	Select the name of the target logical queue.

Field or Control	*Description*
Physical Queue ID	Select the name of the target physical queue.
TaskName	Select the task. Select from *email, chat*, *voice*, *generic* or *none.*
Agent State	Select the state of the agent. **Note:** The agent receives the broadcast message only if the agent is logged in.
Presence	Select the presence. **Note:** If the agent is logged out, the presence should be *inActive.* All agents who are currently logged in receive the broadcast message.
Message Set Number	Select the message set number if you want to broadcast a message from the Message Catalog.
Message Number	Select the message number in the message set.
Default Message	Enter the default message. If no message is found in Message Catalog with the preceding message number, the text of the default message is used for the broadcast instead.
Security Level	(Optional) Enter the security level.
Importance Level	(Optional) Enter the importance level.
Sender Id	Enter the sender ID if you want the sender that is displayed with the notification to be an ID other than the one that sent the broadcast.
Message	Enter the message that you want to broadcast. **Note:** To view the broadcast message that is sent by the supervisor, use the Agent console sample page. Select the same REN cluster as used by the supervisor to broadcast, and the broadcast message is displayed on the Agent console. See Using the Agent Console Page.

Field or Control	*Description*
NameValue Pairs	Enter the name and value pairs to configure your data. **Note:** These name-value pairs are concatenated as a string. You can define any name-value parameters for your application and send them in the name-value pair string. The JSMCAPI passes the same as a string to the console. Your particular application needs to process that string accordingly

Note: For a system-wide PeopleCode Broadcast, the broadcast message from the supervisor is sent to all agents, as well as the supervisors on all active REN clusters.
For a cluster-wide PeopleCode Broadcast, the broadcast message from the supervisor is sent to all agents, as well as the supervisors on the specified REN clusters.
For a queue-wide broadcast, the broadcast message from the supervisor is sent to all agents who are currently logged in to the queue that is specified by the queue ID in all clusters if the MCF cluster ID is not specified. If the cluster ID is specified, all agents that are logged into the queue that is specified by the queue ID in the cluster that is specified by the cluster ID receive the broadcast message.
For agent-wide broadcast, if both the cluster ID and queue ID are specified, the broadcast message is sent to all agents that are currently logged into the queue that is specified by the queue ID. This assumes that the queue is in the cluster that is specified by the cluster ID. However, if only the cluster ID is mentioned, all agents that are logged and active on any queue on that cluster receive the broadcast message.

The following table describes the combinations:

Type of Broadcast	*Cluster*	*Queue*	*Agent Login State*	*Agent Activity State*	*Broadcast Audience and Description*
PeopleCode	Not specified	Not specified	Not specified	Not specified	This is a true system-wide broadcast. All agents on all active REN clusters receive the broadcast.
PeopleCode	Cluster_1	Not specified	Not specified	Not specified	All agents on Cluster_1 receive the broadcast.
PeopleCode	Cluster_1	SALES	Not specified	Not specified	All agents subscribing to SALES receive the broadcast. This assumes that SALES is on Cluster_1.

Type of Broadcast	*Cluster*	*Queue*	*Agent Login State*	*Agent Activity State*	*Broadcast Audience and Description*
PeopleCode	Not specified	SALES	Logged	Not specified	All agents that are logged (active and inactive) to SALES (all clusters with physical queues that are associated with SALES) receive the broadcast.
PeopleCode	Cluster_1	SALES	Logged	Active	All agents that are logged into SALES and are in active state receive the broadcast. This assumes that SALES is on Cluster_1.
PeopleCode	Cluster_1	Not specified	Logged	Active	All agents that are logged and active on any queue on Cluster_1 receive the broadcast.
PeopleCode	Cluster_1	Not specified	Not specified	Active	All agents that are logged and active on any queue on Cluster_1 receive the broadcast. This is the same as the preceding because only agents that are logged in can be active.

Related Links

“MCFBroadcast” (PeopleCode Language Reference)

Viewing Broadcast Logs

To view broadcast log, use the Broadcast log page.

See Viewing Broadcast Logs.

Working with Sample Pages

To demonstrate MCF tools and functionality, use the MCF Sample Pages (MCF_DEMO_CMP) component.

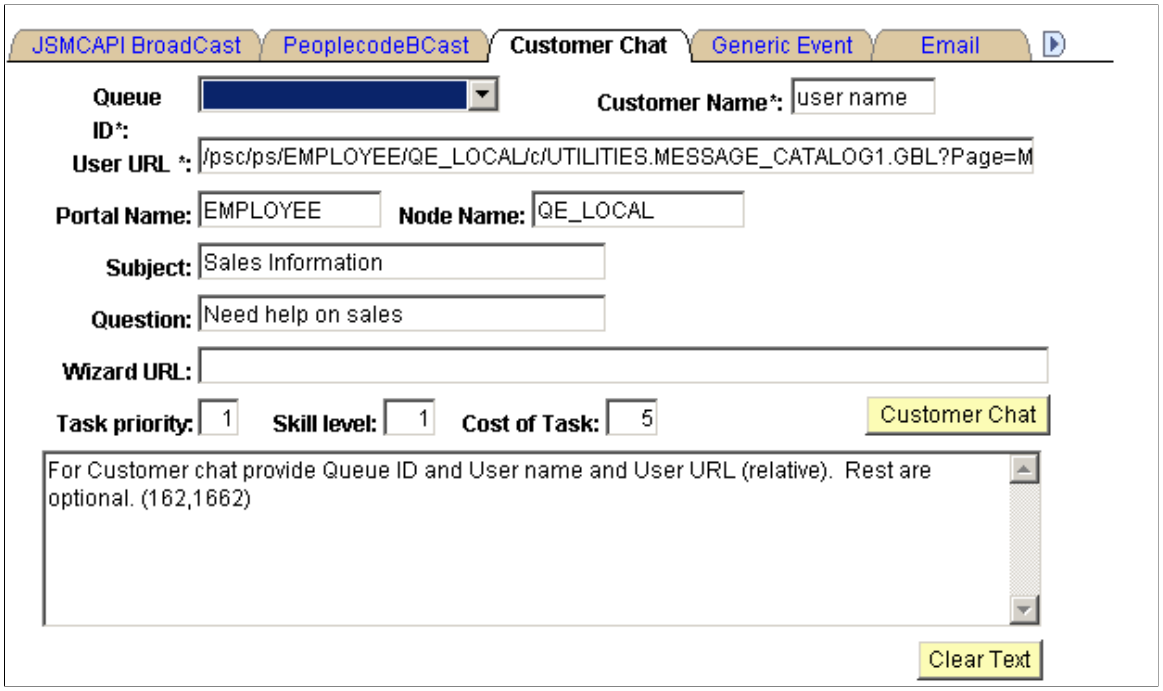
Using the Customer Chat Sample Page

Access the Customer Chat page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **MCF Sample Pages** > **Customer Chat**

Note: This page is available using both queue server and third-party routing server.

This example illustrates the fields and controls on the Customer Chat page.



This page demonstrates the InitChat() built-in function, including required and optional parameters.

This page is not intended for production use. On a typical application page, the developer includes a Live Help or Customer Chat button that calls the InitChat() built-in function and passes required parameters, including the context of the chat request. The sample customer chat page demonstrates values that can be included in the InitChat() parameters. The user may not be prompted for any information.

To run a sample chat:

1. Access the MultiChannel Console.
2. Open the Customer Chat page.
3. Open the **Queue ID** drop-down list box and select the queue that you are logged on to.

The other fields are automatically populated with values that generate a sample chat session. You can change the values as long as valid required values are included.

The User URL is a required field that contains the application page URL to send to the agent chat browser.

The optional Wizard URL field represents a grab URL; the actual page that appears is determined by InitChat() parameters.

The text box displays error messages and additional information for using the sample page.

4. Click **Customer Chat.**

   The customer chat window appears.

5. On the MultiChannel Console, click the flashing chat notification icon to accept the chat session.

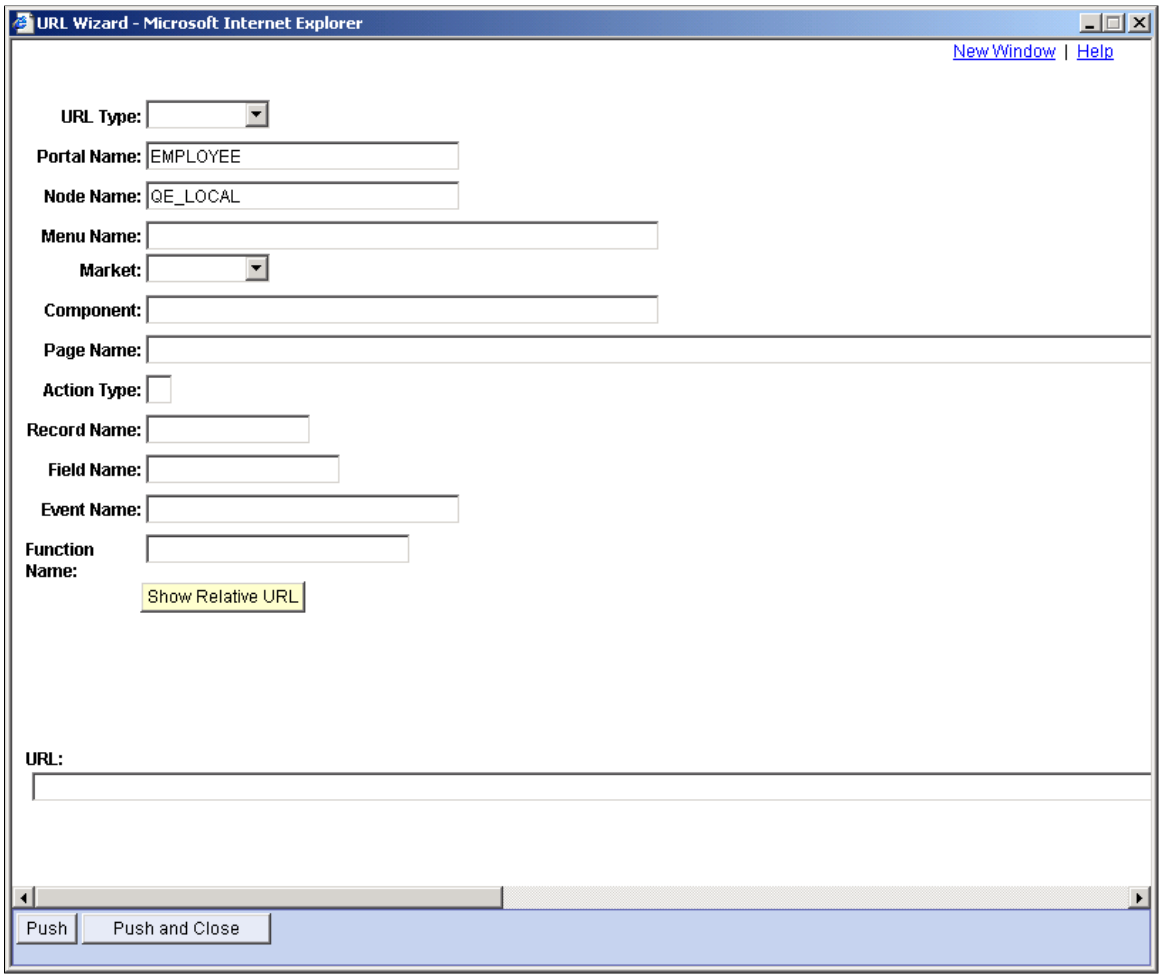
   The agent chat window appears.

   You can enter text as both agent and customer to demonstrate chat functionality.

Using the URL Wizard

Click **Select** from the agent chat window during a chat session to open the URL wizard:

This example illustrates the fields and controls on the URL Wizard page. You can find definitions for the fields and controls later on this page.



Use the wizard to form a URL to send to the customer. The URL Wizard page is a template that demonstrates constructing a URL and sending it to a customer. The URL wizard can be used as is or as the basis for developing URL generation for your application.

Field or Control	*Description*
URL Type	Select the type of URL that is being accessed. Values are: • *Component:* Generates a PeopleSoft Pure Internet Architecture component URL. Component URL generation uses the PeopleCode built-in function **GenerateComponentContentRelURL**. See “GenerateComponentContentRelURL” (PeopleCode Language Reference). • *External:* Not currently used. • *Gen URL:* Not currently used. • *iScript:* Generates a PeopleSoft Pure Internet Architecture iScript URL. iScript URL generation uses the PeopleCode built-in function **GenerateScriptContentRelURL.**. See “GenerateScriptContentRelURL” (PeopleCode Language Reference). Fields that are not required by the selected URL type are not available.
Show Relative URL	Click to display the relative URL that is generated by the values that you have entered into the page's fields.
URL	Displays the generated relative URL.
Push	Click to send the generated relative URL to the customer's chat window. The **Push** button demonstrates functionality that must be included in an application-specific URL wizard.
Push and Close	Click to send the generated relative URL to the customer's chat window and close the URL wizard. The **Push and Close** button demonstrates functionality that must be included in an application-specific URL wizard.

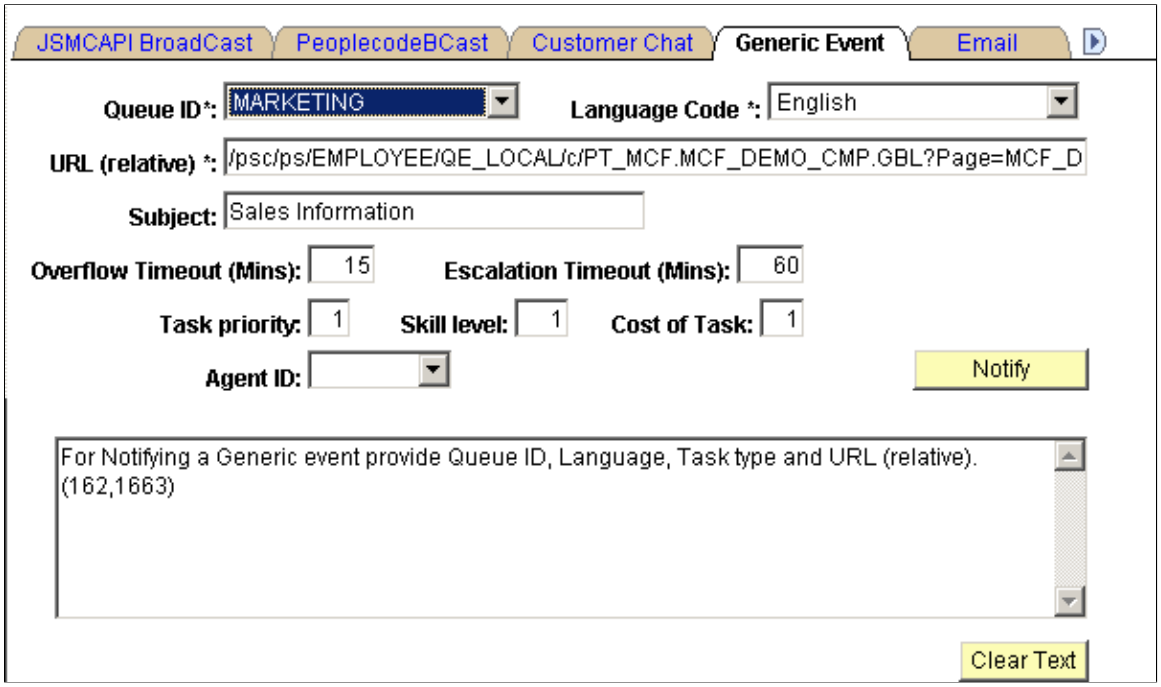
Using the Generic Event Sample Page

Access the Generic Event page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **MCF Sample Pages** > **Generic Event**

Note: This page is available using both queue server and third-party routing server.

This example illustrates the fields and controls on the Generic Event page.



This page demonstrates a generic persistent event using the EnQueue() and NotifyQ() built-in functions, including required and optional parameters.

This page is not intended for production use. On a typical application page, the developer includes an Enqueue or similar button that calls the EnQueue() built-in function and passes required parameters, including the context of the request. The sample customer chat page demonstrates values that can be included in the EnQueue() parameters. The user may not be prompted for any information.

To run a sample generic event:

1. Access the MultiChannel Console.
2. Open the Generic Event page.
3. Open the **Queue ID** drop-down list box, and select the queue that you are logged on to.
4. Open the **Agent ID** drop-down list box, and select the agent ID that you are logged on with.
5. Select *Generic* for the task type.
6. Click **Notify.**
7. On the MultiChannel Console, click the flashing event notification icon to accept the event.

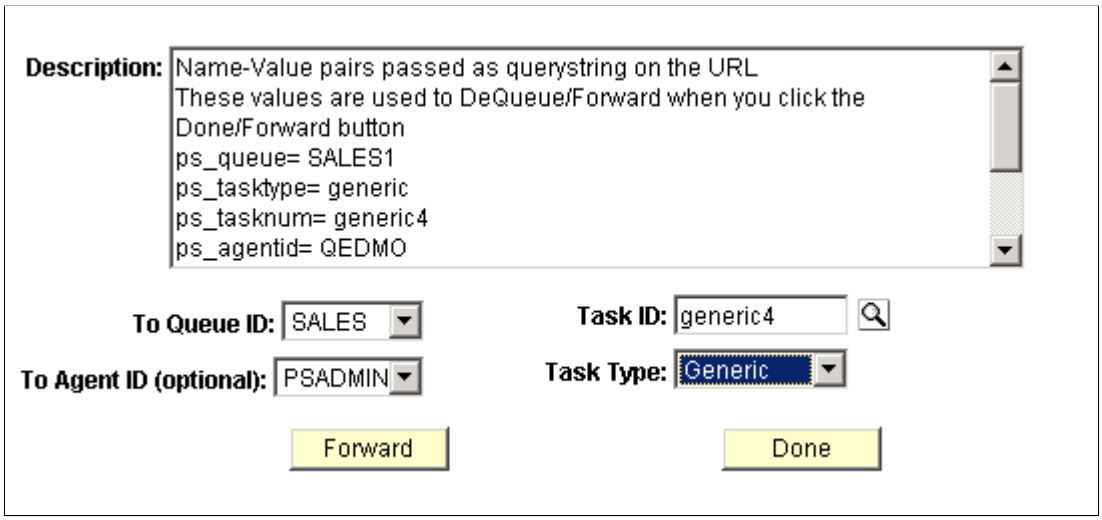
   The Generic Event window appears.

Using the Generic Event Window

The format of the generic event window is determined by application developers. PeopleSoft supplies a sample generic event window to demonstrate the available functionality, including the DeQueue() built-in.

Configure the size and initial location of the generic event window on the Agent Window Configuration page:

This example illustrates the fields and controls on the Generic Event window. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Description	Displays text from the generic event.
To Queue ID	Select a queue to which the generic event will be forwarded.
Task ID	Select the task ID of the generic event to forward.
To Agent ID	Select the agent ID to whom the generic event will be forwarded.
Task Type	Select the task type of the generic event to be forwarded.
Forward	Click to send the generic event to the selected agent or queue. This button demonstrates the functionality of the Forward() built-in function.
Done	Click to notify the real-time event notification (REN) server that you are done with this task. A message asks if you want to close the window. The **Done** button demonstrates the functionality of the DeQueue() built-in function.

Related Links

“DeQueue” (PeopleCode Language Reference)

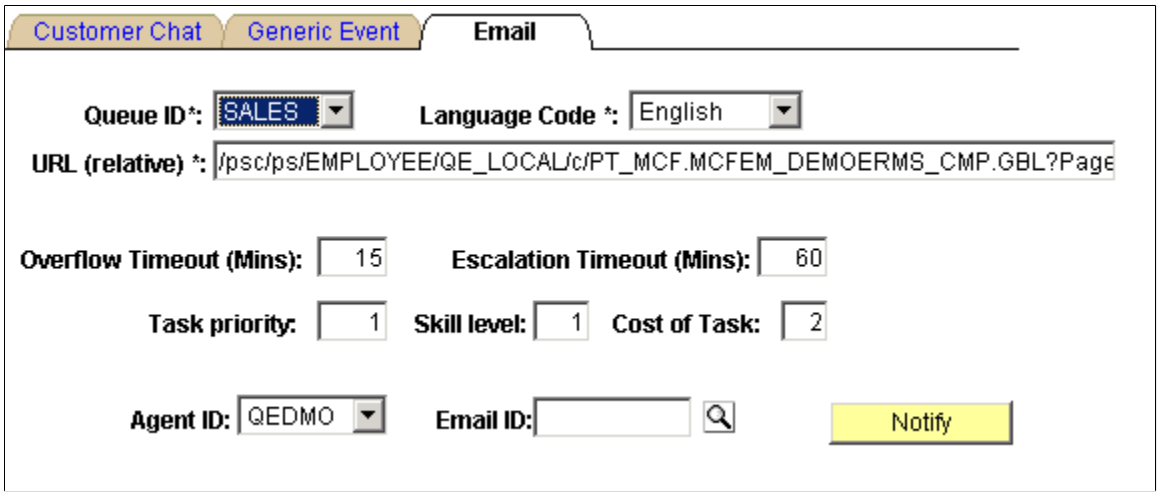
Using the Email Sample Page

Access the Email page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **MCF Sample Pages** > **Email**

Note: This page is available using both queue server and third-party routing server.

This example illustrates the fields and controls on the Email page.



The email sample pages are intended for demonstration purposes and should not be used in production.

EnQueue() and NotifyQ() can also be called from PeopleSoft Application Engine batch programs.

See “EnQueue” (PeopleCode Language Reference) and “NotifyQ” (PeopleCode Language Reference).

Note: For you to fully demonstrate the functionality of the Email sample page, an email must be read and written to the email database using the GetMail - Server sample page.

To process a sample email:

1. Open the **Queue ID** drop-down list box and select the queue that you are logged on to.
2. Open the **Agent ID** drop-down list box and select the agent ID that you are logged on with.
3. Click **Notify.**
4. On the MultiChannel Console, click the flashing email notification icon to accept the email.

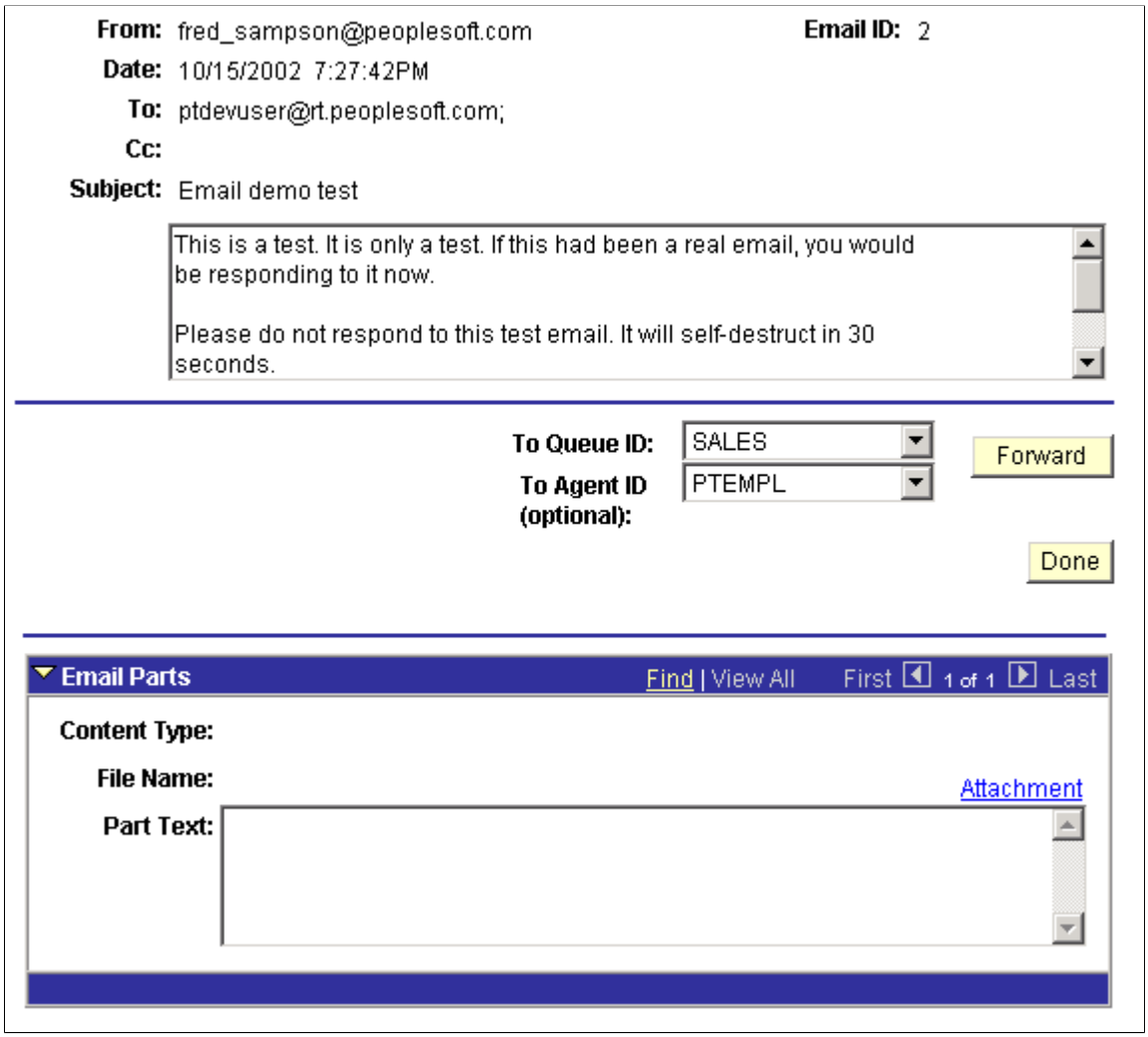
   The email window appears.

Using the Email Window

The format of the email window is determined by application developers. PeopleSoft supplies a sample email event window to demonstrate available functionality.

Configure the size and initial location of the email event window on the agent's Window Configuration page.

This example illustrates the fields and controls on the Email window. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
To Queue ID	To forward the email to another queue, select a queue ID from the drop-down list box and click **Forward.**
To Agent ID	To forward the email to another agent, select an agent ID from the drop-down list box and click **Forward.**
Forward	Click to forward the email to the specified queue or agent. This button demonstrates the functionality of the Forward() built-in function.
Done	Click to quit the email and remove it from the queue. The **Done** button demonstrates the functionality of the DeQueue() built-in.

Email Parts

If an email has been divided into parts stored in the email database, or has an attachment, they can be accessed here.

Related Links

Demonstrating the Email Channel

“Understanding PeopleSoft MultiChannel Framework Mail Classes” (PeopleCode API Reference)

Using and Demonstrating JSMCAPI

To demonstrate MCF consoles and tools that use JSMCAPI, use the CTI Sample Pages (PT_CTI_DEMOOUTB) and MCF Sample Pages (MCF_DEMO_CMP) components.

Understanding JSMCAPI

The JSMCAPI enables custom configuration of MCF consoles (including the CTI console), MCF functionality on a PeopleSoft Pure Internet Architecture page, and queue and agent monitoring. For example, developers can create supervisor dashboards with which to monitor activity on their channels of interest, or developers can modify consoles according to their company's business requirements. The JSMCAPI builds on the REN JavaScript client. JSMCAPI uses standard JavaScript.

PeopleSoft provides sample pages demonstrating the functionality that is enabled by JSMCAPI. These pages are for demonstration purposes only, and should not be used in a production environment.

JSMCAPI is delivered with PeopleTools; you do not need a separate installation. The jsmcapi.js file is located in the <PIA_HOME>\webserv\<domain>\applications\peoplesoft\PORTAL\ps\pMCF folder.

Understanding Reason Objects and Reason Codes

PeopleSoft MCF provides reason codes to explain any error or an event that has happened. The third party vendors uses the same reason codes as provided by PeopleSoft MCF to create reason objects that they send with the events. PSMCAPI internally parses the event, gets the reason code, and sends it to JSMCAPI with the event. The CTI sample console or third-party multichannel console looks up a mapper table that maintains the mapping of the reason codes to the Message Catalog entries and displays the reason code messages associated with the error or event that was sent by PSMCAPI.

To display the reason code messages, the sample console picks up all the reason codes and reason messages from the Message Catalogue. Whenever an error occurs, the JavaScript function will extract the reasoncode from the reason object in event and look for the reason message. Then, the JavaScript code prepares the reason message from the fields of reason object and displays the reason message in a new small window.

Reason codes can also be send with requests from JSMCAPI side. The JavaScript function will prompt for the reason codes and any extra data as part of reason.

The following table lists the reason codes and their corresponding reason messages:

Reason Code	*Description/Message Entry*
0	System or general error has occurred.
1	See provider for details.
2	Extension or agent is busy.
3	No answer.
4	Task is in transfer.
5	Task is in conference.
6	Task is abandoned.
7	Searching for an agent.
8	Incoming task.
9	Task is assigned.
10	Connection is established.
11	User or Task data is updated.
12	Request is completed successfully.
13	State of the agent, group, or extension changed.
14	New task is initiated.
15	Conference is not successful.
16	Transfer is not successful.
17	Retrieve is not successful.
18	Forward is not successful.

Reason Code	*Description/Message Entry*
19	Reject is not successful.
20	Revoke is not Successful.
21	Task is revoked.
22	Task is withdrawn.
23	Do Not Disturb is turned on.
24	Do Not Disturb is turned off.
25	Forwarding of calls is set.
26	Forwarding of calls is canceled.
27	Phone is on hook.
28	Phone is off hook.
29	New call is initiated.
30	Call is put on hold.
31	Call is parked.
32	Call is cleared.
33	Call is alternated between hold and active states.
34	Call is taken off hold.
35	Hold is not successful.
36	Invalid session.
37	Invalid State.

Reason Code	*Description/Message Entry*
38	Incorrect Data.
39	Unsupported Feature.
40	Configuration Error.
41	Server is on.
42	Server is off.
43	Agent is Busy.
44	Unexpected Error.

The following table maps the reason codes to the entries in the Message Catalog:

Message Number *MCFMSGNUM*	*Message Set Number* *MCFMSGSETNUM*	*Reason Code Number* *REASONCODENO*
2790	162	0
2791	162	1
2792	162	2
2793	162	3
2794	162	4
2795	162	5
2796	162	6
2797	162	7
2798	162	8

Message Number ***MCFMSGNUM***	***Message Set Number*** ***MCFMSGSETNUM***	***Reason Code Number*** ***REASONCODENO***
2799	162	9
2800	162	10
2801	162	11
2802	162	12
2803	162	13
2804	162	14
2805	162	15
2806	162	16
2807	162	17
2808	162	18
2809	162	19
2810	162	20
2811	162	21
2812	162	22
2813	162	23
2814	162	24
2815	162	25
2816	162	26

Message Number ***MCFMSGNUM***	***Message Set Number*** ***MCFMSGSETNUM***	***Reason Code Number*** ***REASONCODENO***
2817	162	27
2818	162	28
2819	162	29
2820	162	30
2821	162	31
2822	162	32
2823	162	33
2824	162	34
2825	162	35
2826	162	36
2827	162	37
2828	162	38
2829	162	39
2830	162	40
2831	162	41
2832	162	42
2833	162	43
2834	162	44

Related Links

Understanding JSMCAPI Classes
JSMCAPI Classes

Common Elements Used in This Section

This section discusses common elements used on the CTI sample pages.

Common Elements Used in CTI Sample Pages

The following common elements are used in CTI sample pages.

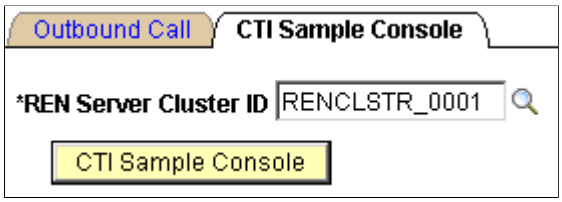
Field or Control	*Description*
Agent Id	Select the CTI or MCF agent's agent ID. In the case of a CTI agent, the agent ID may be different from the agent's user ID.
g/e/c (generic, email, chat)	Represents three task types: generic, email, chat.
	Click to initiate the action that is selected in the associated drop-down list box.
MCF Cluster ID	Select the MCF cluster to be monitored or on which to test the sample console.
Physical Queue	Select the physical queue to be monitored. Each MCF logical queue can comprise one or more physical queues. Because each physical queue is associated with an MCF cluster, only physical queues can be monitored.
State	Displays a value indicating the state of the associated element.
Statistics	Display statistics that are generated by the CTI server.
User Id	Displays the CTI agent's PeopleSoft user ID.

Using the CTI Sample Console

Access the CTI Sample Console page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **CTI Configuration** > **Sample CTI Pages** > **CTI Sample Console**

This example illustrates the fields and controls on the CTI Sample Console page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
REN Server Cluster ID	Select the REN server cluster on which to test the sample console.
CTI Sample Console	Click to initiate the sample console. **Note:** To demonstrate the CTI sample console, you must have a CTI server running and communicating with the specified REN server cluster.

After clicking **CTI Sample Console,** a new browser window appears that displays the sample console along with a tracer window. The Sample CTI Console contains the following group boxes which are explained in detail in the following sections:

- Open/Close Session
- Register User to Session
- Register Group, Login User to Group, and User-Group States
- Register Extension to Session, and Extension Operations
- Buddies
- Error Message/Information
- Server State and Broadcasts

Note: To enable the new browser window to appear, disable any pop-up blocking software for your browser.

Open/Close Session

Access the **Open/Close Session** group box.

This example illustrates the fields and controls on the Open/Close Session group box. You can find definitions for the fields and controls later on this page.

The Open/Close Session group box displays information about the session.

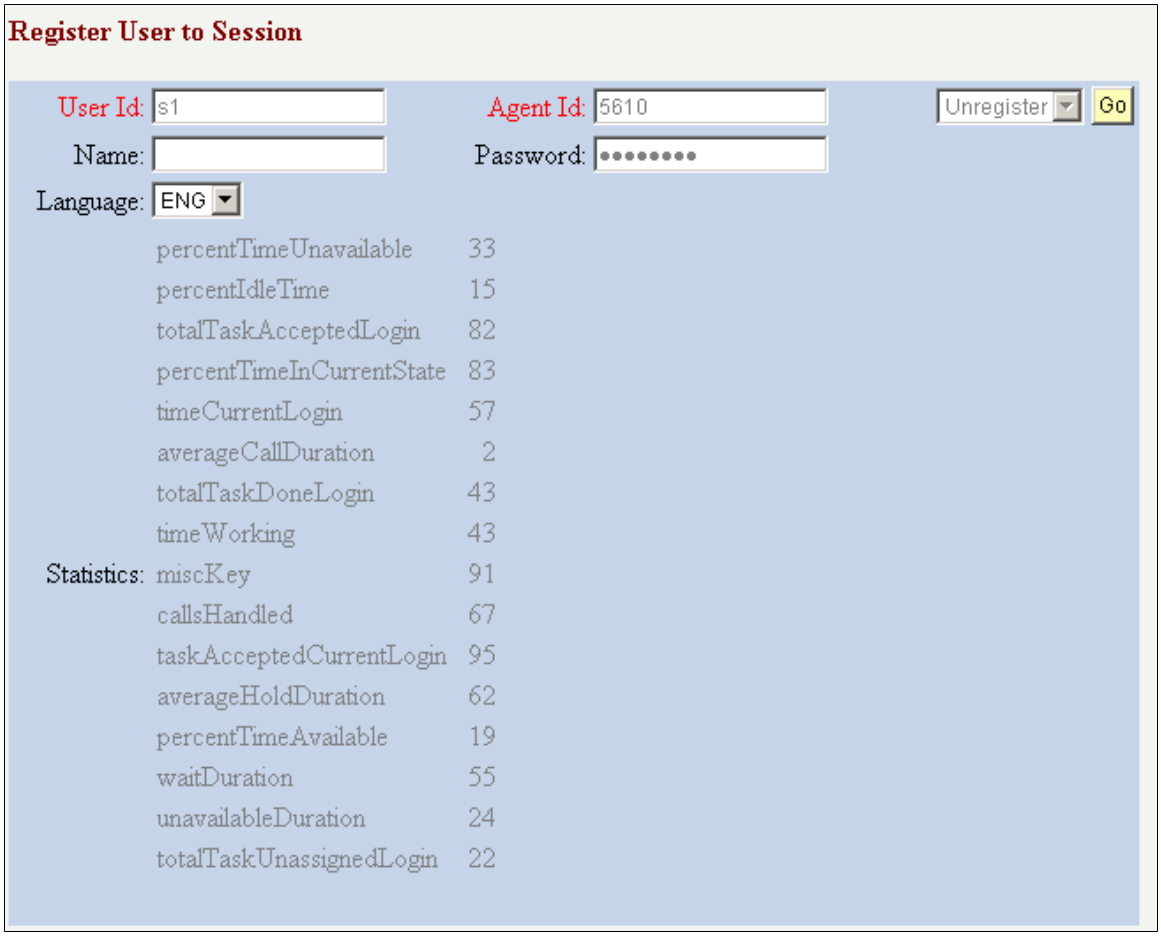
Field or Control	*Description*
CTI Session Id	This field displays a value identifying the session. If no session ID is displayed, check to ensure that the CTI server and REN server are both running.

You can close an active session by clicking the **Go** button. You cannot make a selection from the drop-down list box.

Register User to Session

Access the **Register User to Session** group box.

This example illustrates the fields and controls on the Register User to Session group box. You can find definitions for the fields and controls later on this page.



Register a user to the session in the **Register User to Session** group box.

Because a CTI agent can have an agent ID different from the agent's user ID, both values are required. If the agent has a password, the password is also required.

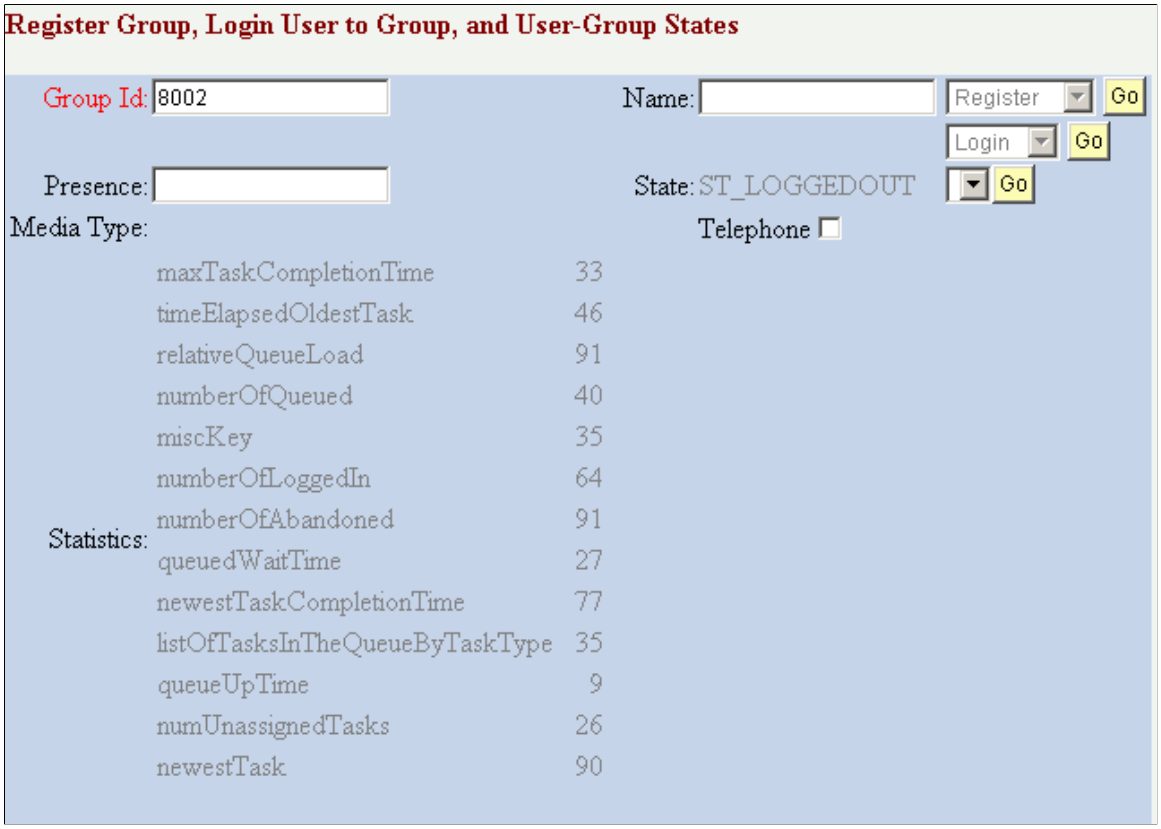
Field or Control	***Description***
Language	Reserved for future use. The default language is English.

You can register or unregister an agent by clicking the **Go** button. You cannot make a selection from the drop-down list box.

Register Group, Login User to Group, and User-Group States

Access the **Register Group, Login User to Group, and User-Group States** group box.

This example illustrates the fields and controls on the Register Group, Login User to Group, and User Group States group box. You can find definitions for the fields and controls later on this page.



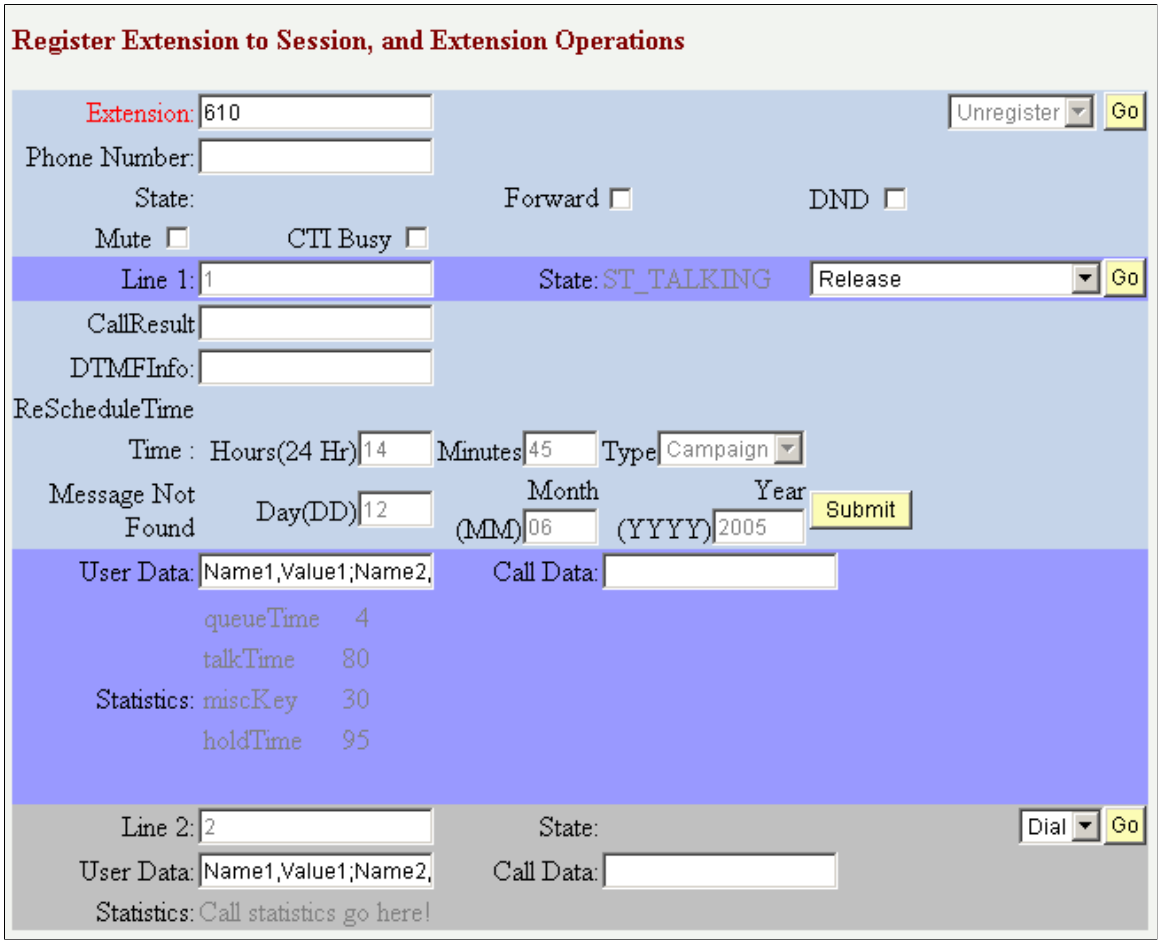
Field or Control	*Description*
Group Id	Enter the group ID to which the agent will be registered. To register the agent with the specified group, click the **Go** button. You cannot make a selection from the drop-down list box.
Login	Click the **Go** button to log in if *Login* appears in the drop-down list box. Click the **Go** button to log out if *Logout* appears in the drop-down list box. **Note:** The agent cannot receive calls until the agent is in the Ready state. **Note:** After you login to the group, a new pop-up window opens prompting for reason code data. If you register to an invalid group, the system displays an error message. See Understanding JSMCAPI.

Field or Control	*Description*
Presence	Enter a presence description to associate with the state that is selected in the drop-down list box. This presence value is different from any presence values that are predefined in the agent configuration. The agent cannot select from the pre-defined values.
State	Select an agent state. State options are: • *Ready* • *Not Ready* • *Work Ready* • *Work Not Ready* Each state value has an associated state code that appears in the **State** field. To enable incoming calls, select *Ready* from the drop-down list box and click the **Go** button.
Media Type	Reserved for future use. The sample console supports only the telephone (voice) channel. However, JSMCAPI can support other channels.

Register Extension to Session and Extension Operations

Access the **Register Extension to Session, and Extension Operations** group box.

This example illustrates the fields and controls on the Register Extension to Session and Extension Operations group box. You can find definitions for the fields and controls later on this page.



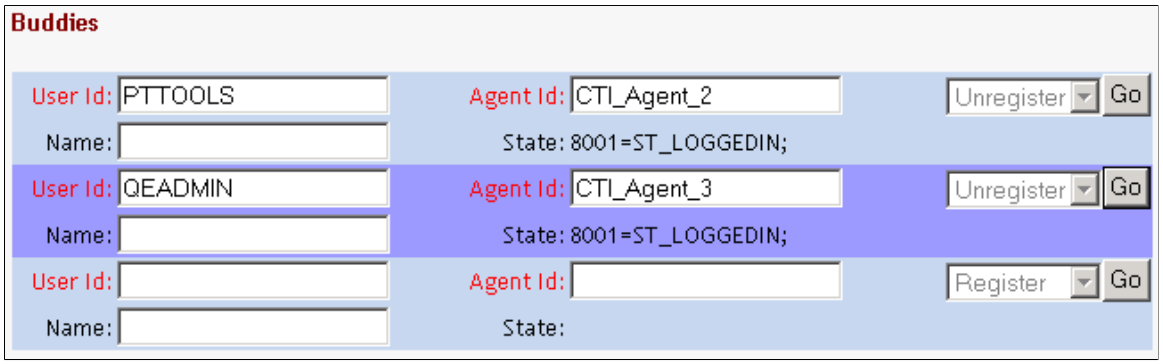
Field or Control	***Description***
Extension	Enter a valid telephone extension number.
Phone Number	Enter a valid telephone number for use in subsequent operations, such as dial, forward, or do-not-disturb (DND).
Mute	Select to mute a call. **Note:** This is enabled only when the agent is on a call.

Field or Control	*Description*
CTI Busy	This flag is checked by the queue server for an agent handling a voice task. **Note:** If the flag is checked, the queue server will not assign new tasks to such an agent. The CTI Busy flag allows the application to implement a request which conveys the status of the agent to the queue server. Because the queue server stores the task list and the status in its temporary cache, a recovered or a rebooted queue server will not have the information of a agent's previous CTI status. The agent can send its CTI status to the queue server to indicate that it is busy with a CTI task.
Forward	Select to forward incoming calls to the specified number.
DND (do not disturb)	Select to put the selected extension in do-not-disturb status.
Line 1 or **Line 2**	Displays a name or value for each line. On the sample console, the value will either be *1* or *2*.
DTMFInfo (Dual-tone Multi frequency information)	Select to send DTMF digits on the line
CallResult	Select to set call result on the line
ReScheduleTime	Select to set reschedule time on the line
Type	Select to set reschedule type on the line.
User Data	Enter name-value pairs representing user data to be attached to a call.
Call Data	Enter any call data to be attached to the call. Call data includes: • *ANI* • *DNIS* • *THISDN* • *OTHERDN*

Buddies

Access the **Buddies** group box.

This example illustrates the fields and controls on the Buddies group box.



Before an agent can chat with a buddy, the buddy must be registered in the **Buddies** group box.

Because a CTI agent can have an agent ID that is different from the agent's user ID, both values are required.

Note: If one agent is logged on the multichannel console and another agent is logged on the sample pages, the agents may see an incorrect buddy state for each other.

Error Messages/Information

Access the Error Messages/Information group box.

This example illustrates the fields and controls on the Error Messages/Information group box. You can find definitions for the fields and controls later on this page.

Any error messages associated with the process display in this section.

Field or Control	*Description*
Clear	Click the button to clear a displayed error message.

Server State and Broadcasts

Access the **Server State and Broadcasts** group box.

This example illustrates the fields and controls on the Server State and Broadcasts group box. You can find definitions for the fields and controls later on this page.

Field or Control	*Description*
CTI State	Displays the status of the CTI server.
REN State	Displays the status of the REN server.
Info (information)	Displays the text of a broadcast message from the CTI server.

Tracer

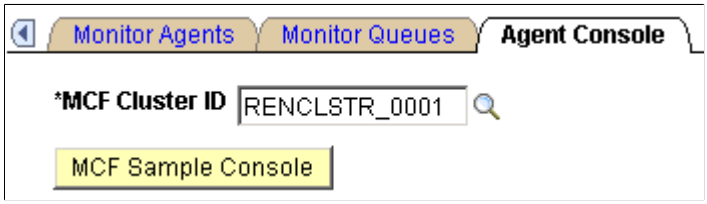
The tracer displays all requests between the CTI server and the console, which can be useful for debugging.

Using the Agent Console Page

Access the Agent Console page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **MCF Sample Pages** > **Agent Console tab**

This example illustrates the fields and controls on the Agent Console page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
MCF Sample Console	Click to initiate the sample console. **Note:** To demonstrate the MCF sample console, you must have an MCF cluster running and communicating with the specified REN server cluster.

After you click **MCF Sample Console,** a new browser window displaying the sample console appears along with a Tracer Window. The MCF Sample Page contains the following group boxes which are explained in detail in the following sections:

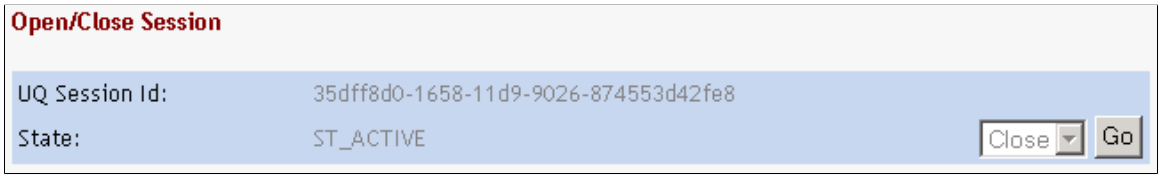
- Open/Close Session
- Register User to Session
- Register Group, Login User to Group, and User-Group States
- Address Operations
- Buddies
- Error Message/Information
- Server State and Broadcasts

Note: To enable the new browser window to appear, disable any pop-up blocking software for your browser.

Open/Close Session

Access the **Open/Close Session** group box.

This example illustrates the fields and controls on the Open/Close Session group box. You can find definitions for the fields and controls later on this page.

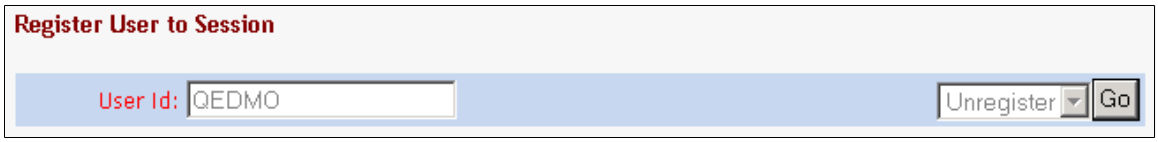


Field or Control	*Description*
UQ Session Id	The universal queue session ID.

Register User to Session

Access the **Register User to Session** group box.

This example illustrates the fields and controls on the Register User to Session group box.

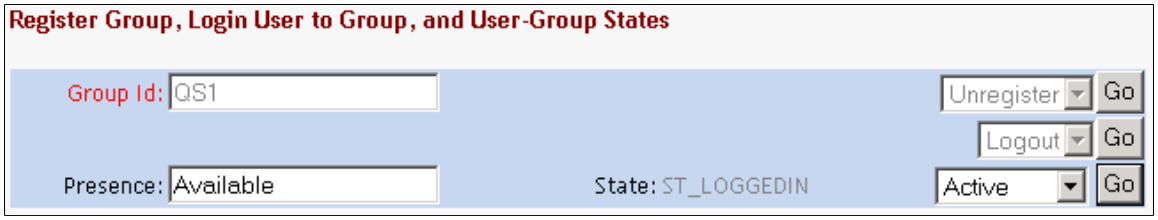


Register Group, Login User to Group, and User-Group States

Access the **Register Group, Login User to Group, and User-Group States** group box.

Note: The presence change happens only when you change from active to inactive or from inactive to active. When you log in, the default value is active and therefore no presence change occurs.

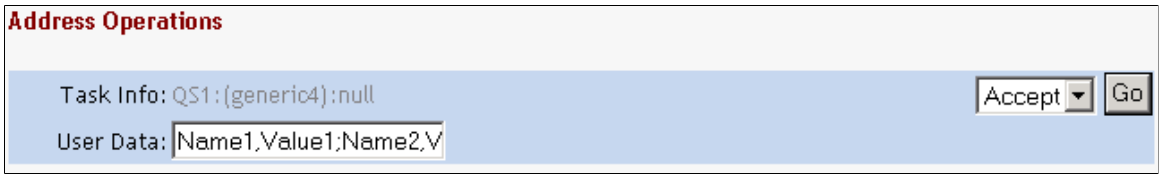
This example illustrates the fields and controls on the Register Group, Login User to Group, and User-Group States group box.



Address Operations

Access the **Address Operations** group box.

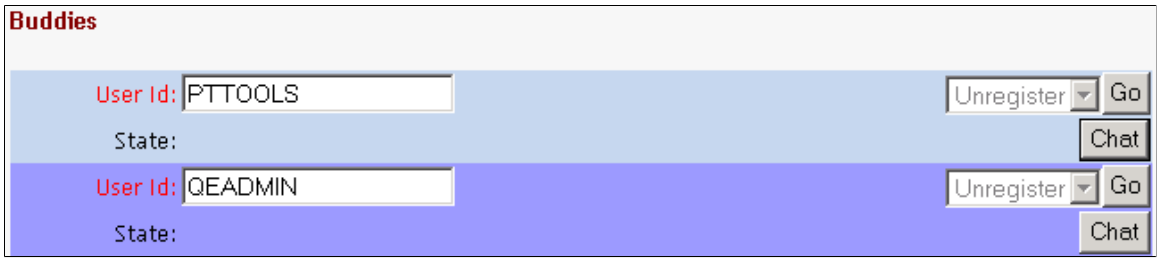
This example illustrates the fields and controls on the Address Operations group box.



Buddies

Access the **Buddies** group box.

This example illustrates the fields and controls on the Buddies group box.



Note: If one agent is logged on the multichannel console and another agent is logged on the sample pages, the agents may see an incorrect buddy state for each other.

Note: The buddy states on this page are not automatically displayed. You must monitor an agent first using the Monitor Agents page before the buddy states are displayed.

See Using the Monitor Agents Page and Sample Monitor - Agent States Page.

Field or Control	*Description*
Chat	Click to initiate an agent-to-agent chat session.

Error Messages/Information

Access the Error Messages/Information group box.

This example illustrates the fields and controls on the Error Messages/Information group box.

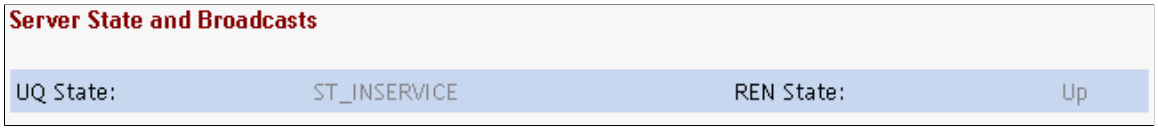
Any error messages associated with the process display here.

Field or Control	*Description*
Clear	Click the button to clear a displayed error message.

Server State and Broadcasts

Access the **Server State and Broadcasts** group box.

This example illustrates the fields and controls on the Server State and Broadcasts group box. You can find definitions for the fields and controls later on this page.



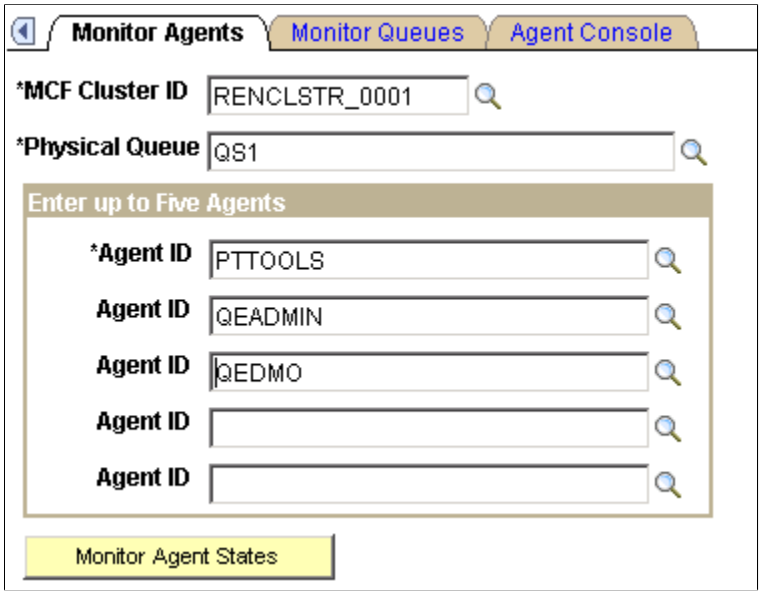
Field or Control	*Description*
UQ State	Displays the state of the queue server.
REN State	Displays the state of the REN server.

Using the Monitor Agents Page and Sample Monitor - Agent States Page

Access the Monitor Agents page using the following navigation path:

PeopleTools > MultiChannel Framework > Universal Queue > MCF Sample Pages > Monitor Agents tab

This example illustrates the fields and controls on the Monitor Agents page.

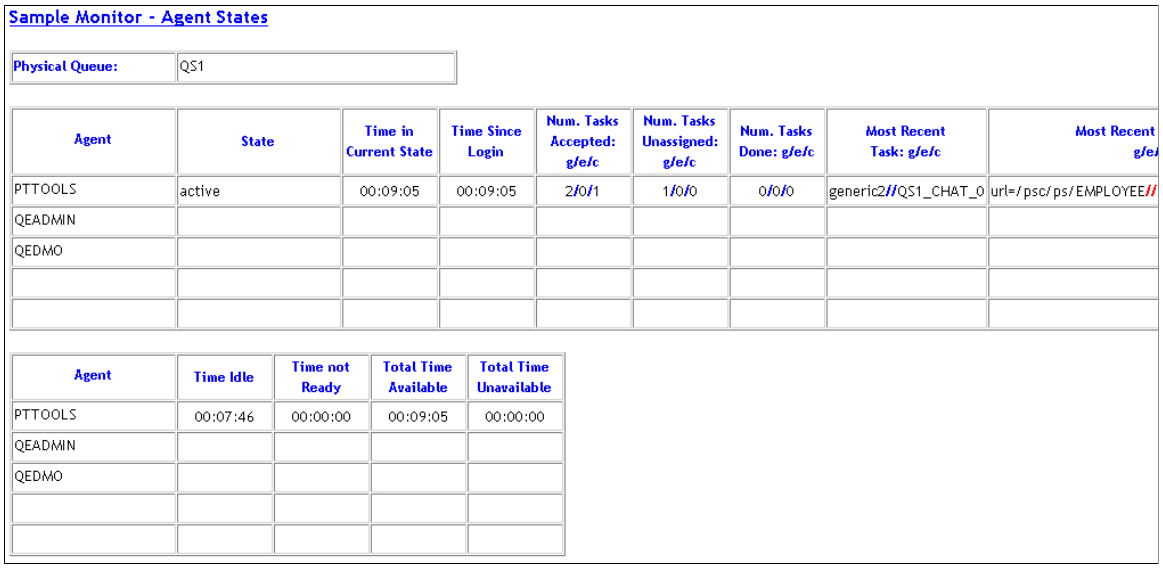


Select the MCF cluster, physical queue on that cluster, and up to five agents on that queue to be monitored.

After you click **Monitor Agent States,** a new browser window appears.

Note: To enable the new browser window to appear, disable any pop-up blocking software for your browser.

This example illustrates the fields and controls on the Sample Monitor - Agent States page. You can find definitions for the fields and controls later on this page.



Sample Monitor - Agent States

Physical Queue:	QS1

Agent	State	Time in Current State	Time Since Login	Num. Tasks Accepted: g/e/c	Num. Tasks Unassigned: g/e/c	Num. Tasks Done: g/e/c	Most Recent Task: g/e/c	Most Recent g/e/
PTTOOLS	active	00:09:05	00:09:05	2/0/1	1/0/0	0/0/0	generic2//QS1_CHAT_0	url=/psc/ps/EMPLOYEE//
QEADMIN								
QEDMO								

Agent	Time Idle	Time not Ready	Total Time Available	Total Time Unavailable
PTTOOLS	00:07:46	00:00:00	00:09:05	00:00:00
QEADMIN				
QEDMO				

Physical Queue

The ID of the physical queue that is being monitored appears in the first table.

Agent Statistics Table 1

The first agent statistics table displays information that is published by the JSMCAPI user.onStat1 event.

Field or Control	*Description*
Time in Current State	Displays the duration of the agent's current state. The value returned is `timeInCurrentState`.
Time Since Login	Denotes the duration since the agent logged in. The value returned is `timeLogin`.
Num. Tasks Accepted: g/e/c (number of tasks accepted)	Displays the number of tasks in each task type that the agent has accepted since login. The value returned is `numTaskAccepted`.
Num. Tasks Unassigned: g/e/c (number of tasks unassigned)	Displays the number of tasks in each task type that have been unassigned from the agent. Returned by `numTasksUnassigned`.
Num. Tasks Done: g/e/c (number of tasks done)	Displays the number of tasks in each task type that the agent has completed. Returned by `numTasksDone`.
Most Recent Task: g/e/c	Identifies the most recent task that was accepted since login for each task type. Returned by `mostRecentTaskId`.
Most Recent Task Data: g/e/c	Displays data on the most recent task that was accepted since login for each task type. Returned by `mostRecentTaskInfo`

Agent Statistics Table 2

The second agent statistics table displays information that is published by the JSMCAPI user.onStat2 event.

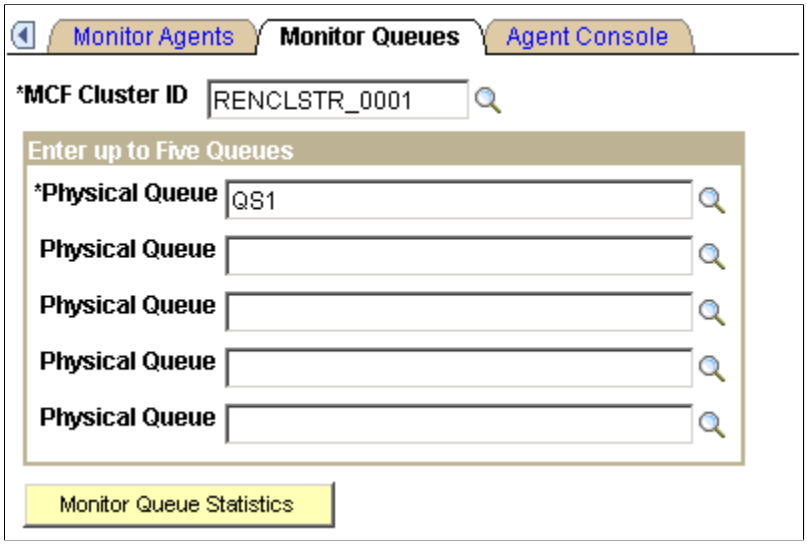
Field or Control	*Description*
Time Idle	Displays the duration of time during which the agent has been idle. An agent is considered to be idle from the time the agent logs on until the agent accepts a task, and during the time between completing the last accepted task and accepting another task. The value returned is `timeIdle`.
Time not Ready	Displays the duration of time during which the queue could not assign tasks to the agent because the agent's maximum workload has been reached. The value returned is `timeNotReady`.
Total Time Available	Displays the total amount of time that the agent has been in an available status. The value returned is `totalTimeAvailable`.
Total Time Unavailable	Displays the total amount of time that the agent has been in an unavailable status. The value returned is `totalTimeUnavailable`.

Using the Monitor Queues Page and Sample Monitor - Queue Statistics Page

Access the Monitor Queues page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Universal Queue** > **MCF Sample Pages** > **Monitor Queues tab**

This example illustrates the fields and controls on the Monitor Queues page.

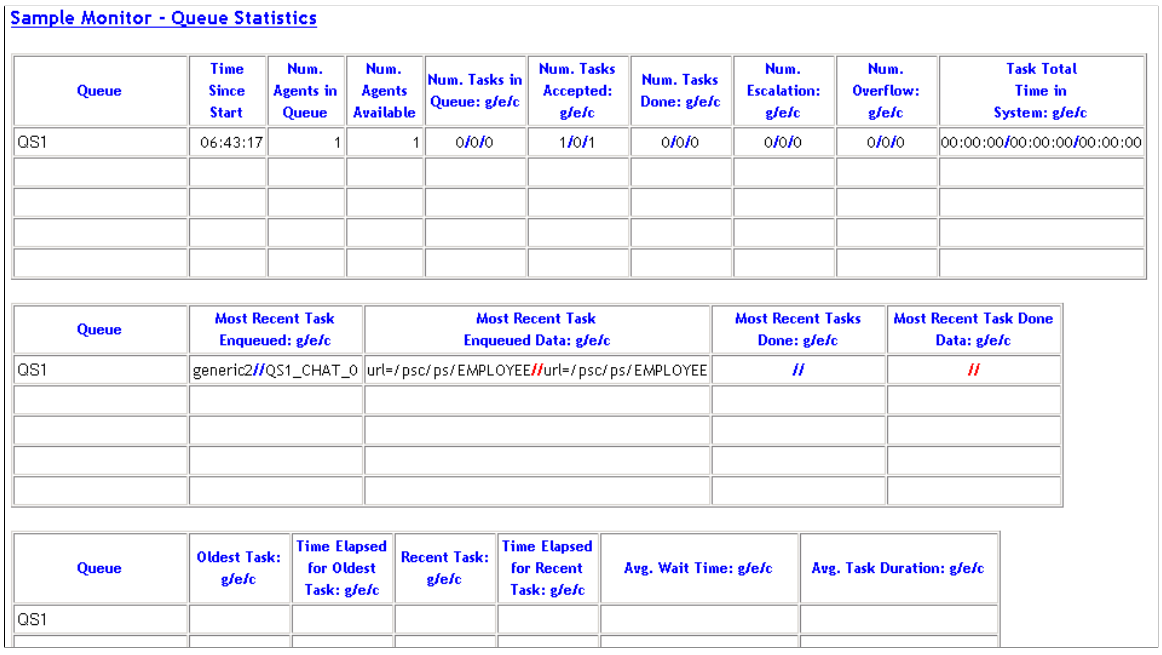


Select the MCF cluster and up to five physical queues on that cluster to be monitored.

After you click **Monitor Queue Statistics,** a new browser window appears.

Note: To enable the new browser window to appear, disable any pop-up blocking software for your browser.

This example illustrates the fields and controls on the Sample Monitor - Queue Statistics page. You can find definitions for the fields and controls later on this page.



Queue Statistics 1

The first queue statistics table displays data that does not change frequently. The table displays information that is published by the JSMCAPI group.onStat1 event.

Field or Control	*Description*
Queue	Displays an identifier for this queue. The value returned is `Queue_Id`.
Time Since Start	The duration since this queue was started (application server domain start). The value returned is `timeSinceStart`.
Num. Agents in Queue (number of agents in queue)	The number of agents that are currently logged in to this queue. The value returned is `numAgentsLoggedIn`.
Num. Agents Available (number of agents available)	The number of agents that are currently logged in to this queue with an available status. The value returned is `numAgentsAvail`.
Num. Tasks in Queue: g/e/c (number of tasks in queue)	The number of tasks that are currently on this queue, by task type. The value returned is `numTasksQueued`.
Num. Tasks Accepted: g/e/c (number of tasks accepted)	The number of accepted tasks that are currently on this queue, by task type. The value returned is `numTaskAccepted`.
Num. Tasks Done: g/e/c (number of tasks done)	The number of tasks on this queue that have been completed, by task type. The value returned is `numTaskDone`.
Num. Escalation: g/e/c (number of escalation)	The number of escalated tasks on this queue, by task type. The value returned is `numEscalation`.
Num. Overflow: g/e/c (number of overflow)	The number of overflow tasks on this queue, by task type. The value returned is `numOverflow`.

Field or Control	*Description*
Task Total Time in System: g/e/c	Displays the total duration of the most recently completed task in the system. The total time is the difference between the time that the task was enqueued and the time at which it was marked done. The value returned is `taskTotalTimeInSystem`.

Queue Statistics 2

The second queue statistics table displays data that changes frequently. The table displays information that is published by the JSMCAPI group.onStat1 event.

Field or Control	*Description*
Most Recent Task Enqueued: g/e/c	Identifies the task that was most recently enqueued on this queue, by task type. The value returned is `mostRecentTaskEnqueued`.
Most Recent Task Enqueued Data: g/e/c	Displays data for the most recently enqueued task on this queue, by task type. The value returned is `mostRecentTasksEnqueuedData`.
Most Recent Tasks Done: g/e/c	Identifies the task that was most recently completed, by task type. The value returned is `mostRecentTaskDone`.
Most Recent Task Done Data: g/e/c	Displays data for the most recently completed tasks, by task type. The value returned is `mostRecentTaskDoneData`.

Queue Statistics 3

The table displays information that is published by the JSMCAPI group.onStat2 event.

Field or Control	*Description*
Oldest Task: g/e/c	Identifies the oldest enqueued and accepted task on this queue, by task type. The value returned is `oldestTask`.

Field or Control	*Description*
Time Elapsed for Oldest Task: g/e/c	Displays the duration that the oldest task on this queue has been enqueued, by task type. The value returned is `timeElapsedOldestTask`.
Recent Task: g/e/c	Identifies the most recent task that was enqueued and accepted on this queue, by task type. The value returned is `recentTask`.
Time Elapsed for Recent Task: g/e/c	Displays the duration that the most recent task on this queue has been enqueued, by task type. The value returned is `timeElapsedRecentTask`.
Avg. Wait Time: g/e/c (average wait time)	Displays the average time between a task being enqueued and being accepted, by task type. The value returned is `averageWaitTime`.
Avg. Task Duration: g/e/c (average task duration)	Displays the average duration before a task is completed. The value returned is `averageTaskDuration`. The average task duration is calculated by referencing a list of completed tasks. The size of that list is determined by the donelistsize parameter that is set on the Cluster Tuning page. See Tuning Cluster Parameters.

Using PeopleCode Built-in Functions with PeopleSoft MultiChannel Framework

PeopleSoft MultiChannel Framework uses several PeopleCode built-in functions. Application developers use these functions to communicate task requests and parameters to the queue server. For example, use InitChat code behind a button on an application page to initiate a chat session with an agent.

The built-in functions are:

- DeQueue

  Use DeQueue to notify the queue server that an enqueued task has been completed and to remove the task from the queue.

- EnQueue

Use EnQueue to add a task to an active physical queue belonging to the specified logical queue.

- Forward

  Use Forward to transfer a task from one agent to another agent or another logical queue.

- InitChat

  Use InitChat to place a customer chat request on a queue.

- NotifyQ

  Use NotifyQ to notify the queue server of a task enqueued by EnQueue.

See PeopleCode Language Reference.

Using Universal Queue Classes

The universal queue classes provide the means by which applications can inspect objects that are processed by the queue server, such as tasks, agents, and queues. The classes also enable tasks to reenter the queue with their original task ID, as well as keep task times based on their original entry into the system.

See “Understanding Universal Queue Classes” (PeopleCode API Reference).

Chapter 11

Configuring PeopleSoft MCF for Third-Party Routing Systems

Understanding Third-Party Routing Systems

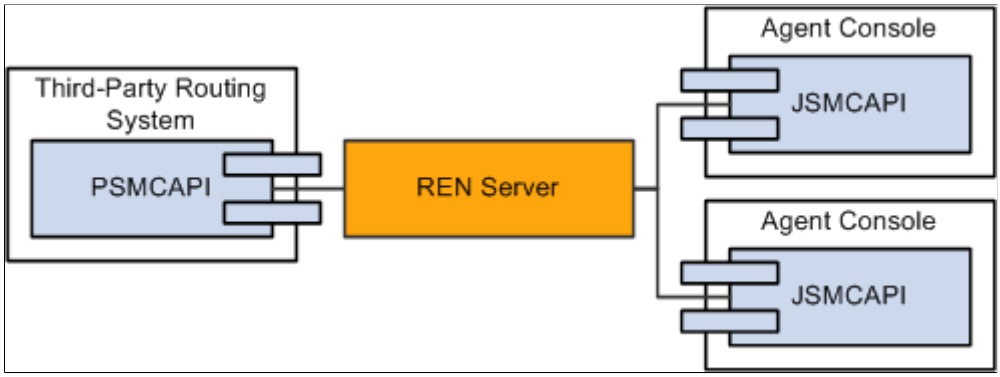
In previous releases, PeopleSoft CTI required a third-party middleware in the form of the Genesys CTI Framework, a Cisco ICM system, or CTI middleware that implements PSMCAPI.

Beginning with PeopleTools 8.48, PeopleSoft MultiChannel Framework (MCF) uses third-party multichannel routing systems to offer a wide range of communication channels to empower PeopleSoft applications like CRM. In the third-party system, the queue is the logical storage unit for representing a work item, and no difference exists between a logical and physical queue. The third party develops the universal routing system to route email, voice, chat, and generic tasks.

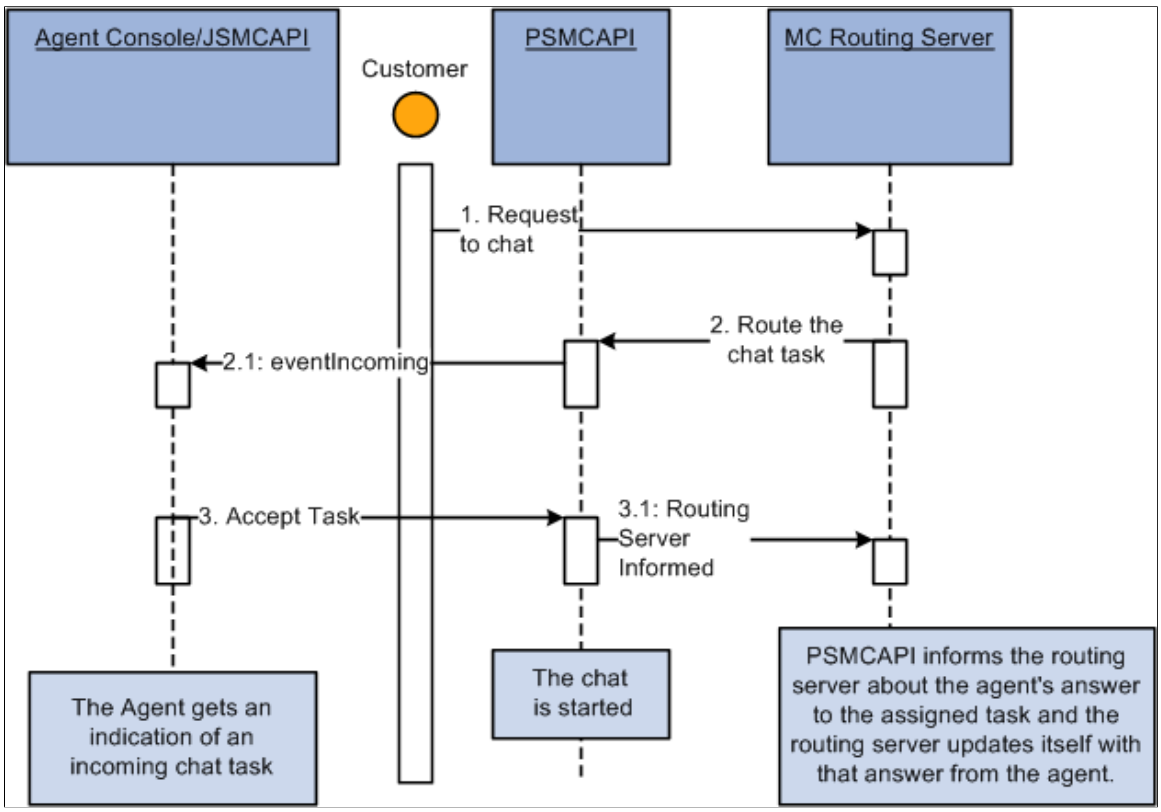
PeopleSoft customers may choose either the pre-8.48 features of queue server and CTI or the new third-party routing system to use email, chat, voice, and generic channels.

PSMCAPI provides an interface to the third-party routing system that enables communication between various PeopleSoft components and the third party. An event, like an incoming task from the third-party routing system, is received by PSMCAPI and pushed to JSMCAPI (MCF console) using the REN server. Similarly, a request from MCF console or JSMCAPI is sent to PSMCAPI using the REN server and is eventually passed on to the third-party routing system. The REN server routes requests and events internally to the PeopleSoft system, including an agent desktop or browser.

The following diagram illustrates how PSMCAPI acts as an interface to the third-party routing system and communicates with JSMCAPI through the REN server.

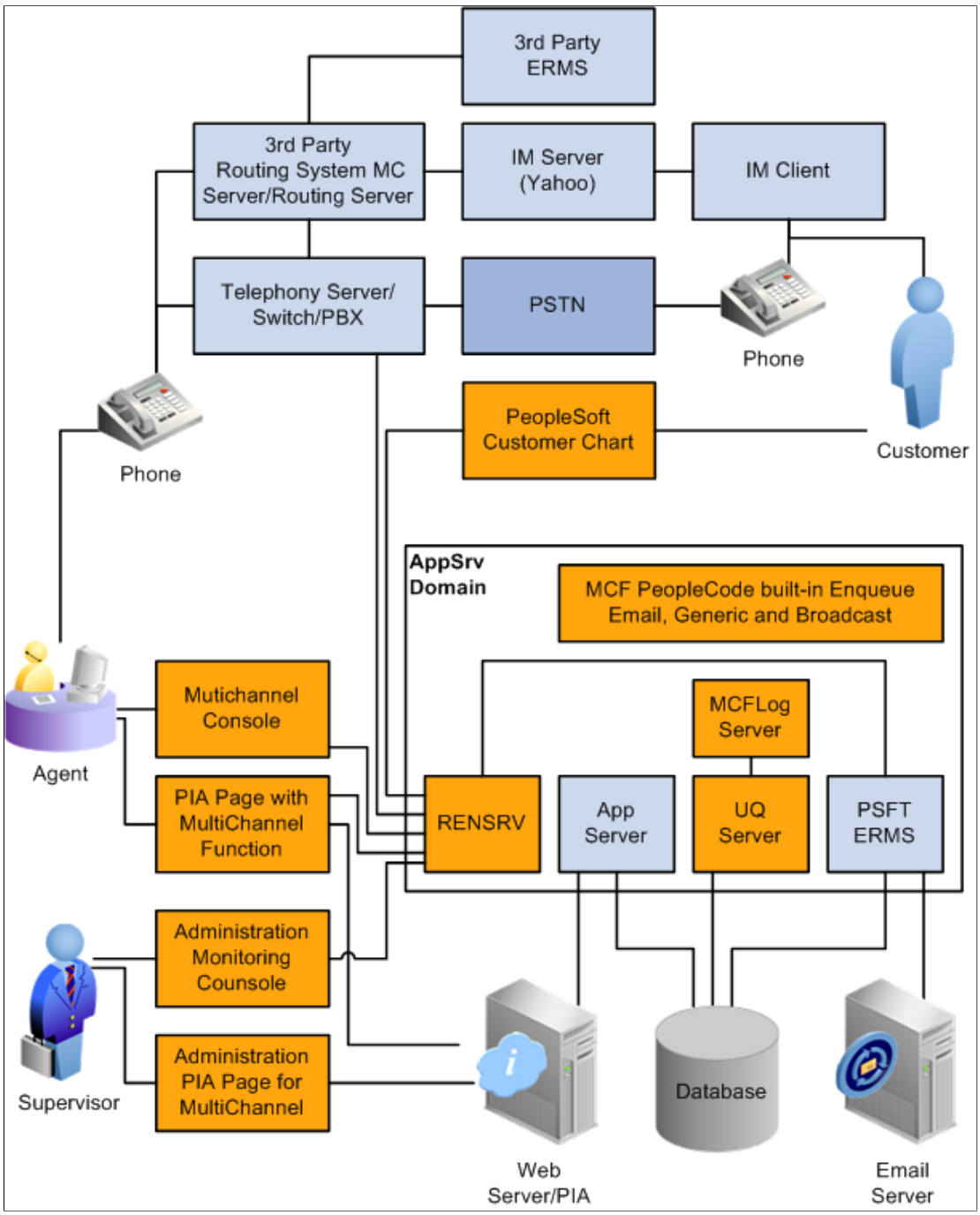


The following diagram illustrates the interaction of PSMCAPI with third party and JSMCAPI.



PSMCAPI and JSMCAPI support multiple channels of communication, queue, and agent statistics information that enable the leverage of third-party routing systems.

The following diagram illustrates how the third-party routing system interacts with PeopleSoft MCF.



The third-party routing system connects to all kinds of media servers, and routes the task to the agent based on routing rules. The third-party routing system connects to the REN server through HTTP or HTTPS using PSMCAPI. All requests sent by the agents to third-party system such as login, logout, accept incoming call, and forward email use the above connection. All media requests sent by PeopleCode built-in functions to the REN server such as InitChat or Enqueue also use the above connection. All events from the routing system such as assignment, broadcast, and statistics, go to the agent or the supervisor through this connection.

In the third-party routing system, the queue server acts as an overflow or escalation adapter that writes the overflow or escalated tasks from the third-party routing system to the PeopleSoft Database. When using a

third-party routing system for escalated and overflowed task, the queue server is used for task insertions to the database. The third-party routing system processes and maintains task properties and timers. On a task time-out, the routing system sends an event to the queue server to insert the task in the database. The routing system creates the events to remove the task from the agents and recalculates their workload.

Defining Third-Party Routing System Requirements

In general, the third-party router:

- Allows business users to define routing strategies.
- Defines escalation and timeout routing paths.
- Allows agents to receive any type of work from multiple channels and queues simultaneously or in parallel.
- Allows pushed and pulled events simultaneously or in parallel.
- Defines intra- and extra-channel escalations.

Defining Third-Party Routing Rules

Third-party routing rules are based on:

- Task type, for example, email, chat, voice, scheduled callback, fax, and so on.
- Time against SLA (Service Level Agreement) in form of overflow and escalation timers.
- Priority of the event expressed as a positive single-digit number.
- Skill within organization or task type.
- Primary language required to resolve any issue.
- Cost of each event type assigned by the administrator to avoid overloading.

The agents or supervisors must have the capabilities to:

- Change the skills, language, priority, or cost of an item to affect its subsequent routing.
- Assign an item to a specific queue or agent.
- Accept, reject, forward, mark as resolved, or mark as awaiting additional information.

Configuring PeopleSoft MCF for a Third Party

This section lists the basic steps to configure PeopleSoft MCF for a third party. Subsequent sections of this PeopleBook describe each step in detail.

You complete these steps to configure PeopleSoft MCF for a third party:

1. Define the third-party flag.

See Defining the Third-Party Flag.

2. Define PeopleSoft MCF clusters for a third party.

   See Defining the PeopleSoft MCF Cluster Page for a Third Party.

3. Tune PeopleSoft MCF clusters for a third party.

   See Tuning PeopleSoft MCF Cluster Parameters for a Third Party.

4. Notify PeopleSoft MCF of changed parameters for a third party.

   See Notifying PeopleSoft MCF Clusters of Changed Parameters for a Third Party.

5. Define PeopleSoft MCF queues for a third party.

   See Defining PeopleSoft MCF Queues for a Third Party.

6. Define PeopleSoft MCF agents for a third party.

   See Defining PeopleSoft MCF Agents for a Third Party.

7. Configure PeopleSoft MCF tasks for a third party.

   See Configuring PeopleSoft MCF Tasks for a Third Party.

8. Configure CTI.

   See Configuring CTI for a Third Party.

9. View event logs.

   See Viewing Event Logs for a Third Party.

10. View broadcast logs.

    See Viewing Broadcast Logs for a Third Party.

11. Work with third-party sample pages.

    See Working with Third-Party Sample Pages.

Defining the Third-Party Flag

This section discusses how to define the third-party flag.

To define the third-party flag, use the Third-party Flag (MCF_TP_FLAG_CMP) component.

Defining the Third-Party Flag

Access the Third Party Flag page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Third-Party Configuration** > **Enable MCF Third-Party Routing**

This example illustrates the fields and controls on the Third-Party Flag page.

The default value of the third-party flag is *False.* If the flag is set to *False,* the queue server routes chat, generic, and email and uses PeopleSoft CTI to route voice. If the flag is set to *True*, all events, chat, generic, email, and voice are routed by the third-party routing system, and you can configure MCF using third-party pages only. The queue server:

- Stops listening to all routing requests.
- Starts up new listeners for overflow and escalation.

Separate database tables are created for queue server and third-party routing server that combine all fields from MCF and CTI. When the third-party flag is set to *True*, all pages using queue server are disabled and grayed out. When the third-party flag is set to *False*, all pages using third-party routing system are disabled and grayed out.

Note: Whenever you change the value of this flag, you must reboot the queue server and MCF Log server.

Defining the PeopleSoft MCF Cluster Page for a Third Party

This section discusses how to define the PeopleSoft MCF Cluster page for the third party. To view the Cluster page, use the Cluster (MCF_TP_RENCFG_CMP) component.

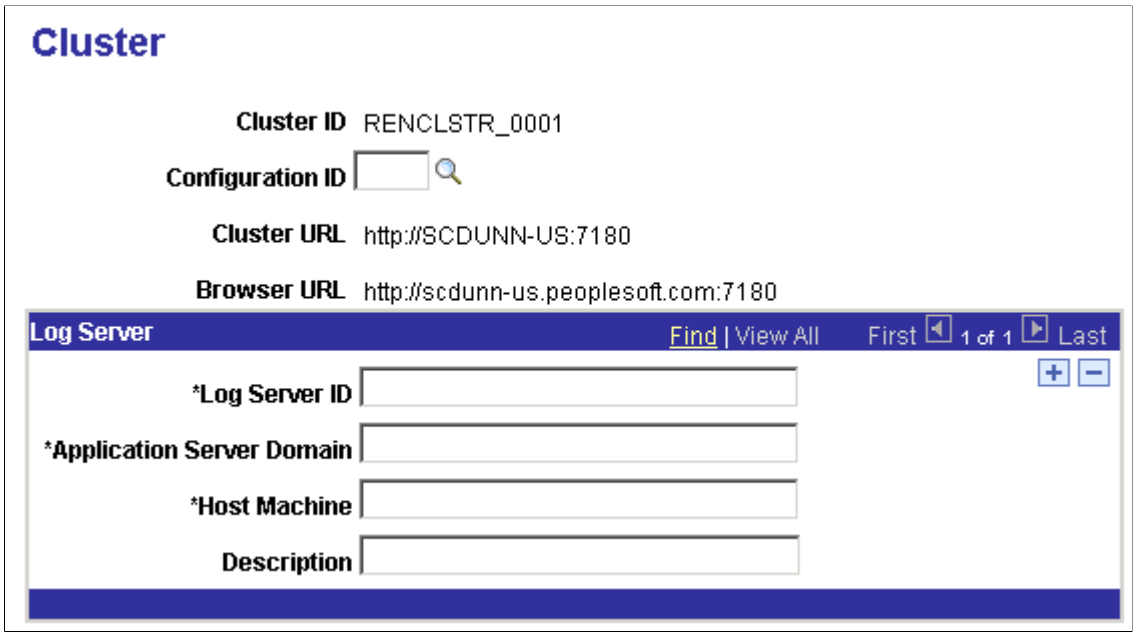
Defining the PeopleSoft MCF Cluster Page for a Third Party

The Cluster page displays the associated PeopleSoft MCF cluster URLs and log server details for the selected cluster.

Access the Cluster page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Third-Party Configuration** > **Cluster**

This example illustrates the fields and controls on the Cluster page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Cluster ID	Displays the cluster ID of the cluster containing the queue server and log server.
Configuration ID	Displays the configuration ID. **Note:** This value is required only for CTI configuration. To select the configuration ID from lookup, define the configuration ID when you are configuring CTI. See Configuring CTI Console Type.
Cluster URL	Displays the URL for the REN server that serves this cluster. **Note:** The **Cluster URL** is defined on the REN Server Definitions page. See Defining REN Servers.
Browser URL	Displays the browser URL. **Note:** The browser URL may be different from the cluster URL, which should not have to go through any firewall, reverse proxy server, or other outward-facing security barrier. The browser URL is same as defined on the REN Server Definitions page. See Defining REN Servers.

Log Server

A cluster consists of a primary log server and any number of backup servers. Each cluster requires a minimum of one log server and one queue server. The primary log server is the first log server started and the remaining log servers are backups. If the primary log server fails, the system determines the subsequent primary log server among the backups.

Note: You can save a cluster without creating a log server or a queue server. But, for MCF to run, you need at least one log server and one queue server.

You can add a log server to a cluster by adding a new row. Before removing a log server, ensure that it is not the primary, then shut down its domain. Then click **Delete** (the minus sign).

Field or Control	*Description*
Log Server ID	Enter a unique identifier for each log server to identify its entries in the log cluster table. The queue server process paired with this log server uses this same ID to identify its entry in the database table.
Application Server Domain	Enter the application server domain of which this log server is a member.
Host Machine	Enter the name of the application server host machine.
Description	Enter the description of the host machine.

Tuning PeopleSoft MCF Cluster Parameters for a Third Party

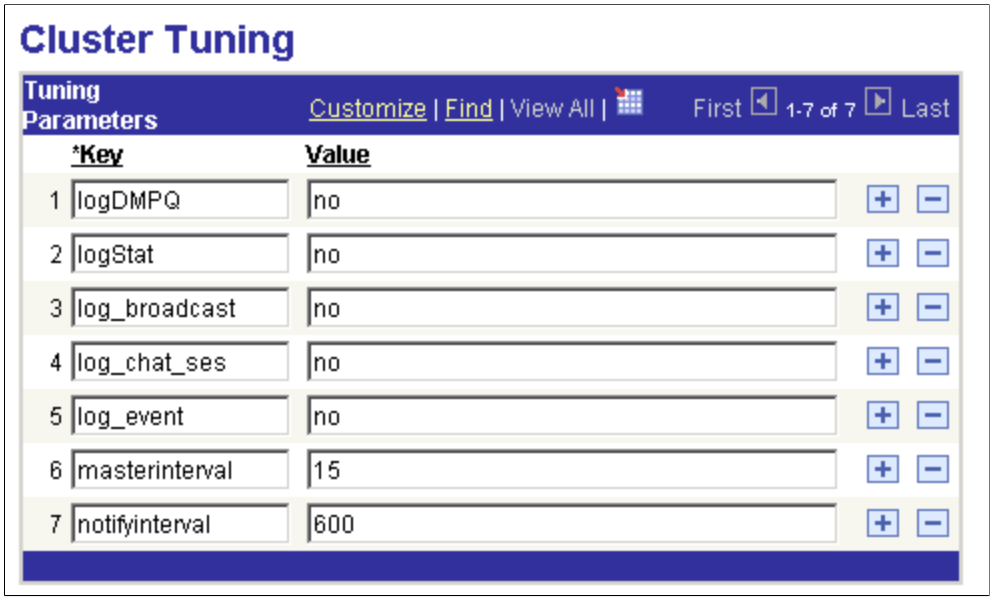
This section discusses how to tune PeopleSoft MCF cluster parameters for a third party. Tuning cluster parameters may give you better performance. To tune cluster parameters, use the Cluster Tuning (MCF_TP_SYS_NV_CMP) component.

Tuning PeopleSoft MCF Cluster Parameters for a Third Party

Access the Cluster Tuning page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Third-Party Configuration** > **Notify MCF Third-Party Cluster**

This example illustrates the cluster tuning parameters on the Cluster Tuning page.



Use the Cluster Tuning page to set MCF cluster parameters to optimize performance or enable logging for a cluster.

If you make changes to a cluster parameter, you must use the third-party Notify Cluster page to propagate the changes.

See Notifying Third-Party Clusters of Changed Parameters.

The following table lists the cluster tuning parameters you can modify and describes the default values and usage of each:

Key	***Default value***	***Usage***
logDMPQ	*No*	Select *Yes* if you want PSMCFLOG to log REN server event notifications resulting from bcastinterval broadcasts. Only the event is logged, not its contents. This is a logging parameter. If you change the value of this parameter you must use the Refresh Logging Parameters button on the third-party Cluster Notify page to notify clusters of the changed parameter See Notifying Third-Party Clusters of Changed Parameters, Viewing Event Logs.

Key	Default value	Usage
logStat	*No*	Select *Yes* to log the statistics returned by the queue server for the onStat1 user and group events to the database. This is a logging parameter. If you change the value of this parameter you must use the Refresh Logging Parameters button on the third-party Cluster Notify page to notify clusters of the changed parameter See Notifying Third-Party Clusters of Changed Parameters, Using and Demonstrating JSMCAPI.
log_broadcast	*No*	Select *Yes* to turn on logging of the broadcast messages that are sent. This is a logging parameter. If you change the value of this parameter you must use the Refresh Logging Parameters button on the third-party Cluster Notify page to notify clusters of the changed parameter See Notifying Third-Party Clusters of Changed Parameters, Viewing Broadcast Logs.
log_chat_ses	*No*	Select *Yes* to turn on logging of the contents of chat sessions. This is a logging parameter. If you change the value of this parameter you must use the Refresh Logging Parameters button on the third-party Cluster Notify page to notify clusters of the changed parameter See Notifying Third-Party Clusters of Changed Parameters, Viewing Chat Logs.

Key	*Default value*	*Usage*
masterinterval	*15*	Interval, in seconds, after which a cluster primary updates its timestamp in its cluster tables. Secondary clusters check the timestamp to determine that the primary cluster is still running. A lower value enables rapid discovery of a failed primary server, but increases log server overhead. A higher value reduces log server overhead, but delays discovery of a failed primary server. If only one log server is configured for an MCF cluster, this value can be large. The masterinterval value also acts as a heartbeat interval for the primary log server connection to user consoles. This is a timing parameter. If you change the value of this parameter you must use the Refresh Timing Parameters button on the third-party Cluster Notify page to notify clusters of the changed parameter. See Notifying Third-Party Clusters of Changed Parameters.
notifyinterval	600	Interval, in seconds, after which the database is checked for any pending enqueued tasks. The third party will be notified if any enqueued task is pending in the database. This is a timing parameter. If you change the value of this parameter you must use the Refresh Timing Parameters button on the third-party Cluster Notify page to notify clusters of the changed parameter. See Notifying Third-Party Clusters of Changed Parameters.

Notifying PeopleSoft MCF Clusters of Changed Parameters for a Third Party

This section describes how to notify PeopleSoft MCF clusters of changed parameters for a third party. To notify clusters of changed parameters, use the Cluster Notify (MCF_TP_NOTIFY_CMP) component.

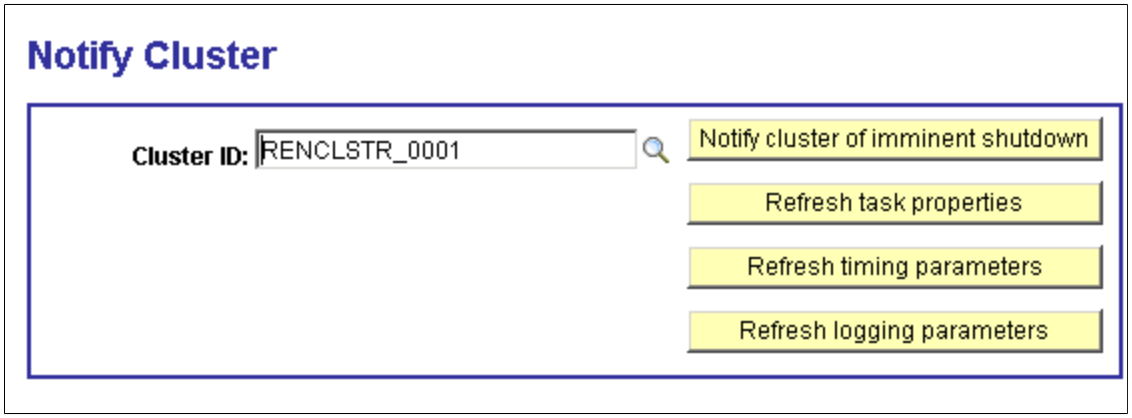
Notifying Third-Party Clusters of Changed Parameters

Use the Notify Cluster page to notify a cluster of certain changes to its parameters, constituent queues, or that its application servers are being shut down.

Access the Notify Cluster page using the following navigation path:

PeopleTools > MultiChannel Framework > Third-Party Configuration > Notify MCF Third-Party Cluster

This example illustrates the fields and controls on the Notify Cluster page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Notify cluster of imminent shutdown	Click to send a message to every agent logged on to the selected cluster that they have been logged off. Send this notification if the cluster's application servers are being shut down.
Refresh task properties	Click to load task properties that have been changed on the Tasks page for the selected cluster.
Refresh timing parameters	Click to reload parameters, other than logging parameters, changed on the Cluster Tuning page for the selected cluster.
Refresh logging parameters	Click to reload logging parameters changed on the Cluster Tuning page for the selected cluster.

Defining PeopleSoft MCF Queues for a Third Party

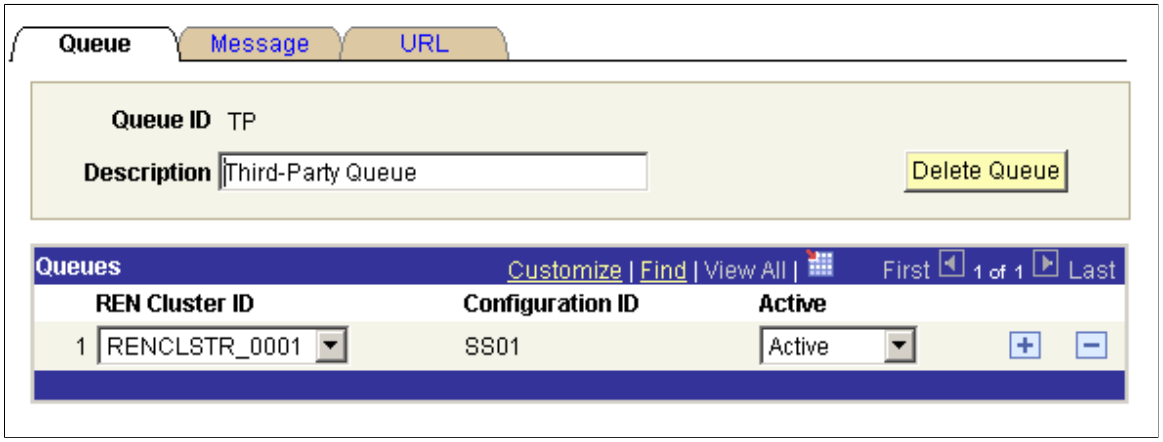
To define PeopleSoft MCF queues for the third-party routing system, use the third-party queue (MCF_TP_Q_CFG_CMP) component. This section discusses how to define peopleSoft MCF Queues for a third party.

Defining PeopleSoft MCF Queues for a Third Party

Access the Queue page using the following navigation path:

PeopleTools > MultiChannel Framework > Third-Party Configuration > MCF Third-Party Queues

This example illustrates the fields and controls on the Queue page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Queue ID	Queue IDs must be alphanumeric, but they may include underscore characters.
Configuration ID	Displays the configuration ID. **Note:** This value is required only for CTI configuration. To select the configuration ID from lookup, define the configuration ID when you are configuring CTI. See Configuring CTI Console Type.
Delete Queue	Click to remove this queue. Deleting a queue means that no work or agents can be assigned to the queue and the queue is removed from all agents' available queues.
Cluster ID	The **Cluster ID** field identifies the cluster that services this queue.

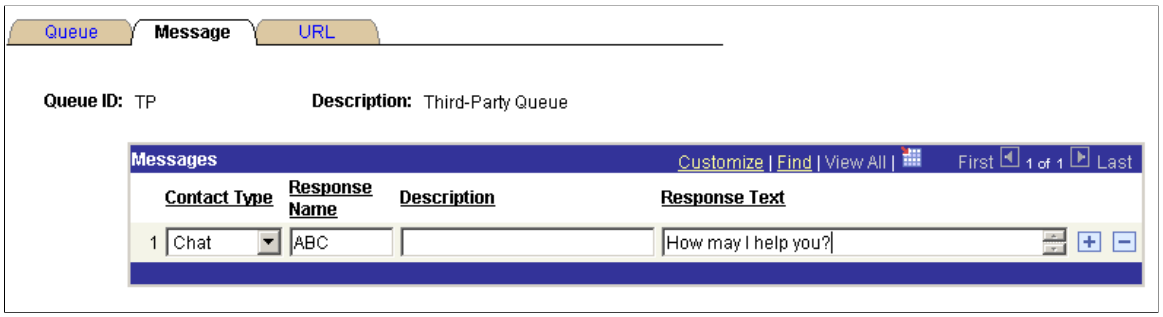
Field or Control	*Description*
Active	Select *Active* or *Inactive* from the drop-down list box. The queue server does not send new tasks to an inactive queue. Agents and existing tasks remain on an inactive queue. To receive new queued tasks, the queue must be active. Active and inactive statuses support “follow-the-sun” practices. For example, SALES1 could be supported in the London office, and SALES2 could be supported in the San Francisco office when the London office is closed by activating and inactivating the appropriate queues.

Defining Canned Queue Messages

Access the Message page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Third-Party Configuration** > **MCF Third-Party Queues** > **Message**

This example illustrates the fields and controls on the Message page. You can find definitions for the fields and controls later on this page.



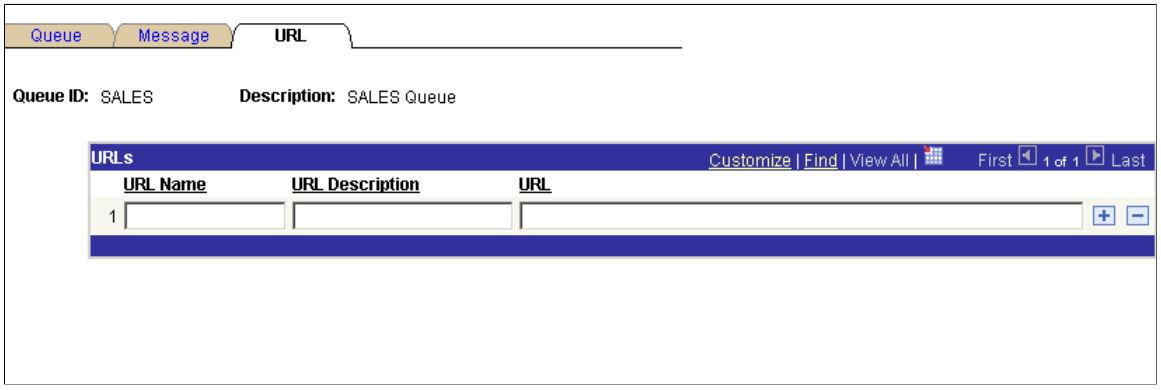
Field or Control	*Description*
Contact Type	Select one of the following contact types: • *Chat* • *Email*
Response Name	The response name appears in the agent's **Template Messages** drop-down list box for all agents who belong to this queue. All messages are downloaded when the agent launches the agent chat console by accepting a customer chat. Template messages are not available in collaborative chat.
Response Text	The specified message appears in the client chat window when selected by the agent.

Defining Canned Queue URLs

Access the URL page using the following navigation path:

PeopleTools > MultiChannel Framework > Third-Party Configuration > MCF Third-Party Queues > URL

This example illustrates the fields and controls on the URL page. You can find definitions for the fields and controls later on this page.



Canned queue URLs enable the agent to push a predefined URL to the customer, which appears in a pop-up window for the customer. The agent can edit these URLs in the chat window after selecting them.

Field or Control	*Description*
URL Name	Enter the name of the URL. The URL name appears in the agent's URL drop-down list box for all agents that belong to this queue.
URL Description	Enter a description of the URL. This description appears only on this page to further describe this URL or, for example, its reason for inclusion.
URL	Enter the URL. The URL must include the protocol (http://) and any required parameters. All canned queue URLs defined for the queue are downloaded when the agent launches the agent chat console by accepting a customer chat. These URLs are not available in collaborative chat. If you send a canned queue URL that is a PeopleSoft Pure Internet Architecture URL, be sure that the recipient has permissions to access that portal, node, or page.

Defining PeopleSoft MCF Agents for a Third Party

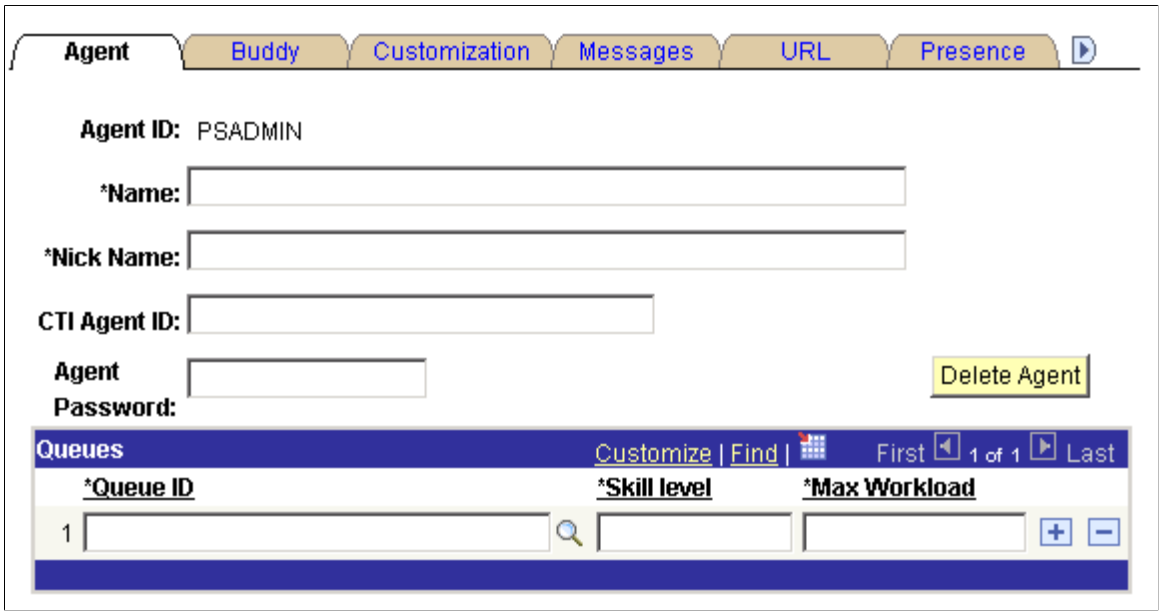
To define PeopleSoft MCF agents for a third party, use the (MCF_TP_AGENT_CMP) component. This section discusses how to define PeopleSoft MCF Agents for a third party.

Creating PeopleSoft MCF Agents for a Third Party

Access the Agent page using the following navigation path:

PeopleTools > MultiChannel Framework > Third-Party Configuration > MCF Third-Party Agents

This example illustrates the fields and controls on the Agent page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Agent ID	Specifies the agent ID.
Name	Enter the full name of this agent in (lastname,firstname) format. The agent name appears in other agents' buddy lists.
Nick Name	Enter a short name for this agent. The agent nickname identifies this agent in chat sessions and logs.
CTI Agent ID	Enter a CTI agent ID for a CTI agent.
Agent Password	Enter a password for the agent.

Field or Control	*Description*
Queue ID	Enter the ID of a queue to which this agent is assigned. Each agent may be assigned to more than one queue. An agent can log on to only one queue at a time from the MultiChannel Console.
Skill	Select the skill level of this agent for the tasks assigned for this queue. This field is required. This option is used by third parties for task assignment. Third parties define skill levels based on business needs, requirements, policy, and so on.
Maximum Workload	Select the maximum load that this agent can be assigned before tasks are held or assigned to other agents. This field is required. This option is used by third parties for task assignment. Third parties define maximum workload based on business needs, requirements, policy, and so on.

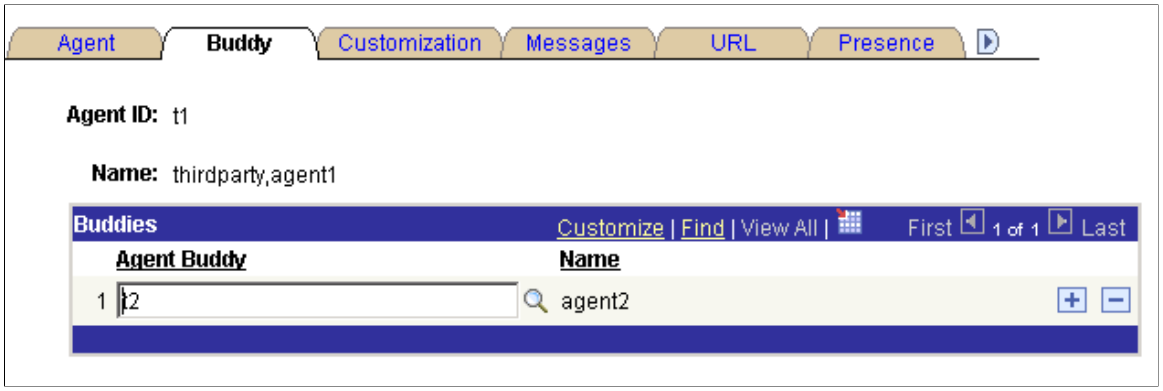
Note: Do not delete a queue from an agent's list unless that agent has no open accepted tasks on that queue.

Setting Up Buddy Lists

Access the Buddy page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Third-Party Configuration** > **MCF Third-Party Agents** > **Buddy**

This example illustrates the fields and controls on the Buddy page. You can find definitions for the fields and controls later on this page.



The agent's buddy list facilitates collaborative chat and chat conferencing.

Field or Control	*Description*
Agent Buddy	Select another agent with whom this agent can have a chat session or can ask to conference onto another chat.
Name	Displays the buddy agent's nickname. The name appears in the buddy list on the MultiChannel Console or agent chat window.

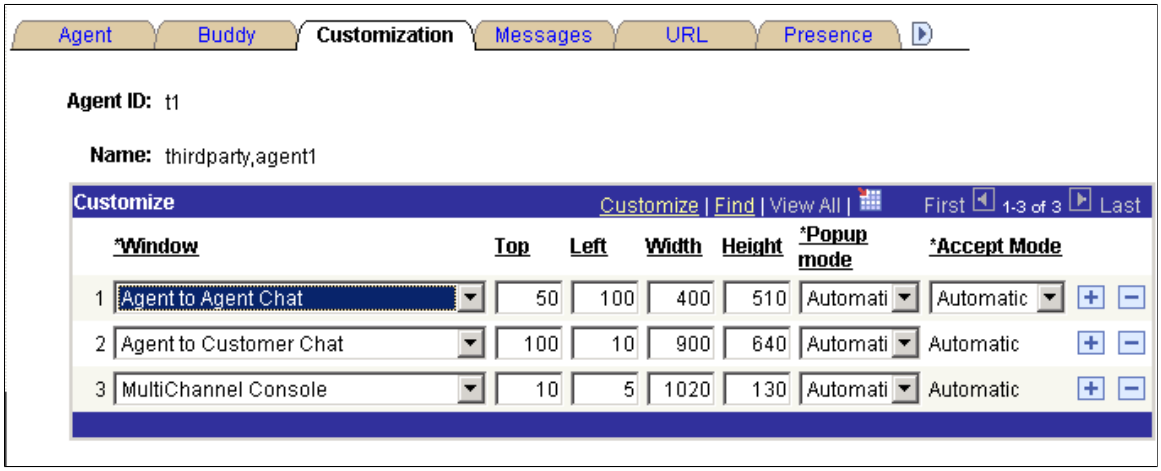
An agent's presence as shown in the buddy list on the MultiChannel Console or on the Invite Agent list on the chat console indicates the agent's availability for chat or conference.

Customizing Windows

Access the Customization page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Third-Party Configuration** > **MCF Third-Party Agents** > **Customization**

This example illustrates the fields and controls on the Customization page. You can find definitions for the fields and controls later on this page.



Set the initial agent window placement and size by specifying parameters on this page. An agent may resize and move the windows.

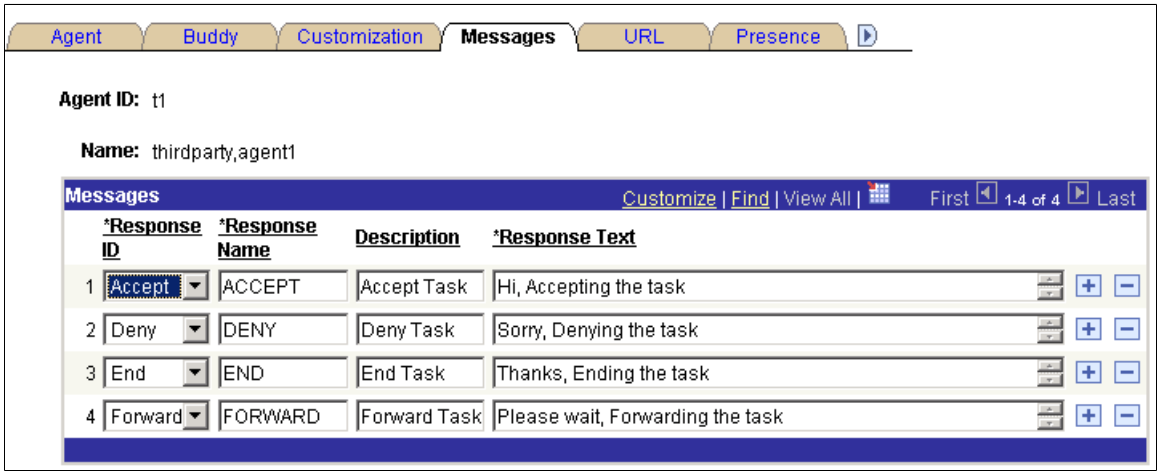
Field or Control	*Description*
Window	Select the window to which the specified configuration applies. Select from: • *Agent to Agent Chat* • *Agent to Customer Chat* • *Email* • *Generic Alert* • *Grab URL* • *MultiChannel Console*
Top and **Left**	Enter the distance in pixels from the top and left edge of the screen when the window first appears.
Width and **Height**	Enter the width and height, in pixels, of the window when it first appears.
Popup Mode	Select from: *Automatic:* The window appears automatically. For customer-initiated chat or tasks initiated from the EnQueue built-in function, the task is automatically accepted as well. For agent-initiated chat, the agent can elect not to accept the task; in effect, the agent can preview the task. If this is the desired behavior, select *Manual* from the **Accept Mode** drop-down list box. If you want agent-to-agent tasks to function like customer-initiated tasks, select *Automatic* from the **Accept Mode** drop-down list box. *Manual:* The window does not appear until the agent clicks the task on the agent MultiChannel Console. For customer-initiated chat or tasks initiated from the EnQueue() built-in function, clicking the task means that the task is automatically accepted as well. For agent-initiated chat, the behavior depends on the setting for the **Accept Mode** field.
Accept Mode	Select from: *Automatic:* Agent-to-agent chats are automatically accepted without requiring the agent to click the icon. *Manual:* Agent-to-agent chats require the agent to click the icon. Accept mode affects only collaborative chat.

Defining Messages

Access the Messages page using the following navigation path:

PeopleTools > MultiChannel Framework > Third-Party Configuration > MCF Third-Party Agents > Messages

This example illustrates the fields and controls on the Messages page. You can find definitions for the fields and controls later on this page.



An agent can create personalized responses in addition to the system responses defined for each queue.

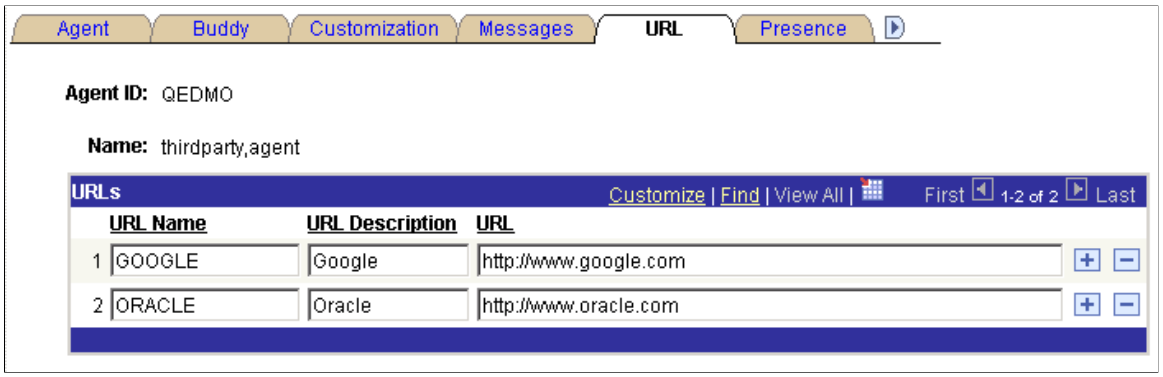
Field or Control	*Description*
Response ID	Responses, except those identified as Other, are linked to specific events. These responses, except for those identified Other, are always sent on these events from this agent. If an agent does not have a customized response for a specific event, it is read from a default response set in the Message Catalog. The response text set here overrides the default text set in the Message Catalog. Select from: • *Abandon:* A chat is abandoned when a chat initiator closes the chat window before the chat is accepted by an agent. This response appears when the agent accepts the abandoned chat. • *Accept:* This response is automatically sent to the chat initiator when an agent accepts a chat in response to a chat request that does not include a question. If the chat request includes a question, the agent's Answer Question text is sent in response instead of the Accept response. • *Answer Question:* This response is automatically sent to the chat initiator when an agent accepts a chat in response to a chat request that includes a question. • *Deny:* Only applies to collaborative chat. If an agent elects not to accept a chat, this response is automatically sent to the chat initiator. • *End:* If either party exits a chat after the chat is accepted, this response is displayed from the agent. • *Forward:* If the agent forwards a chat session to another queue, this response is sent to the customer. • *Other:* These responses are never automatically sent in a chat session. Their message names appear in the **Template Messages** drop-down list box on the agent chat page. These messages are appended to the template messages (chat responses) defined for the queue.
Response Name	This name appears in the agent's template response drop-down list box.
Response Text	Enter the response text to appear in the chat window.

Specifying Agent-Specific URLs

This page defines URLs that this agent can send to a client browser. Access the URL page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Third-Party Configuration** > **MCF Third-Party Agents** > **URL**

This example illustrates the fields and controls on the URL page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
URL Name	The URL name appears in the agent's URL drop-down list box.
URL Description	This description appears only on this page, to further describe this URL or, for example, its reason for inclusion.
URL	Enter the URL. The URL must include the protocol (http://) and any required parameters. All URLs defined for the agent are downloaded when the agent launches the agent chat console by accepting a customer chat. These URLs are not available in collaborative chat. If you send a PeopleSoft Pure Internet Architecture URL, be sure that the recipient has permissions to access that portal, node, or page.

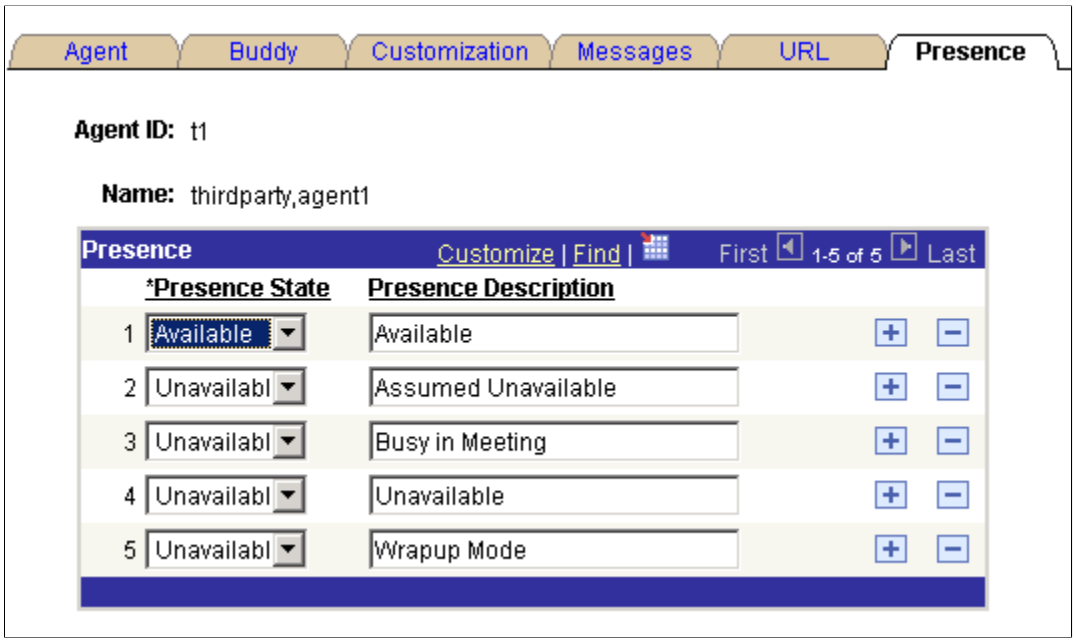
Defining Agent's Presence

Each agent can customize the presence description displayed when the agent is available or unavailable. The queue server only understands the presence state, available or unavailable, but you can enter more specific presence descriptions when displaying or logging an agent's presence. If you do not enter presence descriptions, default values are used.

Access the Presence page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Third-Party Configuration** > **MCF Third-Party Agents** > **Presence**

This example illustrates the fields and controls on the Presence page. You can find definitions for the fields and controls later on this page.



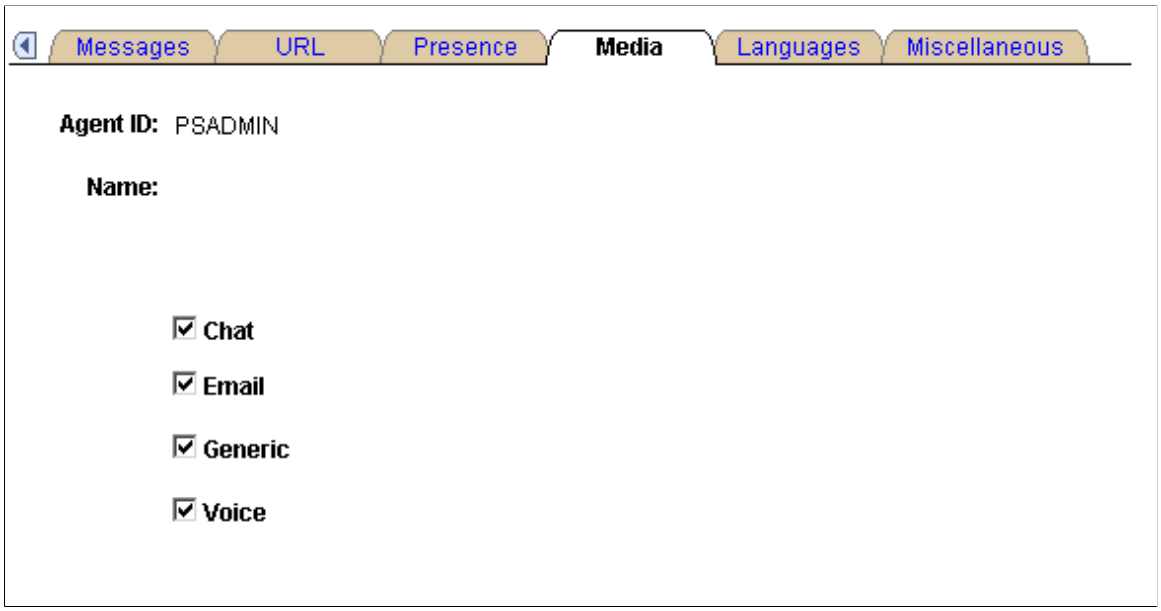
Field or Control	*Description*
Presence State	Select *Available* or *Unavailable.*
Presence Description	Enter a description for each agent state. The description appears in logs of agent activity and when agent presence is displayed. **Note:** An available state has only one presence description, *Available.* For an unavailable state, you can enter any presence description, such as tea break, coffee break, lunch, and so on.

Specifying the Media

The media page defines the capability of each agent to perform a task. Access the Media page using the following navigation path:

PeopleTools > MultiChannel Framework > Third-Party Configuration > MCF Third-Party Agents > Media

This example illustrates the fields and controls on the Media page. You can find definitions for the fields and controls later on this page.



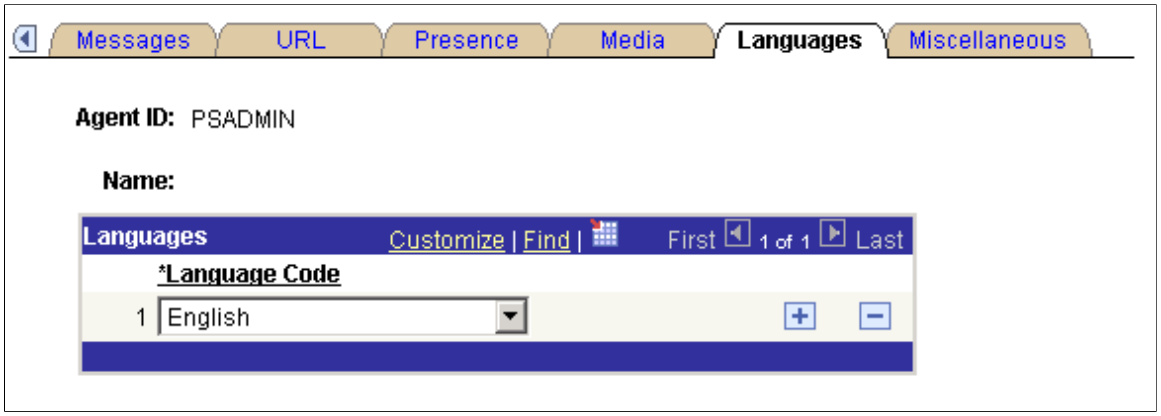
Field or Control	*Description*
Chat	Select this check box if the agent has a capability to chat.
Email	Select this check box if the agent has a capability to email.
Generic	Select this check box if the agent has a capability to perform a generic task.
Voice	Select this check box if the agent has a capability to perform a voice task.

Specifying Languages That an Agent Supports

Access the Languages page using the following navigation path:

PeopleTools > MultiChannel Framework > Third-Party Configuration > MCF Third-Party Agents > Languages

This example illustrates the fields and controls on the Languages page.



Enter the languages that each agent is qualified to support.

The agent is assigned only tasks that have been enqueued with a language code in the agent's language list. For the EnQueue built-in function, the language code is specified as a parameter. For the InitChat built-in function, the language code is determined by the user profile of the initiator.

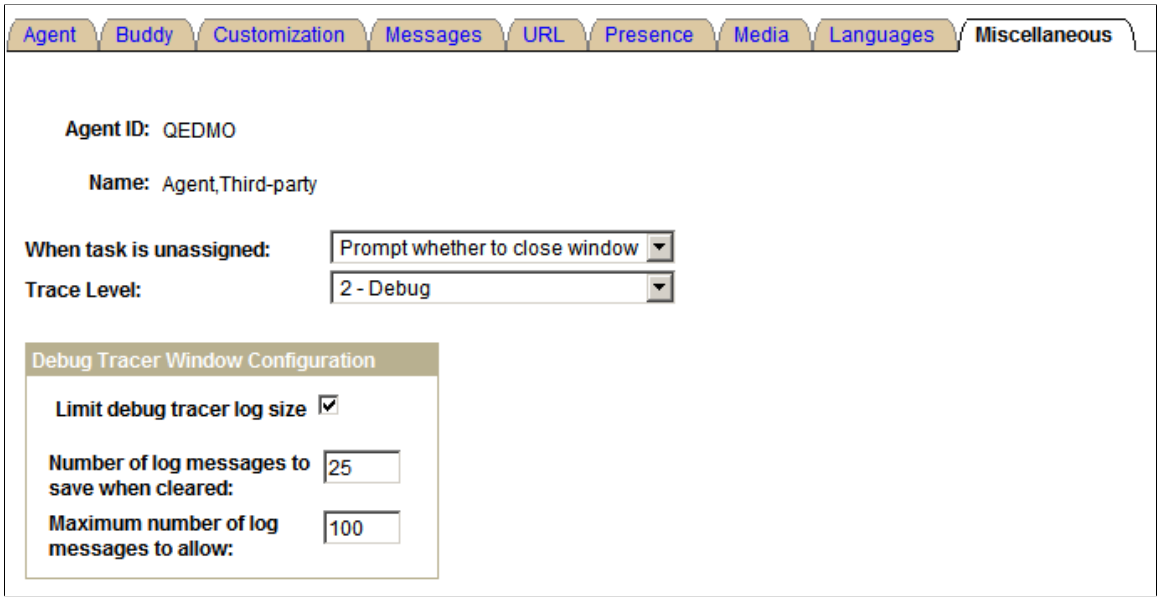
If you do not enter a language code for a new agent, the default value is English.

Specifying Miscellaneous Parameters

Access the Miscellaneous page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Third-Party Configuration** > **MCF Third-Party Agents** > **Miscellaneous**

This example illustrates the fields and controls on the Miscellaneous page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
When task is unassigned	Select from the following options the action that occurs when a task assigned to an agent is unassigned: • *Prompt whether to close window (default).* • *Close the task window.* • *Do not close the task window.*

Field or Control	*Description*
Trace Level	Select from the following log trace levels: • *0 - None* • *1 - Information* • *2 - Debug* **Note:** If a value other than *0* is selected, a tracer window appears to display activities and events on the chat or MultiChannel Console for debugging purposes.
Limit debug tracer log size	This check box is enabled when the value entered for **Trace Level** is not 0. Select this check box to enable the agent to clear the tracer log based on **Number of log messages to save when cleared** and **Maximum number of log messages to allow**. If the check box is cleared, the tracer log will not be cleared and the **Number of log messages to save when cleared** and **Maximum number of log messages to allow** fields will be disabled.
Number of log messages to save when cleared	Specify the minimum number of recent tracer log messages that should be maintained in the tracer window.
Maximum number of log messages to allow	Specify the maximum number of tracer log messages that will be maintained in the tracer window.

Note: Number of log messages to save when cleared and **Maximum number of log messages to allow** fields are required if the **Limit debug tracer log size** check box is selected.

Configuring PeopleSoft MCF Tasks for a Third Party

This section discusses how to configure tasks. To configure tasks, use the MCF Task (MCF_TP_TASKCFG_CMP) component.

See Configuring Tasks.

Configuring CTI for a Third Party

This section discusses how to configure CTI using third-party routing system. To configure CTI, use the MCF_TP_CTI_CONFIG component.

See Configuring PeopleSoft CTI.

Communicating with Customers and Agents Using Chat

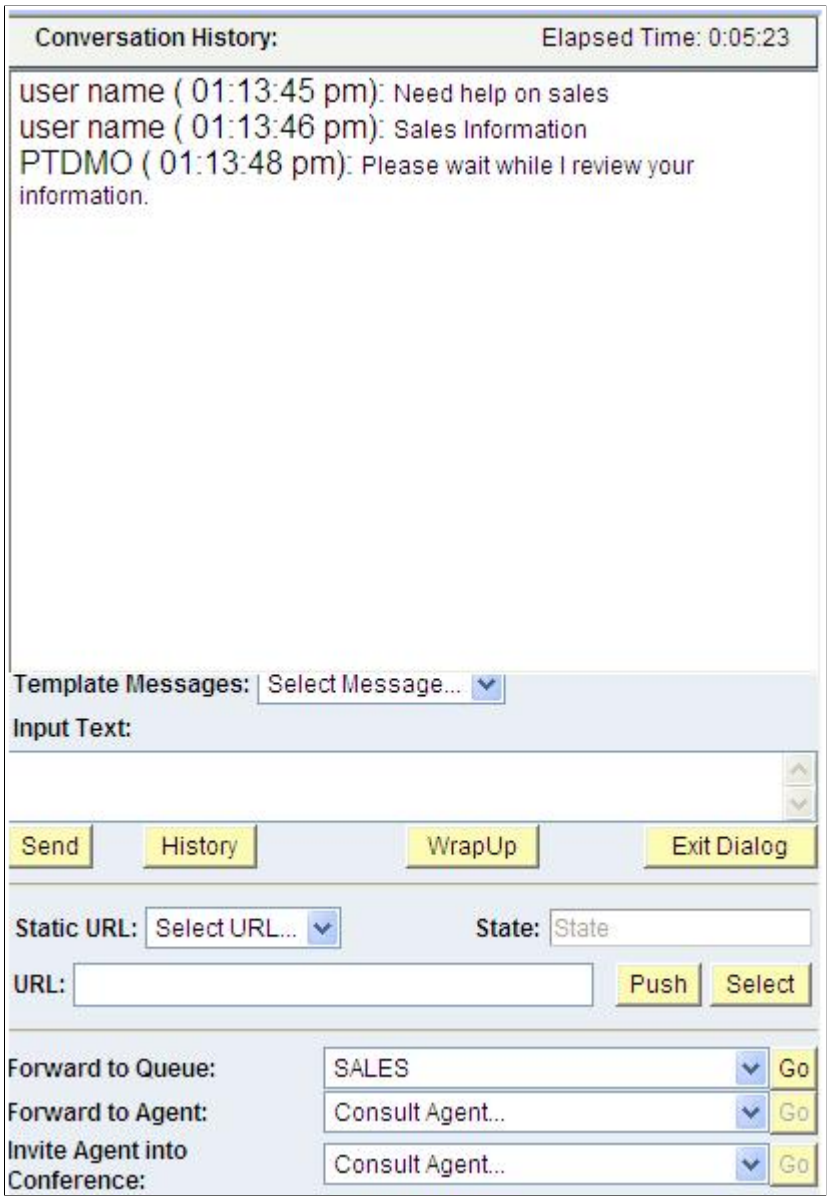
This section discusses how to use the Agent Chat window.

Using the Third-Party Chat Window

When you accept a chat task, a chat window opens.

The format of the agent chat window and the contents of its right pane are delivered as part of PeopleSoft MultiChannel Framework. The content of the left pane is determined by application developers, and is passed as a parameter to the InitChat() built-in function.

This example illustrates the fields and controls on the third-party agent chat window. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Conversation History	Lists progress of the chat, line by line. If chat logging is enabled, the conversation is recorded in the database chat log by the MCF log server. View the chat log on the Chat Log page. If accessibility features are not turned off in the My Personalizations component, an additional text box appears below the conversation history. The most recent customer input appears in the secondary text box, which can be read by screen reading software.

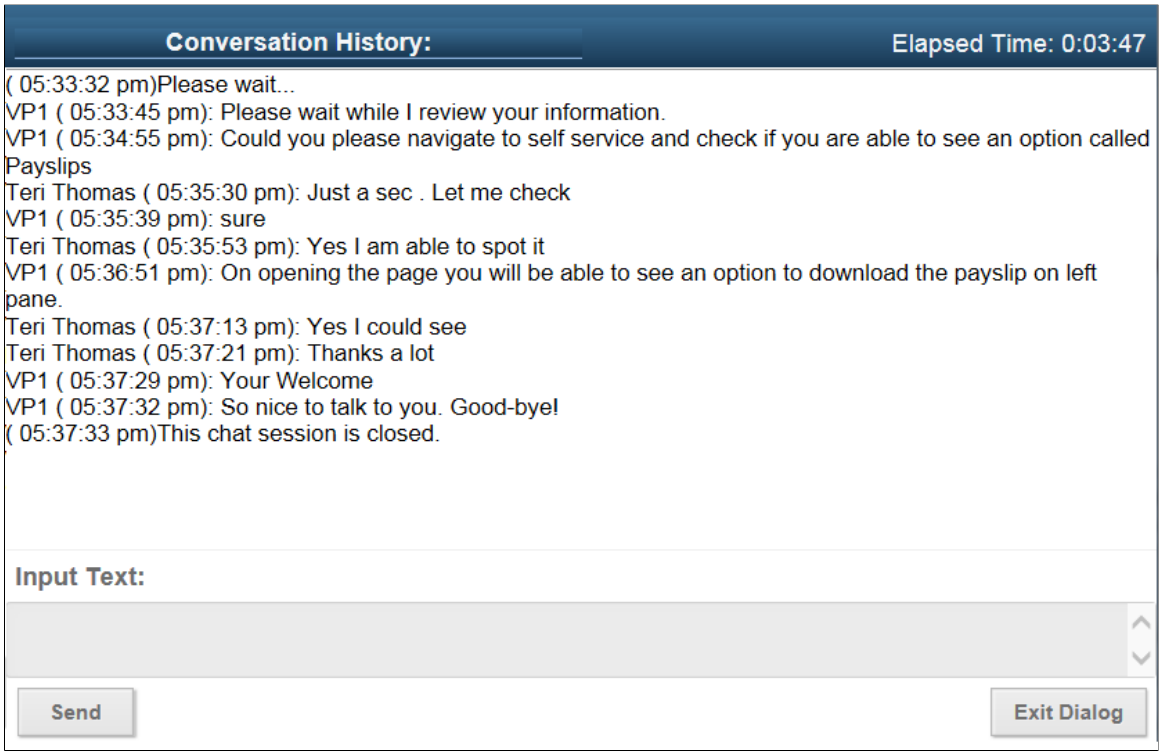
Field or Control	*Description*
Template Messages	Send the customer a standard message by selecting one from the list. The message text appears in the **Input Text** text box. Click **Send** to send the message. Edit template messages for each queue on the queue Chat Responses page.
Input Text	To respond to the customer, enter text, and then click **Send** or press **Enter.** The maximum size of the text that can be sent at one time is 4096 bytes (4 kilobytes).
Send	Click to send the contents of the **Input Text** text box.
Exit Dialog	Click to end the chat. The chat window remains open, enabling you to return to the page for follow-up work after the customer exits the dialog.
Static URL and Push	To send a static URL to the customer, select a URL name, then click **Push.** When a URL is pushed to a customer, a browser window for that URL is launched on the customer's workstation.
URL	Displays static and grabbed URLs.
Select	Click to launch an application page, from which a URL can be returned to populate the URL field. The format of the URL wizard page that appears when you click **Grab** is defined by application developers. PeopleSoft supplies a sample URL wizard page. See Using the URL Wizard.
Forward to Queue	To forward the current chat to another queue, select the queue.
History	Click to get the history for the chat conversation.
Wrapup	Click to write wrapup comments for the chat. Wrapup comments are stored in PS_MCFCHATLOG.
Forward to Agent	To forward the current chat to another agent, select the queue and click Go.

Field or Control	*Description*
Invite Agent into Conference	To request that another agent join the chat, select the agent and click **Go.**

Using the Customer Chat Window

When a customer initiates a chat, a chat window appears:

This example illustrates the fields and controls on a customer chat window. You can find definitions for the fields and controls later on this page.



The format and content of the customer chat window is delivered as part of PeopleSoft MultiChannel Framework. The customer chat window includes a conversation history text box, input text box, and **Send** and **Exit Dialog** buttons.

If accessibility features are not turned off in the My Personalizations component, an additional text box appears below the conversation history. The most recent agent input appears in the secondary text box, which can be read by screen reading software.

Field or Control	*Description*
Input Text	The customer enters text here. The maximum size of the text that can be sent at one time is 4096 bytes (4 kilobytes).

Field or Control	*Description*
Send	Click to send the input text to the agent.
Exit Dialog	Click to end the chat and close the chat window.

Note: The agent collaborative chat window is substantially the same as the customer chat window.

Viewing Event Logs for a Third Party

This section discusses how to view event logs. To view event logs, use the (MCF_TP_EVNTLOG_CMP) component.

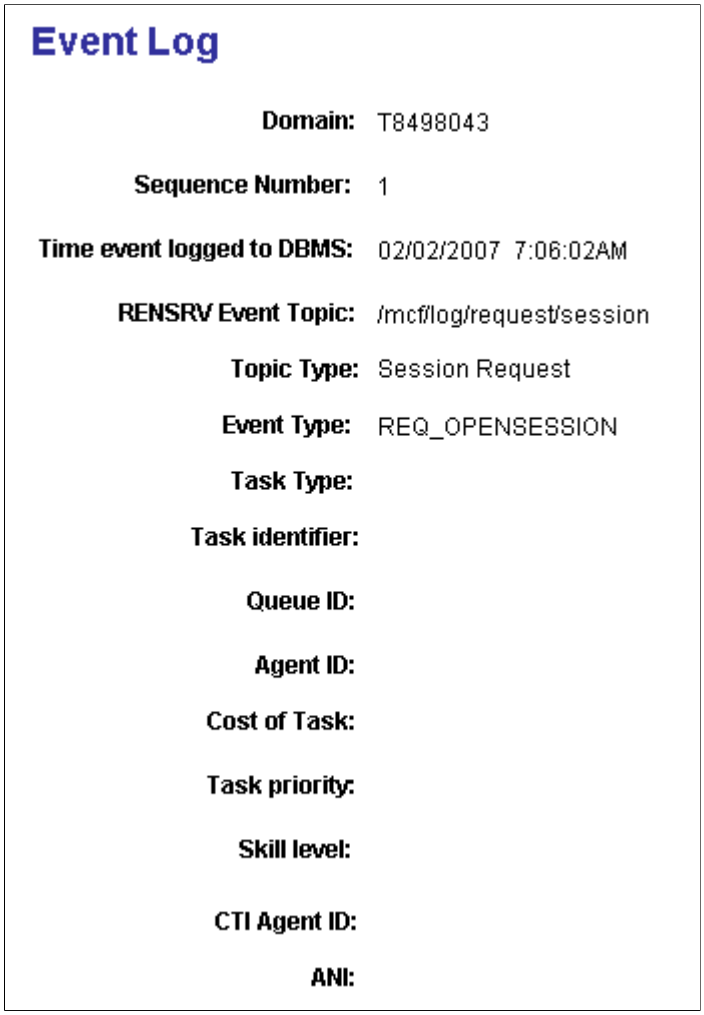
This example illustrates an event log.

Domain	Time event logged to DBMS	Event type	Task Type	Queue ID	Agent ID	CTI Agent ID	ANI	DNIS	This DN	Other DN	Call ID
MCF1	12/07/2005 5:02:30PM	Restrt Ack	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/12/2005 12:09:40PM	User RQ	Voice	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/12/2005 12:09:40PM	User Event	Voice	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/15/2005 2:01:29PM	Q Shutdown	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/15/2005 2:01:29PM	Q Shutdown	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/15/2005 2:01:34PM	Restrt Ack	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/19/2005 1:33:14PM	Q Shutdown	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/19/2005 1:33:14PM	Q Shutdown	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/19/2005 1:34:02PM	Restrt Ack	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/19/2005 2:23:06PM	Accept	Generic	MARKETING	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/19/2005 2:23:06PM	DB Cntct	Generic	MARKETING	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/21/2005 11:05:00AM	Q Shutdown	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/21/2005 11:05:00AM	Q Shutdown	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)
MCF1	12/28/2005 11:28:28AM	Q Shutdown	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)	(blank)

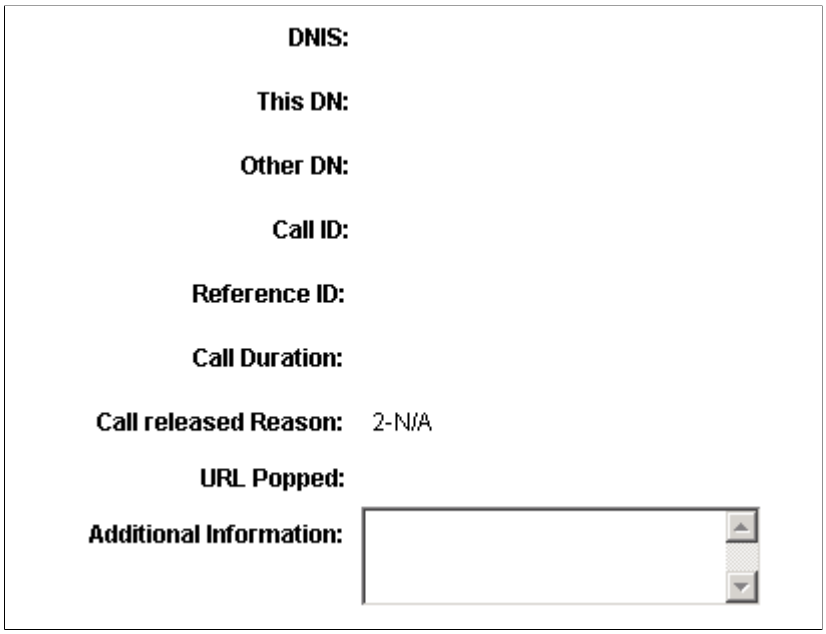
Viewing Event Logs for a Third Party

Select any event from the Event Log page to display the details of any event.

This example illustrates the fields and controls on the Event Log page (page 1 of 2).



This example illustrates the fields and controls on the Event Log page (page 2 of 2).



Viewing Broadcast Logs for a Third Party

To view the broadcast log, access the broadcast log (MCF_TP_BCASTLG_CMP) component.

See Viewing Broadcast Logs.

Working with Third-Party Sample Pages

To demonstrate tools and functionality, use the Sample Pages (MCF_TP_DEMO_CMP) component. These sample pages demonstrate how JSMCAPI can be used with the PeopleCode behind it. The third party may develop their own interface, which may look significantly different from the sample pages included in the section.

Using the Customer Chat Sample Page

See Using the Customer Chat Sample Page.

Using the Generic Event Sample Page

See Using the Generic Event Sample Page.

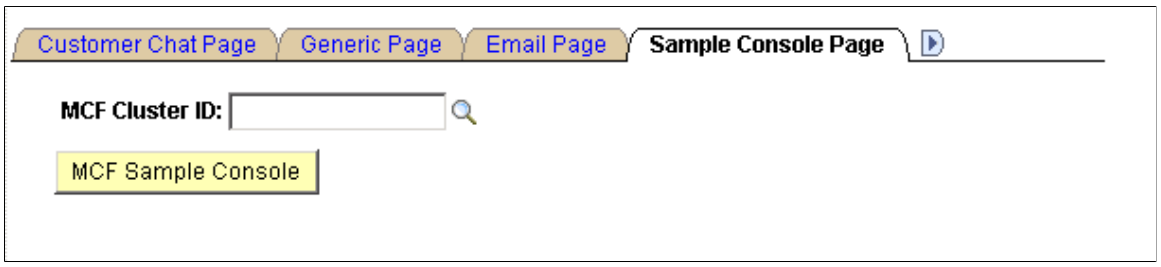
Using the Email Sample Page

See Using the Email Sample Page.

Using the Sample Console Page

Access Sample Console Page using the navigation path: **PeopleTools** > **MultiChannel Framework** > **Third-Party Configuration** > **MCF Third-Party Sample Console** > **Sample Console Page**. Click the lookup button and select the REN cluster.

This example illustrates the fields and controls on Sample Console Page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
MCF Sample Console	Click to initiate the sample console. **Note:** To demonstrate the MCF sample console, you must have an MCF cluster running and communicating with the specified REN server cluster.

After clicking **MCF Sample Console,** a new browser window displaying the sample console appears along with a tracer window. The Sample Console contains the following group boxes, which are detailed in the following topics:

- Open/Close Session
- Register User to Session
- Register Group, Login User to Group, and User-Group States
- Address Operations
- Register Extension to Session, and Extension Operations
- Buddies
- Buddies Bulk Registration
- Groups Bulk Registration
- Error Messages / Information
- Server State and Broadcasts

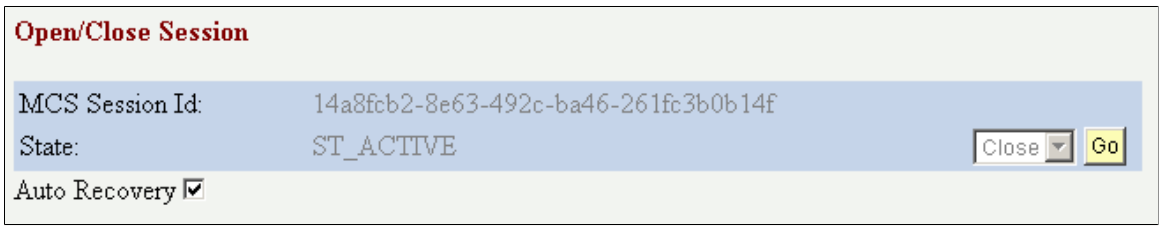
Note: To enable the new browser window to appear, disable any pop-up blocking software for your browser.

Before accessing the **Open/Close Session** group box, enable the third-party routing server.

Open/Close Session

Access the **Open/Close Session** group box. (Select **PeopleTools** > **MultiChannel framework** > **Third-Party Configuration** > **MCF Third-Party Sample Console**. Enter the MCF Cluster ID and click the Sample Console button).

This example illustrates the fields and controls on the Open/Close Session group box. You can find definitions for the fields and controls later on this page.

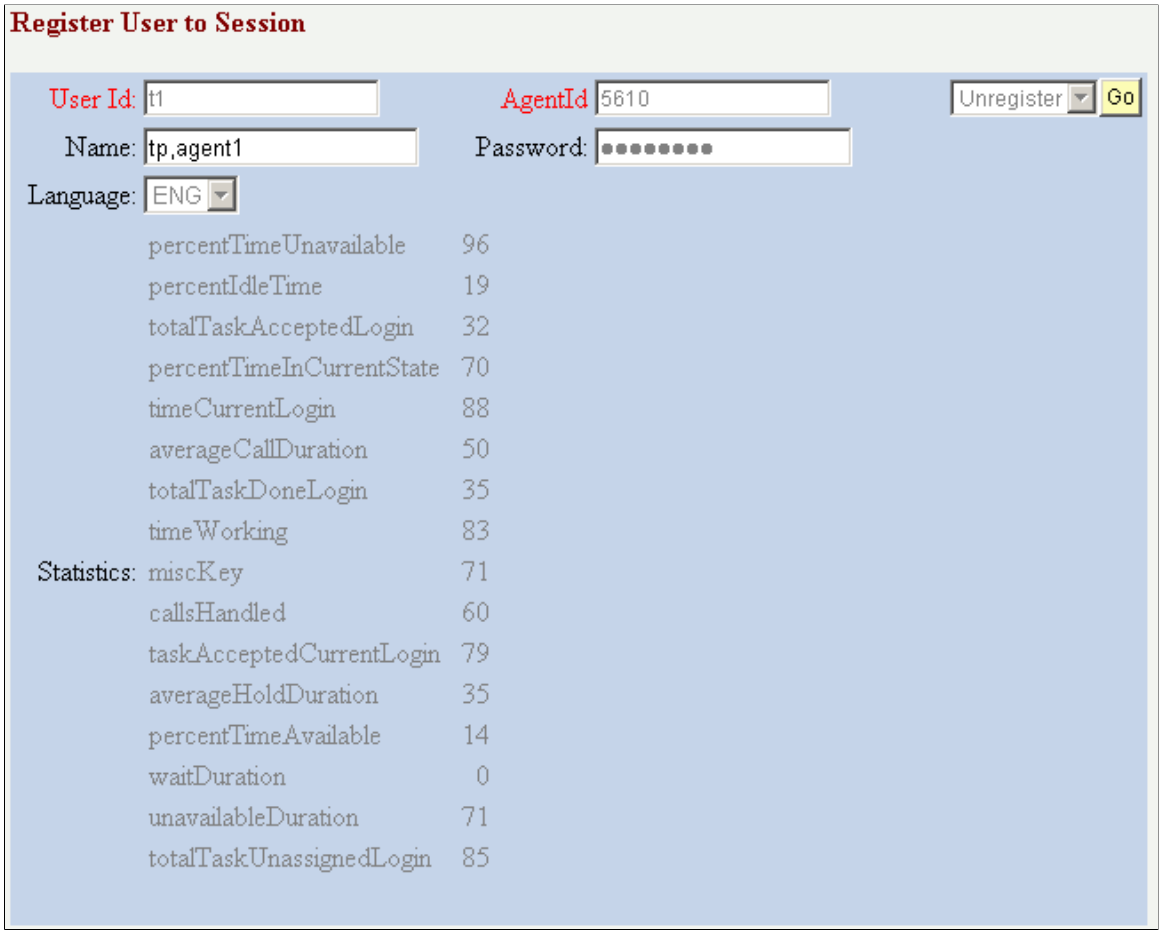


Field or Control	*Description*
MCS Session Id	Displays the session ID of the MultiChannel server.
State	Displays the state of the session. **Note:** If the state is *ST_FAILED*, check the third-party routing server. It must be up and running.

Register User to Session

Access the **Register User to Session** group box. (Select **PeopleTools** > **MultiChannel framework** > **Third-Party Configuration** > **MCF Third-Party Sample Console**. Enter the MCF Cluster ID and click the Sample Console button).

This example illustrates the fields and controls on the Register User to Session group box. You can find definitions for the fields and controls later on this page.



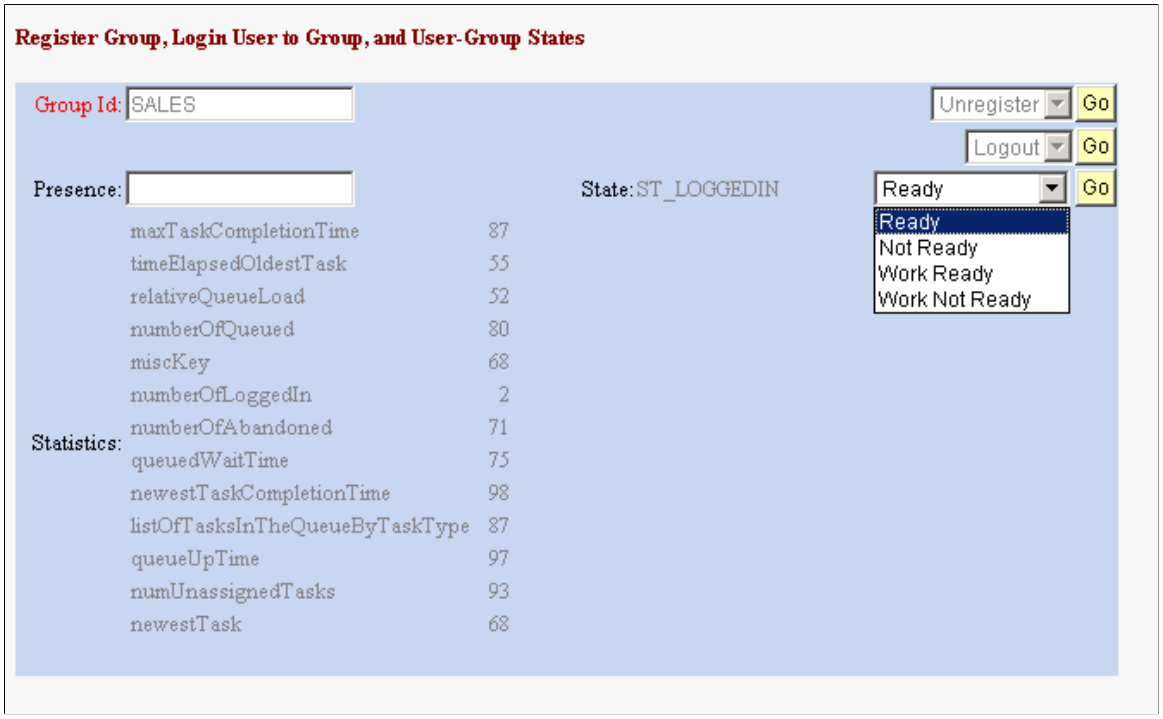
Register a user to the session in the **Register User to Session** group box.

Field or Control	*Description*
User Id	Enter user ID of the agent.
Agent Id	Enter agent's ID. **Note:** Because a CTI agent can have an agent ID different from the agent's user ID, both values are required. If the agent has a password, the password is also required.
Name	Enter the name of the agent.
Password	Enter the password if the agent has a password.
Language	Select the language. The language field is reserved for future use. The default language is English.
Statistics	Displays the agent statistics.

Register Group, Login User to Group, and User-Group States

Access the **Register Group, Login User to Group, and User-Group States** group box. (Select **PeopleTools** > **MultiChannel framework** > **Third-Party Configuration** > **MCF Third-Party Sample Console**. Enter the MCF Cluster ID and click the Sample Console button).

This example illustrates the fields and controls on the Register Group, Login User to Group, and User-Group States group box. You can find definitions for the fields and controls later on this page.



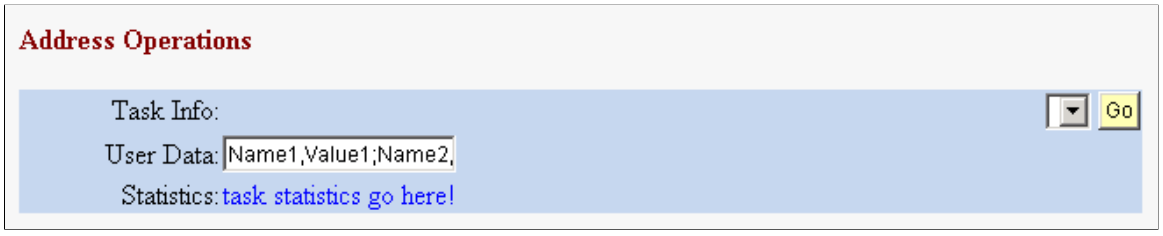
Field or Control	*Description*
Group Id	Enter the name of the group to which the agent will be registered. The group ID is the same as the name of the queue to which the agent is logged in. To register the agent with the specified group, click the **Go** button next to the **Register** field. **Note:** After you click **Go,** the field value changes to *Unregister.*
Login	Click the **Go** button to log in if *Login* appears in the drop-down list box. Click the **Go** button to log out if *Logout* appears in the drop-down list box. **Note:** After you log in to the group, a new pop-up window opens, prompting you for reason code data. If you register to an invalid group, a pop-up window opens showing the error messages. See Understanding JSMCAPI.

Field or Control	*Description*
Presence	Enter a presence description to associate with the state selected in the drop-down list box. This presence value is different from any presence values predefined in the agent configuration. The agent cannot select from the predefined values.
State	Select an agent state from the following options: • *Ready* • *Not Ready* • *Work Ready* • *Work Not Ready* Each state value has an associated state code that appears in the **State** field. To enable incoming calls, select *Ready* from the drop-down list box and click the **Go** button.
Statistics	Displays the group statistics.

Address Operations

Access the **Address Operations** group box. (Select **PeopleTools** > **MultiChannel framework** > **Third-Party Configuration** > **MCF Third-Party Sample Console**. Enter the MCF Cluster ID and click the Sample Console button).

This example illustrates the fields and controls on the Address Operations group box. You can find definitions for the fields and controls later on this page.



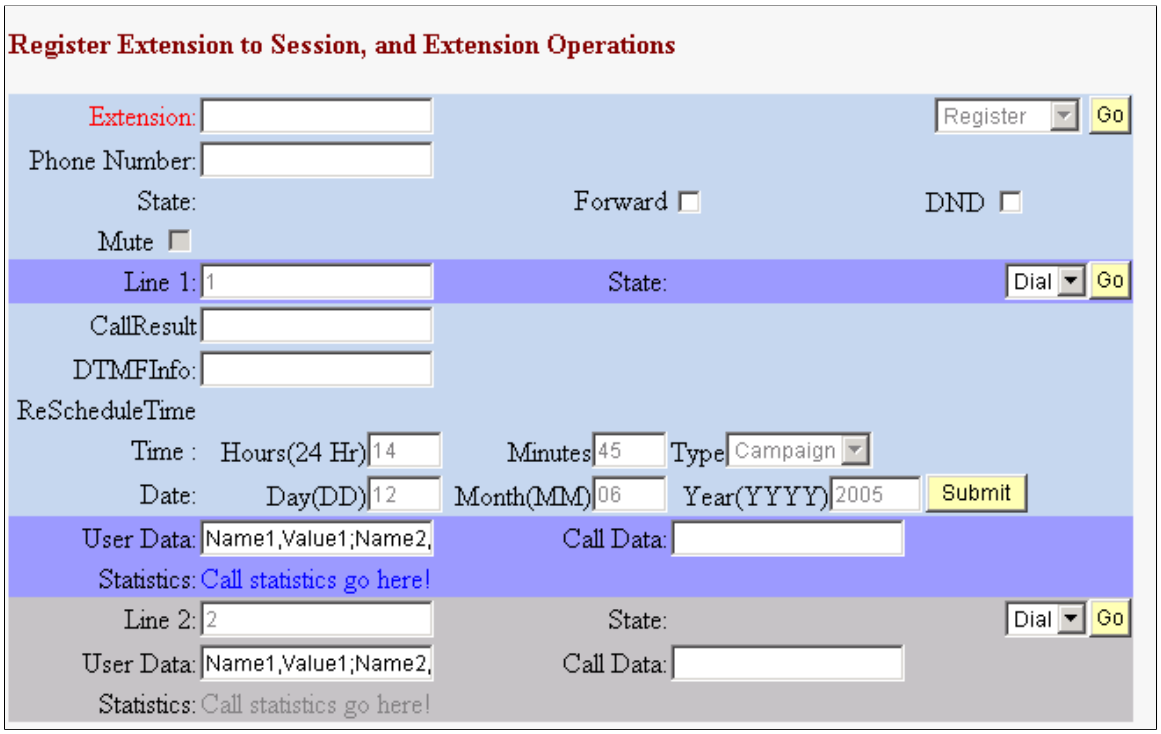
Field or Control	*Description*
Task Info (task information)	Displays task information.
User Data	Displays user data.

Field or Control	*Description*
Statistics	Displays task statistics.

Register Extension to Session and Extension Operations

Access the **Register Extension to Session, and Extension Operations** group box. (Select **PeopleTools** > **MultiChannel framework** > **Third-Party Configuration** > **MCF Third-Party Sample Console**. Enter the MCF Cluster ID and click the Sample Console button).

This example illustrates the fields and controls on the Register Extension to Session and Extension Operations group box . You can find definitions for the fields and controls later on this page.



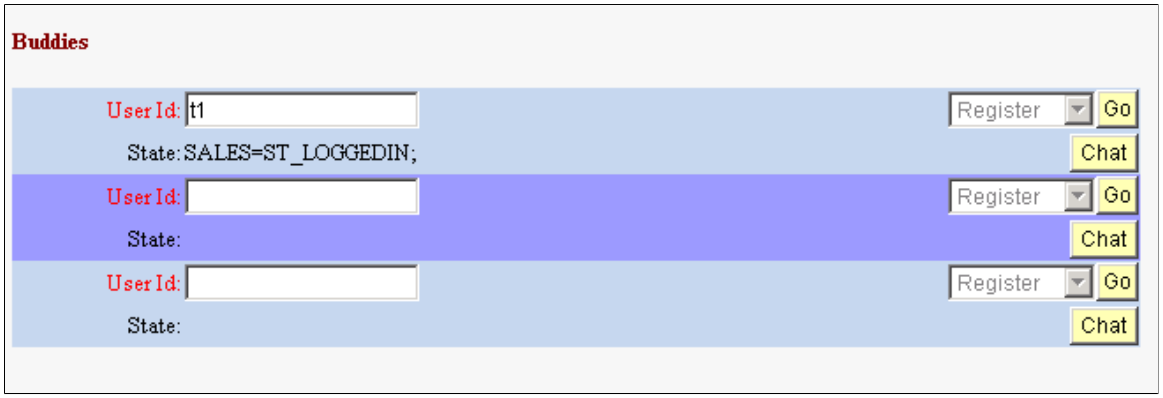
Field or Control	*Description*
Extension	Enter a valid telephone extension number.
Phone Number	Enter a valid telephone number for use in subsequent operations, such as dial, forward, or do-not-disturb (DND).
Mute	Select to mute the call. **Note:** Mute is enabled only when the agent is on the call.
Forward	Select to forward incoming calls to the specified number.

Field or Control	*Description*
DND (do not disturb)	Select to put the selected extension in do-not-disturb status.
Line 1 or **Line 2**	Displays a name or value for each line. On the sample console, the value will either be *1* or *2*.
DTMF Info (Dual Tone Multi-Frequency information)	Select to send DTMF digits on the line.
CallResult	Select to set call result on the line.
ReScheduleTime	Select to set reschedule time on the line.
Type	Select to set reschedule type on the line.
User Data	Enter name-value pairs representing user data to be attached to a call.
Call Data	Enter any call data to be attached to the call. Call data includes: • *ANI* • *DNIS* • *THISDN* • *OTHERDN*

Buddies

Access the **Buddies** group box. (Select **PeopleTools** > **MultiChannel framework** > **Third-Party Configuration** > **MCF Third-Party Sample Console**. Enter the MCF Cluster ID and click the Sample Console button).

This example illustrates the fields and controls on the Buddies group box. You can find definitions for the fields and controls later on this page.

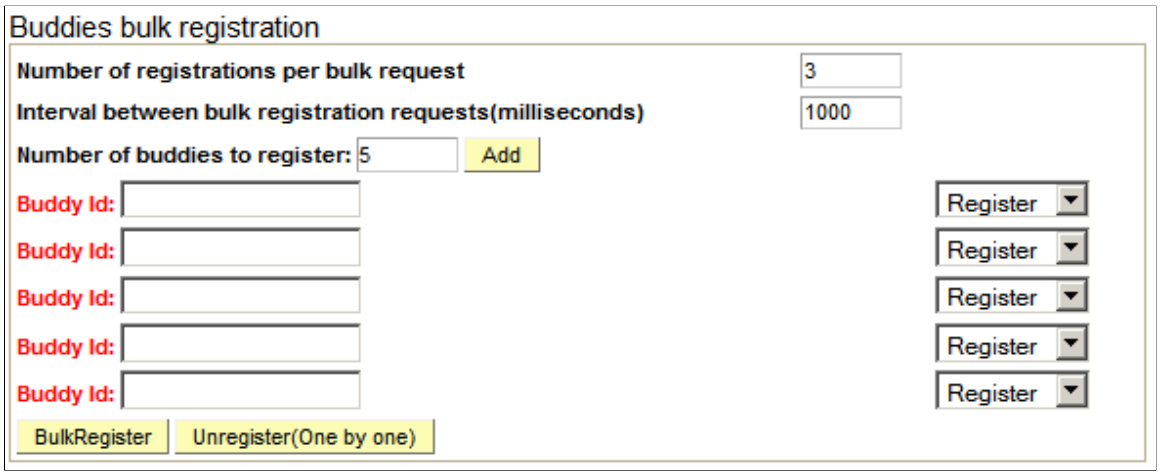


Field or Control	*Description*
User Id	Enter the user ID of the buddy with whom you want to chat.
State	Displays the state of the buddy.
Chat	Click to initiate an agent-to-buddy chat session.

Buddies Bulk Registration

Access the **Buddies bulk registration** group box. (Select **PeopleTools** > **MultiChannel framework** > **Third-Party Configuration** > **MCF Third-Party Sample Console**. Enter the MCF Cluster ID and click the Sample Console button).

This example illustrates the fields and controls on the Buddies bulk registration group box. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Number of registrations per bulk request	Enter the number of registrations per bulk request.
Interval between bulk registration requests(milliseconds)	Enter the interval between bulk registration requests in milliseconds.
Number of buddies to register	Enter the number of buddies to add and click the Add button. Edit boxes for Buddy Id will appear.
BulkRegister	Click to register the buddies. An Unregister button will appear next to each Group Id. You can use this button to unregister a specific group.
Unregister(One by One)	Click to unregister groups one by one.

Groups Bulk Registration

Access the **Groups bulk registration** group box. (Select **PeopleTools** > **MultiChannel framework** > **Third-Party Configuration** > **MCF Third-Party Sample Console**. Enter the MCF Cluster ID and click the Sample Console button).

This example illustrates the fields and controls on the Groups bulk registration group box. You can find definitions for the fields and controls later on this page.

Field or Control	*Description*
Number of registrations per bulk request	Enter the number of registrations per bulk request.
Interval between bulk registration requests(milliseconds)	Enter the interval between bulk registration requests in milliseconds.
Number of groups to register	Enter the number of groups to add and click the Add button. Edit boxes for Group Id will appear.

Field or Control	*Description*
BulkRegister	Click to register the buddies. An Unregister button will appear next to each Buddy ID. You can use this button to unregister a specific buddy.
Unregister(One by One)	Click to unregister buddies one by one.

Error Messages/Information

Access the **Error Messages/Information** group box (Select **PeopleTools** > **MultiChannel framework** > **Third-Party Configuration** > **MCF Third-Party Sample Console**. Enter the MCF Cluster ID and click the Sample Console button).

This example illustrates the fields and controls on the Error Messages/Information group box. You can find definitions for the fields and controls later on this page.

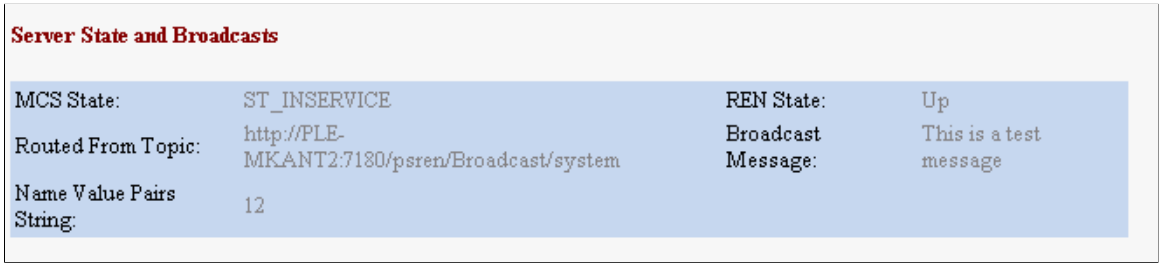
Any error messages associated with the process appear here.

Field or Control	*Description*
Clear	Click the button to clear a displayed error message.

Server State and Broadcasts

Access the **Server State and Broadcasts** group box. (Select **PeopleTools** > **MultiChannel framework** > **Third-Party Configuration** > **MCF Third-Party Sample Console**. Enter the MCF Cluster ID and click the Sample Console button).

This example illustrates the fields and controls on the Server State and Broadcasts group box. You can find definitions for the fields and controls later on this page.

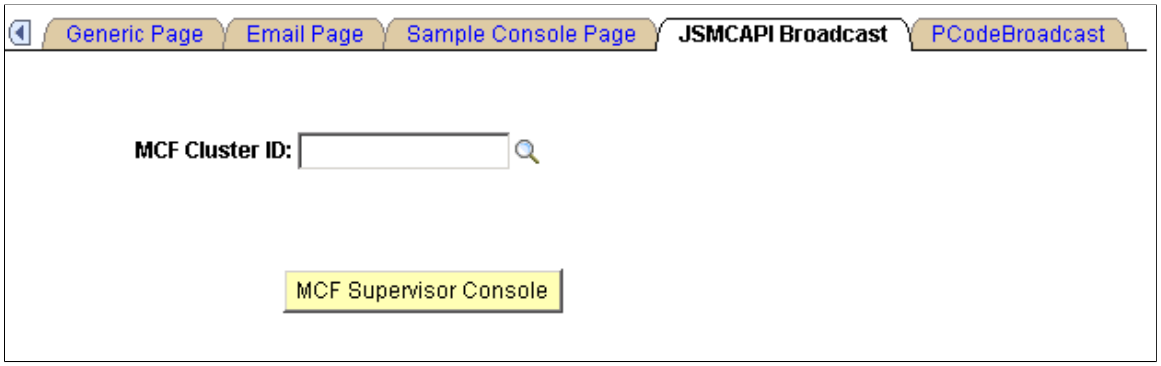


Field or Control	*Description*
MCS State	Displays the MultiChannel server's state.
Routed From Topic	Displays the machine name from where the broadcast message was sent.
Name Value Pairs	Displays the name value pair string.
REN State	Displays the status of the REN server.
Broadcast Message	Displays the supervisor's broadcast message.

Using the MCF Broadcast Page

This section discusses how to use the JSMCAPI Broadcast page for the third party.

This example illustrates the fields and controls on the JSMCAPI Broadcast page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
MCF Cluster ID	Select the REN server cluster on which to test the MCF Supervisor console. **Note:** The MCF Supervisor console is specific to the REN cluster Id selected.
MCF Supervisor Console	Click to initiate the supervisor console. **Note:** To demonstrate the MCF Supervisor console, you must have MCF servers, both queue and log servers, running and communicating with the specified REN server cluster.

After you click **MCF Supervisor Console,** a new browser window appears that displays the sample supervisor console.

Note: To enable the new browser window to appear, disable any pop-up blocking software for your browser.

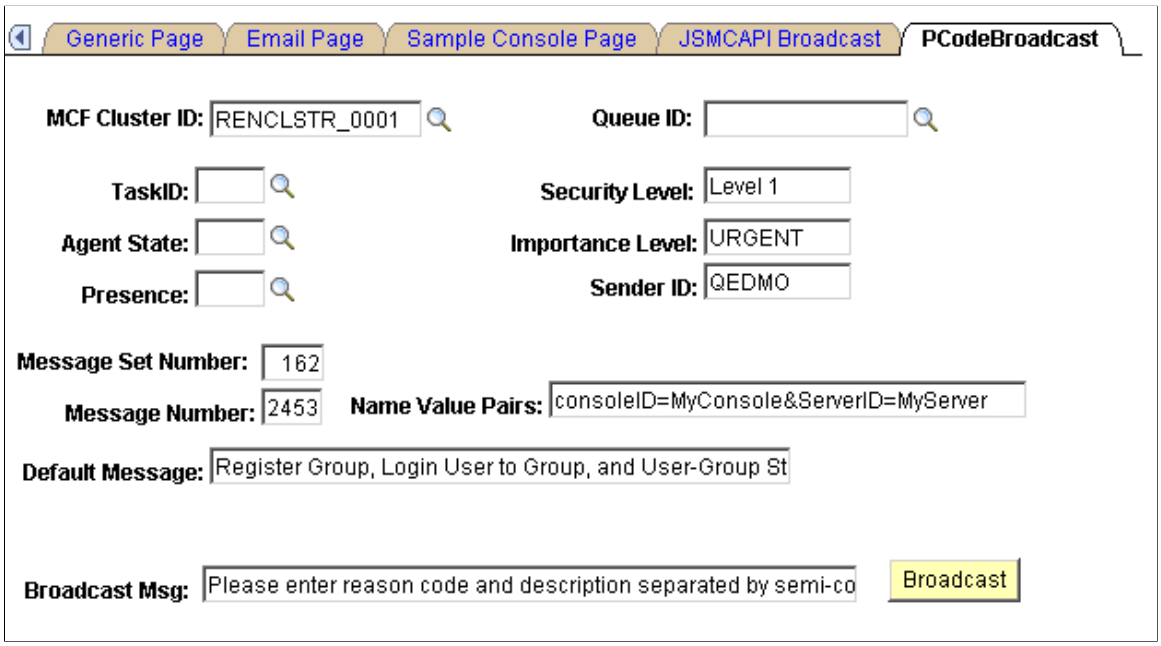
Related Links

Using JSMCAPI Broadcast with MCF Supervisor Console

Using the PCodeBroadcast Page

This section discusses how to use the PCodeBroadcast page.

This example illustrates the fields and controls on the PCodeBroadcast page.



See Using PeopleCode Broadcast.

Chapter 12

Understanding JSMCAPI Classes

Understanding JSMCAPI

This section discusses JSMCAPI, a JavaScript-based application programming interface (API) used to build custom applications, such as MultiChannel consoles, or to enable MultiChannel functionality on the PeopleSoft Pure Internet Architecture page. The API is built on the REN server JavaScript client and is a pure JavaScript API.

JSMCAPI is the interface through which the application developer accesses the JSMCAPI functionality. JSMCAPI provides a set of objects, such as User, Address, Group, and so on. The PSMC, a global object, provides all those methods called to send agent requests to the server; it also provides an event handler callback method for the PeopleSoft MultiChannel Application Programming Interface (PSMCAPI) call backs for events coming from the CTI server.

The JSMCAPI communicates with the PSMCAPI through the REN server using MCP (Multi-Channel Protocol). MCP includes topic and event definition. In general, JSMCAPI:

- Sends agent requests to the MultiChannel Server.

  The requests can be agent state requests such as login, logout, set ready, and so on. Requests can also be call requests, such as dial-out, answer the call, transfer/conference, and so on.

- Receives PBX/Switch events from the CTI server.

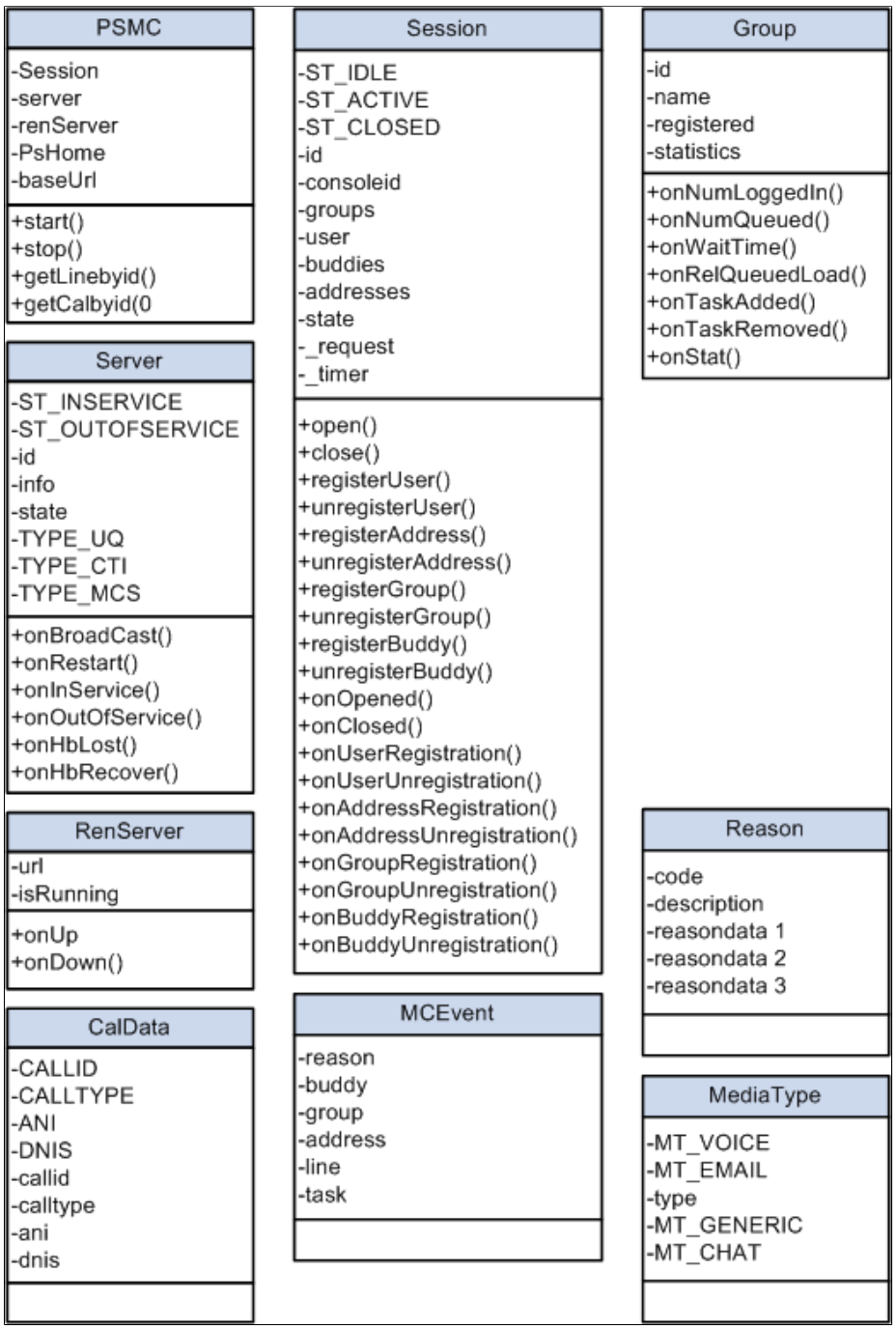
  Events can be the agent state change, such as event ready, not ready, and so on. Events can also be call events, such as incoming call, call released, call transferred, and so on.

Understanding JSMCAPI Classes

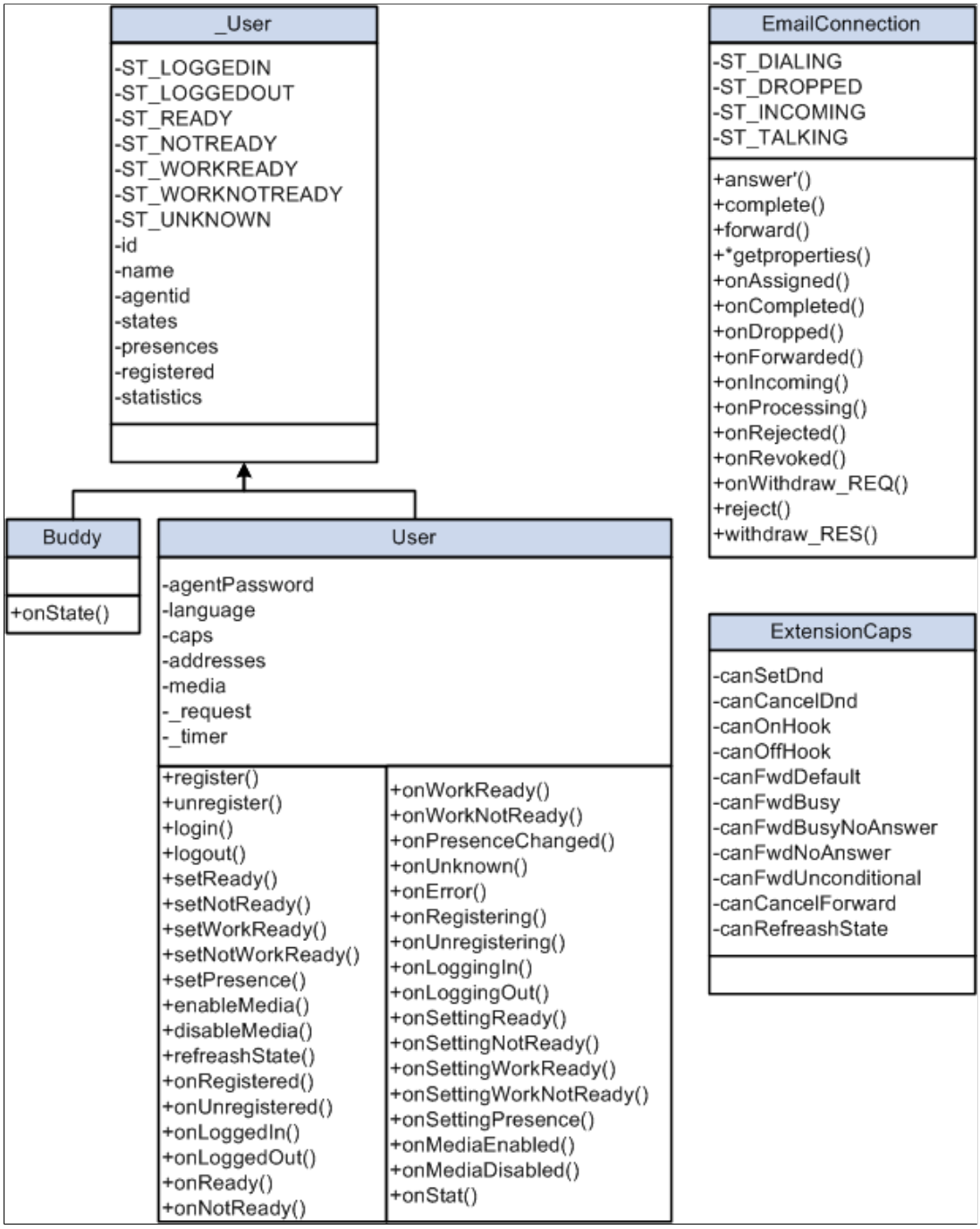
This sections gives an overview of all the JSMCAPI classes.

The following class diagrams explain all the classes, fields, and the methods.

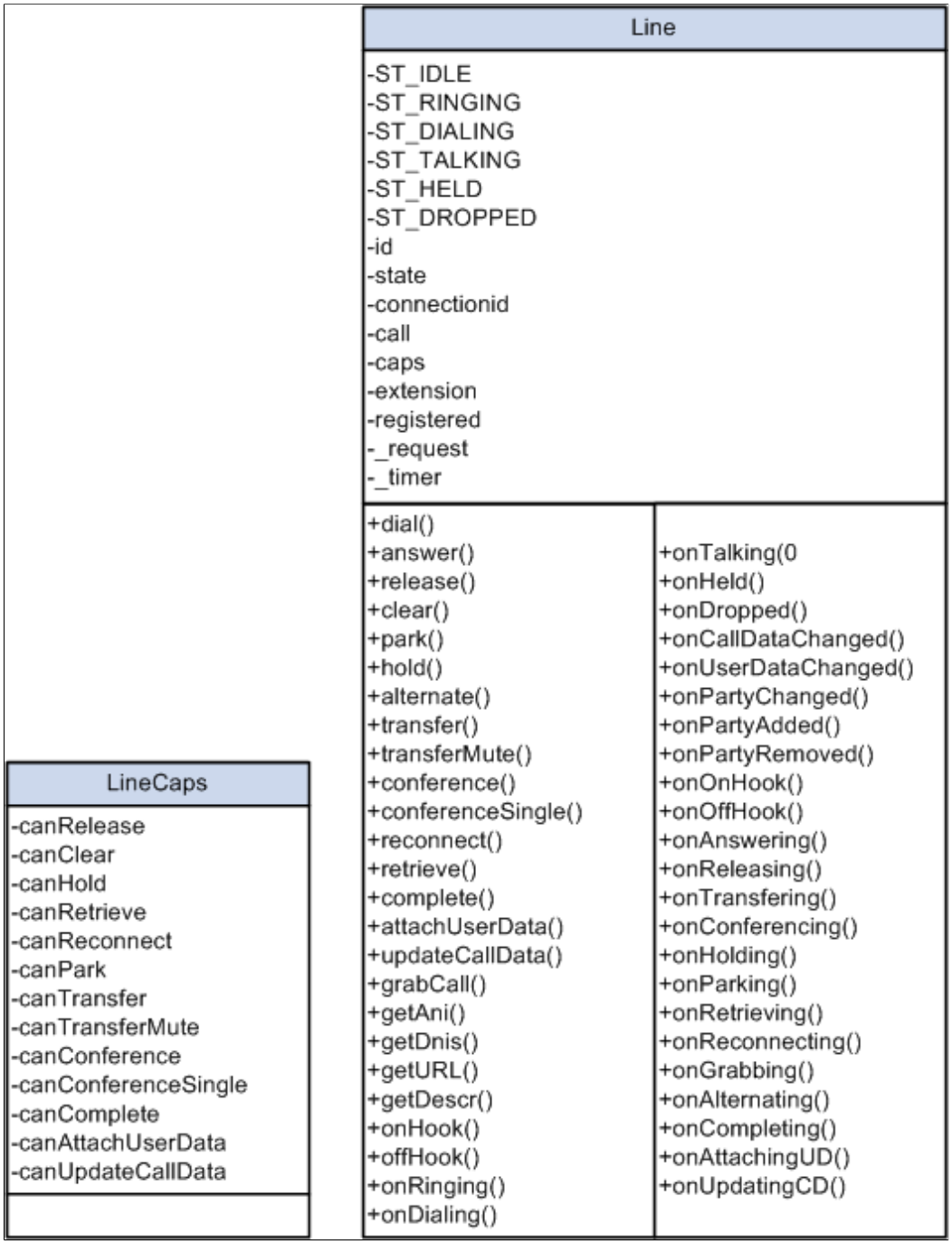
The following class diagram explains the classes, fields, and methods of JSMCAPI (diagram 1 of 4).



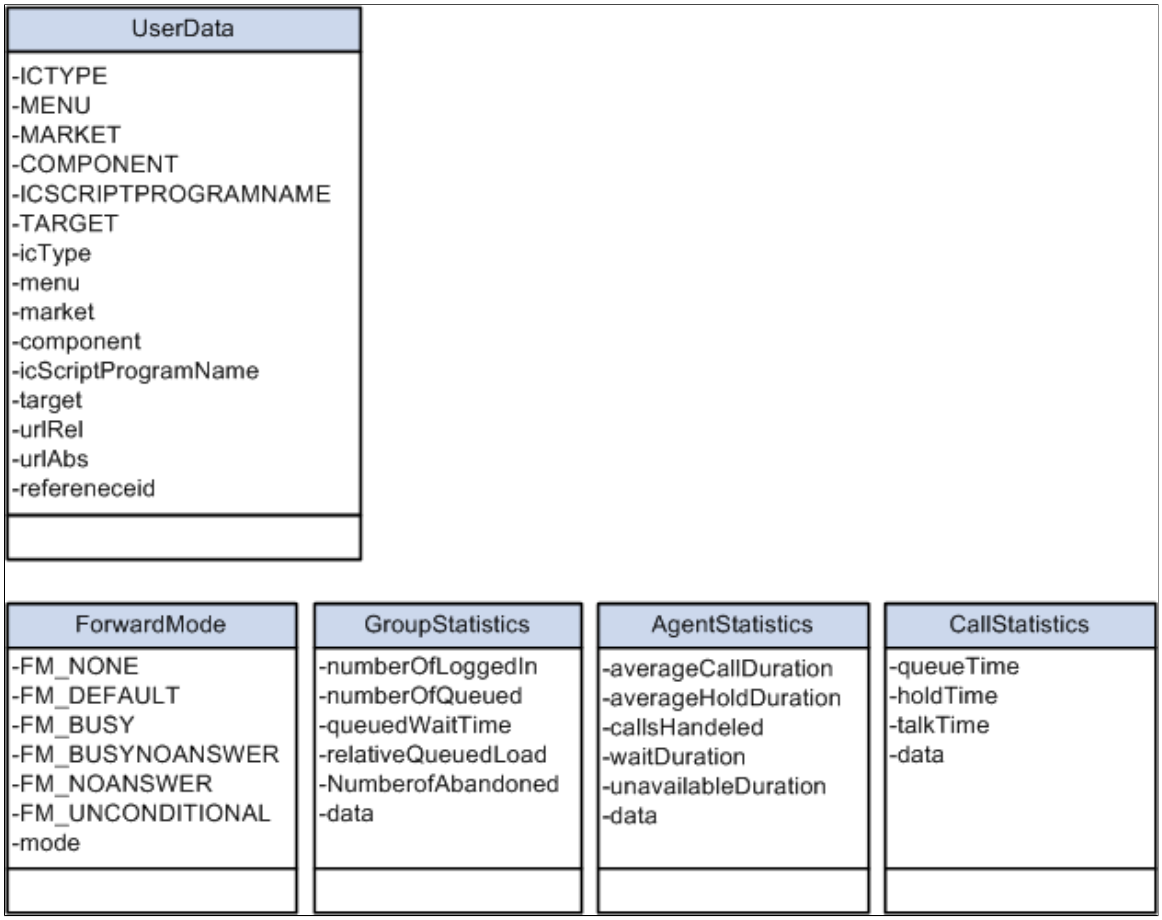
The following class diagram explains the classes, fields, and methods of JSMCAPI (diagram 2of 4).



The following class diagram explains the classes, fields, and methods of JSMCAPI (diagram 3 of 4).



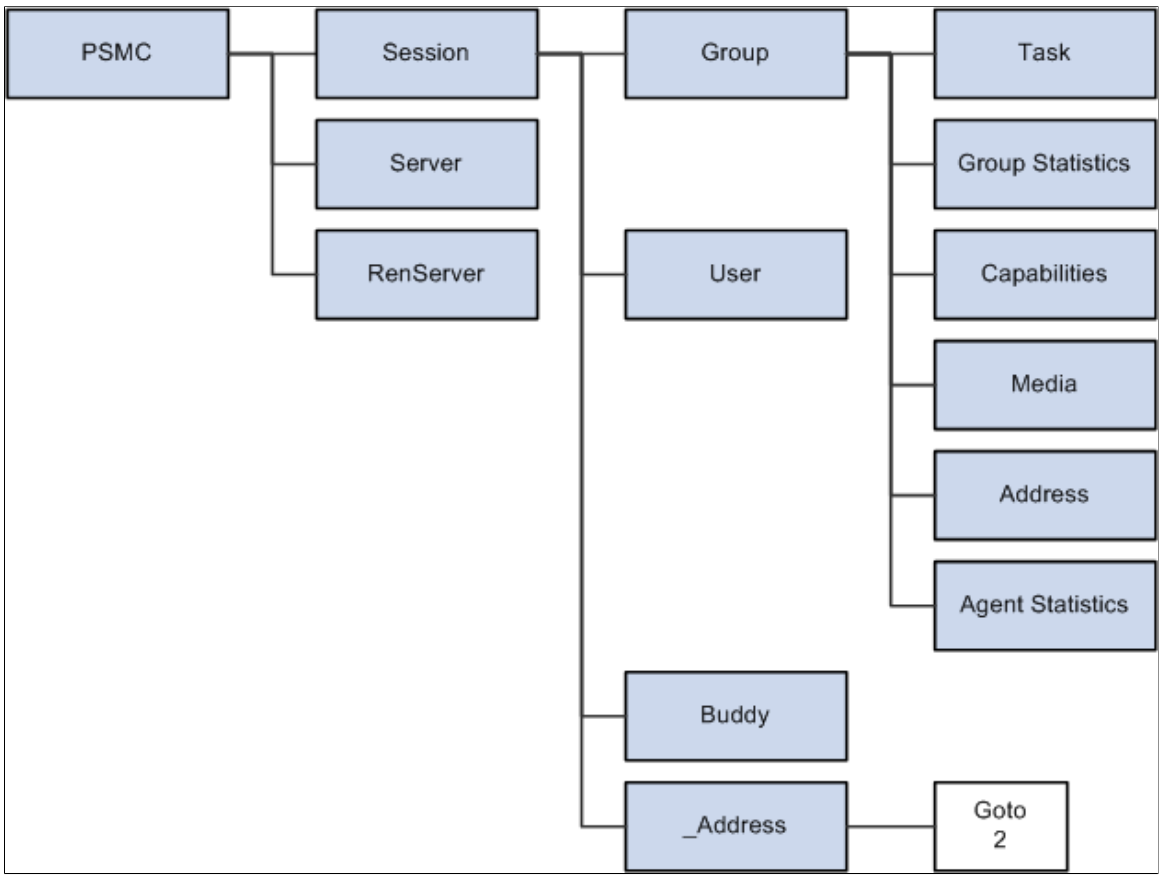
The following class diagram explains the classes, fields, and methods of JSMCAPI (diagram 4 of 4).



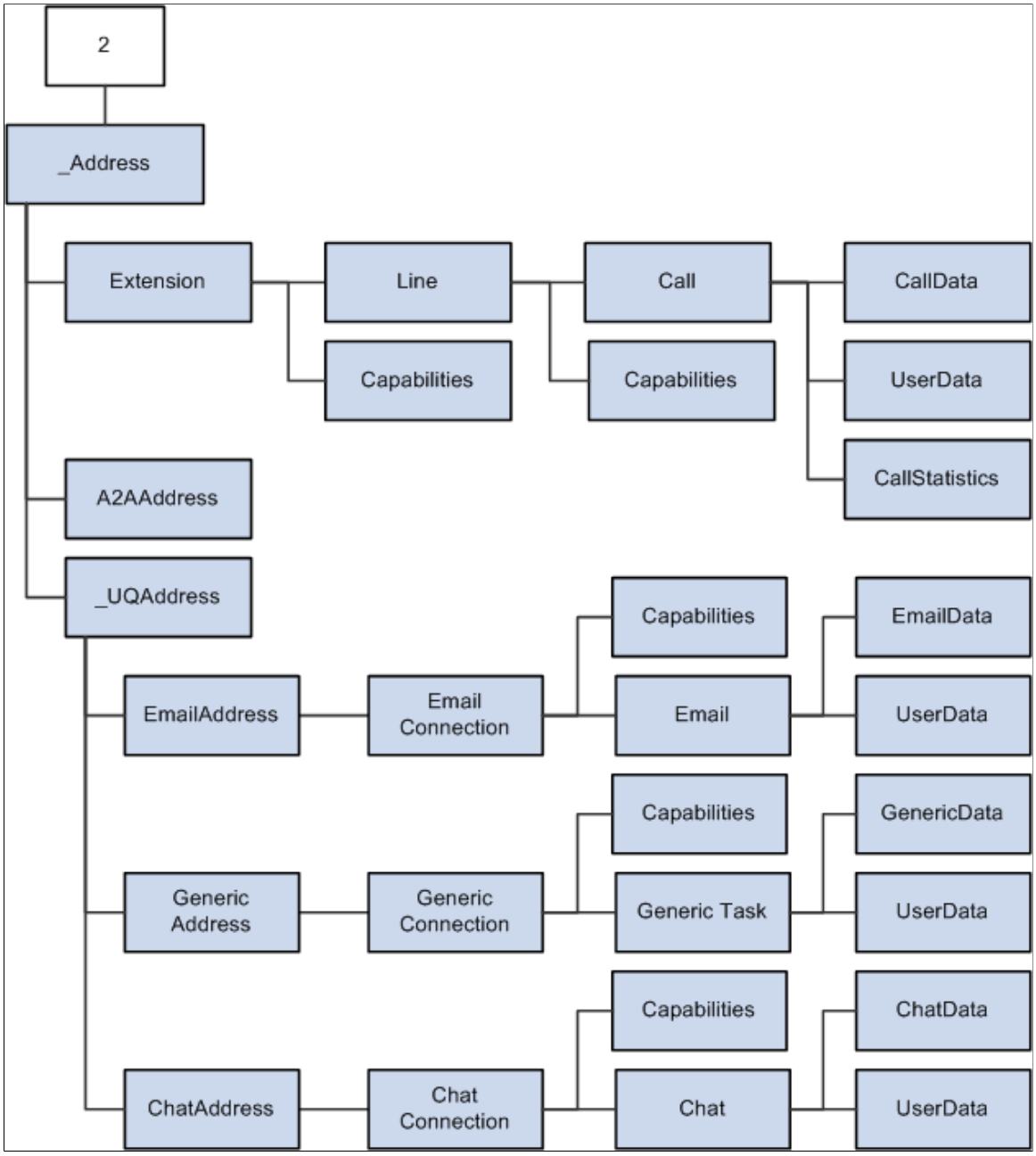
PSMC

PSMC is the base class that an application accesses to start an instance of JSMCAPI. The application can access all JSMCAPI functionality through this object. A session object is created from this class.

The following diagram explains the relationship of PSMC with all other classes (diagram 1 of 2).



The following diagram explains the relationship of PSMC with all other classes (diagram 2 of 2).



Related Links

PSMC Class Constructor

Server

The Server class refers to the routing server and can execute events for specific server states. Three types of servers are available: CTI, queue server, and MultiChannel server. The constants are CTI, UQ and MCS, respectively.

Related Links

Server Class Constructor

RENServer

The RENServer cluster is represented in the RENServer class. It provides the URL and the server status (active or shutdown).

Related Links

RenServer Class Constructor

Session

Sessions are set for users registering with the server. Addresses, buddies, and groups are registered for the user with the session. The connection between the server and JSMCAPI is a session. There is only one session object per PSMC object.

Related Links

Session Class Constructor

_Address

The _Address class identifies the user with a unique ID. The ID identifies the user to the routing system and is unique for each channel or media type.

The subclasses of _Address class are:

- Extension
- _UQAddress
- A2AchatAddress

Further delineation of address by media type is provided by:

- ChatAddress
- EmailAddress
- GenericAddress

ChatAddress, EmailAddress, and GenericAddress classes extend _UQAddress.

Related Links

Extension Class Constructor
_UQAddress Class Constructor
A2AChatAddress Class Constructor
ChatAddress Class Constructor
EmailAddress Class Constructor
GenericAddress Class Constructor

Line

Line class describes the line of the extension for a call task. This version supports one extension with two lines and two extensions with one line in each.

Related Links

Line Class Constructor

Connection

Tasks are routed to agents through a connection. Email, chat and generic have a dedicated connection class, like EmailConnection, ChatConnection, and GenericConnection. These connections provide task-specific manipulation functions.

Related Links

ChatConnection Class Constructor
EmailConnection Class Constructor

Group

The group class defines the group or queue information. Each session can have one or more group objects.

Related Links

Group Class Constructor

Task

This abstract base class defines a unit of work. The classes that extend task per media type are:

- Call
- Email
- Chat
- A2AChat
- GenericTask

Related Links

Call Class Constructor
Email Class Constructor
Chat Class Constructor
A2AChat Class Constructor
GenericTask Class Constructor

_User

_User is a base class. The subclasses are User and Buddy. These classes define the properties that pertain to the user such as the addresses, languages, or presence. _User is a virtual class and should not be instantiated.

Related Links

_User Class Constructor

MediaType

This class defines the media that an agent can handle.

Related Links

MediaType Class Constructor

Reason

The Reason class defines the message or error message that accompanies an event or request. Globalization of the messages is implemented. Furthermore, extra data can be passed in this object for providing a detailed message.

Related Links

Reason Class Constructor

Statistics

Statistics are provided by the routing server for agent, call, task, group, and user. The following classes describe the statistics for each component:

- AgentStatistics
- CallStatistics
- TaskStatistics
- GroupStatistics
- GroupStatistics1
- GroupStatistics2
- UserStatistics1
- UserStatistics2

Related Links

AgentStatistics Class Constructor
CallStatistics Class Constructor

TaskStatistics Class Constructor
GroupStatistics Constructor
GroupStatistics1 Class Constructor
GroupStatistics2 Class Constructor
UserStatistics1 Class Constructor
UserStatistics2 Class Constructor

Data

When a task is introduced to the system, data pertaining to the task is defined in the following classes:

- CallData
- EmailData
- ChatData
- GenericData

Application-specific data is provided for tasks via the AppData class. Similarly, user data is defined in UserData.

Related Links

CallData Class Constructor
EmailData Class Constructor
ChatData Class Constructor
GenericData Class Constructor

Globals

This class defines functions that are used universally.

Related Links

GLOBALS Class Fields

MCEvent

Events passed to the application handler are defined by MCEvent.

Related Links

MCEvent Class Constructor

Caps

Capabilities (Caps) define the ability or allowed actions for a component. The following classes define specific capabilities:

- ChatConnectionCaps

- EmailConnectionCaps
- GenericConnectionCaps
- ExtensionCaps
- LineCaps
- ChatConnectionCaps
- EmaiConnectionCaps
- GenericConnectionCaps
- UserCaps

Related Links

ChatConnectionCaps Class Constructor
EmailConnectionCaps Class Constructor
GenericConnectionCaps Class Constructor
ExtensionCaps Class Constructor
LineCaps Class Constructor

ForwardMode

ForwardMode describes various forward modes that can be used while setting the forwarding mode for an Address.

Related Links

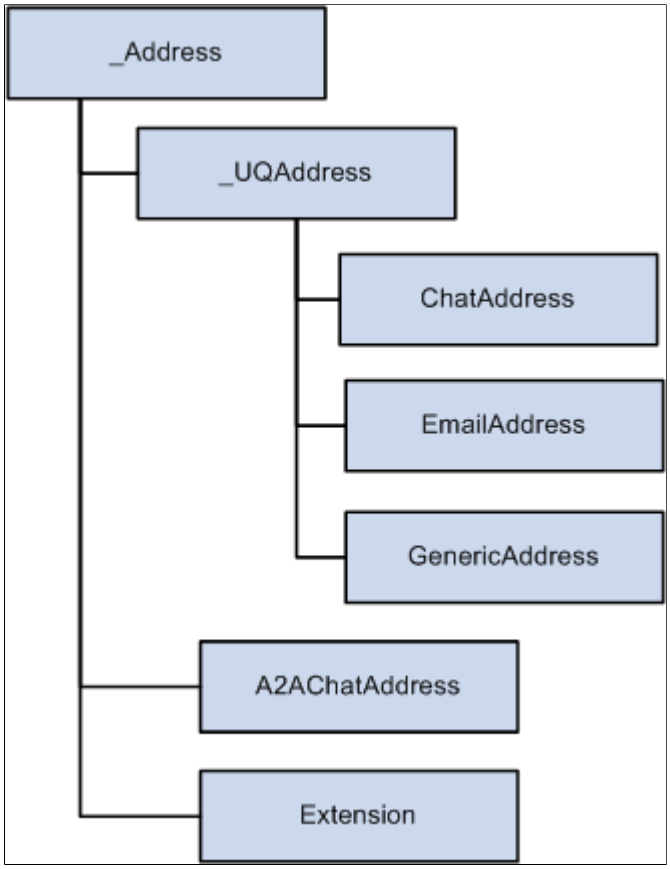
ForwardMode Class Constructor

_Address Class Hierarchy

The _Address class can be extended by other subclasses.

Do not instantiate _Address or _UQAddress class objects. Instead, use the child classes.

The following flow chart shows the different subclasses of _Address class and how they interrelate.



Related Links

_Address Class Constructor
_UQAddress Class Constructor
A2AChatAddress Class Constructor
Extension Class Constructor
ChatAddress Class Constructor
EmailAddress Class Constructor
GenericAddress Class Constructor

_Address Class Constructor

The following is the _Address class constructor.

_Address

Syntax

```
_Address()
```


Description

Creates an _Address object that describes the address.

Parameters

None.

Returns

An _Address object.

_Address Class Fields

This section discusses the JSMCAPI _Address class fields, which are described in alphabetical order.

caps

Description

The capacities of the address.

Type: object.

id

Description

The ID of the address.

Type: string.

_Address Class Callback Event Method

The following is the callback event method used with a JSMCAPI _Address object.

onError

Syntax

onError(*event*)

Description

Fires on the event of an address error.

_UQAddress Class Constructor

The following is the UQAddress class constructor.

_UQAddress

Syntax

```
_UQAddress()
```

Description

Creates a _UQAddress object, which is the base for the various addresses associated with the universal queue server.

Parameters

None.

Returns

A universal queue address object.

_UQAddress Class Fields

The _UQAddress class inherits the following fields from the _Address class:

- caps
- id

See _Address Class Fields.

The following are the _UQAddress class fields.

Tasks

Description

The associative array of the tasks on the queue managed by the address.

_UQAddress Methods

The following are the _UQAddress methods.

acceptTask

Syntax

acceptTask(*task, reason*)

Description

Sends a request to the universal queue server to signal that the client has accepted the task.

Parameters

Parameter	*Description*
task	The task ID.
reason	The associated reason code.

Returns

Request number.

dequeueTask

Syntax

dequeueTask(*task*)

Description

Sends a request to the universal queue server to remove the task from its queue.

Parameters

Parameter	*Description*
task	The task ID.

Returns

Request number.

_UQAddress Class Callback Event Methods

The _UQAddress class inherits the onError callback event method from the _Address class.

See _Address Class Callback Event Method.

The following are the callback event methods used with a JSMCAPI universal queue address object. The callback event methods are described in alphabetical order.

onAccepted

Syntax

onAccepted(*event*)

Description

Fires when the task is accepted.

onAcceptingTask

Syntax

onAcceptingTask(*event*)

Description

Fires as the task is being accepted.

onDequeueingTask

Syntax

onDequeueingTask(*event*)

Description

Fires as the task is being dequeued.

onNotify

Syntax

onNotify(*event*)

Description

Fires on task notification.

onTaskAdded

Syntax

onTaskAdded(*event*)

Description

Fires when the task is added to the addressed queue.

onTaskRemoved

Syntax

onTaskRemoved(*event*)

Description

Fired when the task is removed

onUnassigned

Syntax

onUnassigned(*event*)

Description

Fired when the task is unassigned.

_User Class Constructor

The following is the _User class constructor.

_User

Syntax

_User()

Description

Describes the base class of user/agent.

Parameters

None.

Returns

A _User object.

_User Class Fields

The following are the fields associated with the JSMCAPI _User class. These fields are discussed in alphabetical order.

agentID

Description

The agent's agent ID.

Type: string.

caps

Description

The agent capabilities on each group.

Type: associative array.

id

Description

The agent's PeopleSoft user ID.

Type: string.

name

Description

The agent's user name.

Type: string.

presences

Description

The agent's presence on each group.

Type: associative array.

states

Description

Agent state on each group.

Type: associative array, including the constants beginning with ST_*.

These constants must be accessed from an instantiated object.

ST_LOGGEDIN

Description

The agent is logged in.

ST_LOGGEDOUT

Description

The agent is logged out.

ST_NOTREADY

Description

The agent is not ready.

ST_READY

Description

The agent is ready.

ST_UNKNOWN

Description

The agent's state is unknown.

ST_WORKNOTREADY

Description

The agent is in the work not ready state.

ST_WORKREADY

Description

The agent is in the work ready state.

statistics

Description

Agent statistics for CTI.

Type: AgentStatistics object.

Related Links

AgentStatistics Class Constructor

statistics1

Description

Agent statistics for the universal queue server.

Type: UserStatistics1 object.

Related Links

UserStatistics1 Class Constructor

statistics2

Description

Agent statistics for the universal queue server.

Type: UserStatistcs2 object.

Related Links

UserStatistics2 Class Constructor

A2AChat Class Constructor

The following is the A2AChat class constructor.

A2AChat

Syntax

A2AChat(*event*,*address*,*chatType*)

Description

Creates an agent-to-agent chat object.

The chat is related to the A2AChatAddress. It does not extend Task as it is not generated or tracked by the universal queue server. Construction occurs inside the A2AChatAddress.

Parameters

Parameter	***Description***
event	Enter the event associated with the agent-to-agent chat.
address	Enter the address from A2AChatAddress.
chatType	Enter the chat type, for example consult or answer.

Returns

An agent-to-agent chat object.

A2AChat Class Fields

The following are the A2AChat fields.

address

Description

The address containing the agent-to-agent chat.

Type: A2AChatAddress object.

Related Links

A2AChatAddress Class Constructor

agentID

Description

The associated agent's ID.

Type: string.

agentName

Description

The associated agent's name.

Type: string.

appData

Description

The application data that is provided to the client with the notification event.

Type: AppData object.

Related Links

AppData Class Constructor

chatType

Description

The type of chat, either A2AChat.TYPE_CONSULT or A2AChat.TYPE_ANSWER.

Type: string, with one of the following constants:

Value	*Description*
TYPE_ANSWER	The A2AChat type answer. This type is generated when another agent wants this agent to answer an A2AChat.
TYPE_CONSULT	The A2AChat type consult. This type is generated when this agent wants to consult another agent.

Example

The A2AChat.TYPE_* can only be accessed as in the following example:

```
var value = A2AChat.TYPE_CONSULT;
```

Related Links

GLOBALS Class Fields

customerName

Description

The name of the customer in the ongoing chat.

Type: object.

id

Description

The ID of this chat.

Type: string.

isConference

Description

A flag to check whether it is conference or not.

Type: object.

jr

Description

JournalRoute

Type: string.

question

Description

Question that is asked when customer initiates chat.

Type: object.

subject

Description

Subject of the chat. Customer will provide this subject when he initiates chat.

Type: object.

type

Description

The task type, chat.

Type: string.

uniqueueId

Description

The unique ID.

Type: string.

A2AChat Class Method

The following is the A2AChat class method.

getURL

Syntax

getURL(*defaultURL*)

Description

Returns the URL for the given task.

Parameters

Parameter	*Description*
defaultURL	If not null, this value will override the generated base URL.

Returns

The URL associated with this task.

A2AChatAddress Class Constructor

The following is the A2AChatAddress constructor.

A2AChatAddress

Syntax

```
A2AChatAddress()
```

Description

Agent-to-agent chat address. Handles the creation of agent-to-agent chats for incoming and outgoing chat communication.

Parameters

None.

Returns

An A2AChatAddress object.

A2AChatAddress Class Fields

The A2AChatAddress class inherits the following fields from the _Address class:

- id
- caps

See _Address Class Fields.

The following are the fields associated with the A2AChatAddress class.

id

Description

The address ID.

Type: string.

tasks

Description

The array of A2AChats associated with this address.

Type: associative array.

Related Links

GLOBALS Class Fields

A2AChatAddress Class Method

The following is the A2AChatAddress class method.

initiateChat

Syntax

initiateChat(*agentId*)

Description

Initializes the A2AChatAddress by creating an A2AChat task.

Parameters

Parameter	***Description***
agentId	The ID of the agent with whom you want to chat.

Returns

Request number.

A2AChatAddress Class Callback Event Methods

The A2AChatAddress class inherits the onError callback event method from the _Address class.

See _Address Class Callback Event Method.

The following are the A2AChatAddress class callback methods:

onChatEnded

Syntax

onChatEnded(*event*)

Description

This event gets called when an agent to agent chat is closed.

onInitiatingChat

Syntax

onInitiatingChat(*event*)

Description

This event gets called when chat is getting initiated.

onNotify

Syntax

onNotify(*event*)

Description

This event gets called when there is a new A2Achat notification.

AgentStatistics Class Constructor

The following is the AgentStatistics class constructor.

AgentStatistics

Syntax

```
AgentStatistics()
```

Description

The agent statistics information.

Parameters

None.

Returns

An AgentStatistics object.

AgentStatistics Class Fields

The following are the AgentStatistics class fields.

averageCallDuration

Description

The average call duration, in seconds, for an agent.

Type: number

averageHoldDuration

Description

The average hold duration, in seconds, for an agent.

Type: number.

callsHandled

Description

The total number of calls handled by an agent.

Type: number.

data

Description

An associative array of key-value pairs that includes all agent statistics.

Type: associative array.

percentIdleTime

Description

The percentage of time for which the agent is idle.

Type: number

percentTimeAvailable

Description

The percentage of time for which the agent is in ready state.

Type: number

percentTimeInCurrentState

Description

The percentage of time in current state.

Type: number

percentTimeUnavailable

Description

The percentage of time for which the agent is in not ready state.

Type: number

timeCurrentLogin

Description

The time since login for the agent.

Type: number.

timeWorking

Description

The time spent working on tasks.

Type: number.

totalTaskAcceptedLogin

Description

The number of tasks accepted since login time

Type: number

totalTaskDoneLogin

Description

Number of tasks done by the agent since login time i.e. current login.

Type: number.

totalTaskUnassignedLogin

Description

Number of tasks unassigned since login time

Type: number.

unavailableDuration

Description

The total time for which an agent is unavailable.

Type: string.

waitDuration

Description

The total time an agent has to wait for a call.

Type: string.

AppData Class Constructor

The following is the AppData class constructor.

AppData

Syntax

```
AppData()
```

Description

The AppData object describes the key-value pairs of the data received on the agent-to-agent chat event.

Parameters

None.

Returns

Type: AppData object.

AppData Class Fields

The following are the AppData class fields.

data

Description

An associative array of the data in name value pairs.

Type: associative array.

groupId

Description

The group ID passed in by the notify event.

Type: string.

jr

Description

The JournalRoute.

Type: string.

question

Description

The initial question of the task.

Type: string.

strData

Description

The raw application data as a string.

Type: string.

subject

Description

The subject of the task.

Type: string.

uniqueId

Description

The unique ID.

Type: string.

url

Description

The URL passed in by the AppData.

Type: string with the following constant:

Value	*Description*
URL	The associated URL.

userId

Description

The agent's user ID.

Type: string.

username

Description

The agent's user name.

Type: string.

wizUrl

Description

The wizard URL that is used for popping up new task window.

Type: string.

AppData Class Method

The following is the AppData class method.

addKeyValue

Syntax

addKeyValue(*key, value*)

Description

Add the key-value pair to the AppData object.

Parameters

Parameter	*Description*
key	The key.
value	The value.

Returns

None.

Buddy Class Constructor

The Buddy class extends the _User class.

See _User Class Constructor.

The following is the Buddy class constructor.

Buddy

Syntax

```
Buddy()
```

Description

Describes the buddy. An agent can register buddies and is notified of state changes.

Parameters

None.

Returns

A Buddy object.

Buddy Class Fields

The Buddy class inherits the following fields from the _User class:

- agentID
- caps

- id
- name
- presences
- states
- statistics
- statistics1
- statistics2

See _User Class Constructor.

Buddy Class Callback Event Methods

The following are the Buddy class callback event methods.

onStat1

Syntax

onStat1(*event*)

Description

Fires when statistics1 is received.

onStat2

Syntax

onStat2(*event*)

Description

Fires when statistics2 is received.

onState

Syntax

onState(*event*)

Description

Fires when the state event is received.

Call Class Constructor

The Call class extends the Task class.

See Task Class Constructor.

The following is the Call class constructor.

Call

Syntax

Call(*strCall*)

Description

The Call object describes the call task information associated with the line.

Parameters

Parameter	***Description***
strCall	Enter the call to use with this object.

Returns

A Call object.

Call Class Fields

The Call class inherits the following fields from the Task class:

- caseid
- cost
- customerid
- group
- id
- onStat
- priority
- type

- urlAbs
- urlRel

See Task Class Fields.

The following are additional Call class fields.

line

Description

The line object that is associated with this call.

Type: line object.

Related Links

Line Class Constructor

statistics

Description

Call statistics object associated with the call.

Type: CallStatistics object.

Related Links

CallStatistics Class Constructor

CallData Class Constructor

The following is the CallData class constructor.

CallData

Syntax

```
CallData()
```

Description

The CallData object describes the key-value pairs of the call data with the call object.

Parameters

None.

Returns

A CallData object.

CallData Class Fields

The following are the CallData class fields:

ani

Description

The ANI caller id.

Type: string.

callId

Description

The call ID.

Type: string.

callType

Description

The call type.

Type: string.

data

Description

The key-value pairs of the call data.

Type: string.

dnis

Description

The DNIS callee ID.

Type: string.

CallData Class Method

The following is the CallData class method.

addKeyValue

Syntax

addKeyValue(*key*), *value*

Description

Add key value to the CallData object.

Parameters

Parameter	*Description*
key	The key.
value	The value.

Returns

None.

CallStatistics Class Constructor

The following is the CallStatistics class constructor.

CallStatistics

Syntax

CallStatistics()

Description

The call statistics information.

Parameters

None.

Returns

A CallStatistics object.

CallStatistics Class Fields

The following are the CallStatistics class fields.

data

Description

An associative array of key-value pairs that includes all statistics.

Type: associative array.

holdTime

Description

Duration that the call is on hold.

Type: string.

queueTime

Description

Duration that this call has been in the queue.

Type: string.

talkTime

Description

Duration that the call is established.

Type: string.

Chat Class Constructor

The Chat class extends the Task class.

See Task Class Constructor.

The following is the Chat class constructor.

Chat

Syntax

Chat(*event*, *address*)

Description

The Chat object describes the task information for tasks associated with the ChatAddress.

Parameters

Parameter	*Description*
event	The onNotify event.
address	The chat address.

Returns

A Chat object.

Chat Class Fields

The Chat class inherits the following fields from the Task class:

- caseid
- cost
- customerid
- group
- id
- onStat
- priority
- type

  The task type for chat that is Task.TYPE_CHAT.
- urlAbs

- urlRel

See Task Class Fields.

The following are the additional Chat class fields.

address

Description

The address containing the agent-to-agent chat.

Type: A2AChatAddress object.

Related Links

A2AChatAddress Class Constructor

agentId

Description

The agent's ID.

Type: string.

appData

Description

The application data that is provided to the client with the notification event.

Type: AppData object.

Related Links

AppData Class Constructor

chatconnection

Description

The chat connection object associated with this call.

Type: line object

chatType

Description

Type of chat, for example consulting chat or answering chat

Type: string

customerName

Description

The customer username.

Type: string.

groupId

Description

The group ID.

Type: string.

question

Description

The question for this chat.

Type: string.

subject

Description

The subject of the chat.

Type: string.

statistics

Description

The chat statistics object that associated with this chat.

Type: string.

userData

Description

The user data associated with the chat.

Type: object.

Chat Class Method

The Chat class extends the Task class.

See Task Class Constructor.

The following is the Chat class method.

gettpUrl

Syntax

gettpUrl(*defaultUrl*)

Description

Returns the URL for the given task for a third-party routing server.

Parameters

Parameter	***Description***
defaultUrl	If not null, this value will override the generated base URL.

Returns

Returns the URL associated with this task.

getUrl

Syntax

getUrl(*defaultUrl*)

Description

Returns the URL for the given task.

Parameters

Parameter	*Description*
defaultUrl	The default URL for this chat. If not null this value will override the generated base URL.

Returns

Returns the URL for the chat.

ChatAddress Class Constructor

The ChatAddress class extends the _UQAddress class.

See _UQAddress Class Constructor.

The following is the ChatAddress class constructor.

ChatAddress

Syntax

```
ChatAddress()
```

Description

Handles the creation of customer chat tasks.

Parameters

None.

Returns

A ChatAddress object.

ChatAddress Class Fields

The ChatAddress class inherits the following fields from the _Address class:

- id

  The address's ID is equal to Task.TYPE_CHAT.

- caps

See _Address Class Fields.

The ChatAddress class inherits the following field from the _UQAddress class:

tasks

See _UQAddress Class Fields.

The following are ChatAddress class fields.

chatconnections

Description

List of ChatConnection objects that are associated with chat address.

Type: chatconnection object.

ChatAddress Class Methods

The ChatAddress class inherits the following methods from the _UQAddress class:

- acceptTask
- dequeueTask

See _UQAddress Methods.

The following are ChatAddress class methods.

chat

Syntax

chat(*agentId*,*buddyId*,*reason*)

Description

Sends a request to initiate a new chat.

Parameters

Parameter	***Description***
agentId	The source agent ID.

Parameter	*Description*
buddyId	The Id of the buddy.
reason	The reason code for chat request.

Returns

Request number.

getChatconnectionByConnectionId

Syntax

getChatconnectionByConnectionId(*connectionId*)

Description

Get the chatconnection object with the chatconnection Id.

Parameters

Parameter	*Description*
connectionId	The connection id associated with a chat.

Returns

Line object, null if there is no connection with that chatconnection id.

getChatconnectionindexByConnectionId

Syntax

getChatconnectionindexByConnectionId
(*connectionId*)

Description

Get the chatconnection index with the chatconnection Id.

Parameters

Parameter	*Description*
connectionId	The connection id associated with a chat.

Returns

index, −1 if there is no connection with that chatconnection id.

getFreeChatconnection

Syntax

```
getFreeChatconnection()
```

Description

Get the free chatconnection object. A connection is free if there is no activity on this line.

Returns

Returns chatconnection object, null if there is no free line.

getFreeChatconnectionIndex

Syntax

```
getFreeChatconnectionIndex()
```

Description

Get the free chatconnection index. A connection is free if there is no activity on this line.

Returns

Returns chatconnection index, null if there is no free line.

ChatAddress Callback Event Methods

The ChatAddress class inherits the onError callback event method from the _Address class.

See _Address Class Callback Event Method.

The ChatAddress class inherits the following callback event methods from the _UQAddress class:

- onAccepted
- onAcceptingTask
- onDequeueingTask
- onNotify
- onTaskAdded
- onTaskRemoved
- onUnassigned

See _UQAddress Class Fields.

The following are ChatAddress callback event methods:

onCapabilitiesChanged

Syntax

onCapabilitiesChanged*event*

Description

Fires when ChatAddress capabilities change

ChatConnection Class Constructor

The following is the chatconnection class constructor.

ChatConnection

Syntax

ChatConnection()

Description

Chat connection object.

Parameters

None

Returns

Returns a chatconnection object.

ChatConnection Class Fields

The following are chatconnection class fields:

caps

Description

The capabilities of the ChatConnection.

chat

Description

Chat task associated with the connection.

Type: string

connectionId

Description

The connection Id.

Type: string

id

Description

The constant representing ChatConnection Id.

Type: string

state

Description

The connection state.

Type: string

Value	***Description***
ST_DIALING	Dialing state.

Value	*Description*
ST_DROPPED	Dropped state.
ST_IDLE	Idle state.
ST_INCOMING	Incoming state.
ST_TALKING	Talking state.
ST_WRAPUP	Wrap up state.

ChatConnection Class Methods

The following are chatconnection class methods:

answer

Syntax

answer(*agentId*,*reason*)

Description

Answer a chat request.

Parameters

Parameter	*Description*
agent Id	The Id of the agent answering the call.
reason	The reason code for the answer request.

Returns

Request number.

attachUserData

Syntax

attachUserData(*userData*,*reason*)

Description

Attach user data for the chat connection.

Parameters

Parameter	*Description*
userData	User data of the chat connection.
reason	The reason code for the answer request.

Returns

Request number.

conference

Syntax

conference(*agentId*,*reason*)

Description

Conference a chat with another agent.

Parameters

Parameter	*Description*
agent Id	The Id of the agent invited to the conference.
reason	The reason code for the conference request.

Returns

Request number.

forward

Syntax

forward(*fromagentId*,*toagentId*, *qid*, *userdata*, *chatdata*, *reason*)

Description

Forward chat to another agent or another queue.

Parameters

Parameter	*Description*
fromagentid	The agent id from whom task is forwarded.
toagentid	The target agentid to whom task is forwarded.
qid	The target queue to forward the task.
userdata	The user data.
chatdata	The chat data.
Reason	The reason code to forward the task.

Returns

Request number.

gethistory

Syntax

gethistory(*agentId*,*noofLines*, *reason*)

Description

Request chat history by specifying number of lines of chat history needed.

Parameters

Parameter	*Description*
agent Id	The source agent ID.
noofLines	Number of lines of history required.
reason	The reason code for the history request.

Returns

Request number.

getUrl

Syntax

getUrl(*defaultURL*)

Description

Get the URL for screen popup.

Parameters

Parameter	*Description*
defaultUrl	The default popup URL.

Returns

Returns a URL string.

message

Syntax

message(*agentId*,*message*, *reason*)

Description

Send a chat message while chatting.

Parameters

Parameter	*Description*
agent Id	The source agent ID.
message	The message to send to other party.
reason	The reason code for the message request.

Returns

Request number.

pushURL

Syntax

pushURL(*URL*)

Description

Request to push a URL to the customer.

Parameters

Parameter	*Description*
URL	URL to push.

Returns

Request number.

reject

Syntax

reject(*agentId*,*reason*)

Description

Reject a chat.

Parameters

Parameter	*Description*
agent Id	The source agent ID.
reason	The reason code for reject.

Returns

Request number.

release

Syntax

release(*agentId*,*reason*)

Description

Release a chat.

Parameters

Parameter	*Description*
agent Id	The source agent ID.
reason	The reason code for release request.

Returns

Request number.

typing

Syntax

typing(*agentId*,*reason*)

Description

Indicates typing in a chat conversation by one party.

Parameters

Parameter	*Description*
agent Id	The source agent ID.
reason	The reason code for typing.

Returns

Request number.

wrapup

Syntax

wrapup(*agentid*,*message*,*tasknum*,*reason*)

Description

Stores chat wrap up comments.

Parameters

Parameter	*Description*
agentid	The agent id who enters wrap up comments.
message	Chat message.
tasknum	Task number.
reason	The reason code.

Returns

Request number.

ChatConnection Class Callback Event Methods

The following are ChatConnection class callback event methods.

onAccepted

Syntax

onAccepted(*event*)

Description

Fires when chat is accepted by a agent.

onAnswering

Syntax

onAnswering(*event*)

Description

Fires when answering the chat.

onCapabilitiesChanged

Syntax

onCapabilitiesChanged(*event*)

Description

Fires when chat capabilities change.

onChatdataChanged

Syntax

onChatdataChanged(*event*)

Description

Fires when there is a change in chat data.

onConferencing

Syntax

onConferencing(*event*)

Description

Fires when conferencing the chat.

onDialing

Syntax

onDialing(*event*)

Description

Fires when chat is in the process of connecting.

onDropped

Syntax

onDropped(*event*)

Description

Fires when chat is dropped.

onError

Syntax

onError(*event*)

Description

Fires when a there is a chat connection error.

onForwarded

Syntax

onForwarded(*event*)

Description

Fires when chat is forwarded.

onForwardError

Syntax

onForwardError(*event*)

Description

Fires when there is error in forwarding the task.

onForwarding

Syntax

onForwarding(*event*)

Description

Fires when forwarding the chat.

onHistory

Syntax

onHistory(*event*)

Description

Fires when agent receives chat history.

onIncomingChat

Syntax

onIncomingChat(*event*)

Description

Fires when ChatConnection state is changed to INCOMING.

onMessage

Syntax

onMessage(*event*)

Description

Fires when there is incoming message.

onPartyAdded

Syntax

onPartyAdded(*event*)

Description

Fires when a new chat party is added.

onPartyChanged

Syntax

onPartyChanged(*event*)

Description

Fires when the chat party changes.

onPartyRemoved

Syntax

onPartyRemoved(*event*)

Description

Fires when a chat party is removed.

onProperties

Syntax

onProperties(*event*)

Description

Fires when a agent receives chat properties.

onPushURL

Syntax

onPushURL(*event*)

Description

Fires when a routing system push URL.

onRejected

Syntax

onRejected(*event*)

Description

Fires when a chat is rejected.

onReleased

Syntax

onReleased(*event*)

Description

Fires when a chat is released.

onReleasing

Syntax

onReleasing(*event*)

Description

Fires when conferencing the chat.

onRevoked

Syntax

onRevoked(*event*)

Description

Fires when a chat is revoked.

onTalking

Syntax

onTalking(*event*)

Description

Fires when an agent/customer is in the state of talking.

onTyping

Syntax

onTyping(*event*)

Description

Fires when typing in a chat conversation.

onUserdataChanged

Syntax

onUserdataChanged(*event*)

Description

Fires when there is a change in user data.

ChatConnectionCaps Class Constructor

The following is the chatconnectioncaps class constructor.

ChatConnectionCaps

Syntax

ChatConnectionCaps(*strCaps*)

Description

Describes the ChatConnection capabilities.

Parameters

Parameter	*Description*
strCaps	Chat connection capability.

Returns

Returns a chatconnection object.

ChatConnectionCaps Class Fields

The following are the ChatConnectionCaps class fields.

canAnswer

Description

Answer capability.

Type: boolean

canConference

Description

Conference capability.

Type: boolean

canConferenceSingle

Description

Conference single capability.

Type: boolean

canForward

Description

Forward capability.

Type: boolean

canIndicateTyping

Description

Indicate Typing capability.

Type: boolean

canPushURL

Description

PushURL capability.

Type: boolean

canReject

Description

Reject capability.

Type: boolean

canSendMessage

Description

SendMessage capability.

Type: boolean

ChatData Class Constructor

The following is ChatData class constructor.

ChatData

Syntax

```
ChatData()
```

Description

The ChatData object describes the key-value pairs of the chat data with the chat object.

ChatData Class Fields

The following is ChatData class field.

data

Description

Key value pairs that includes data.

Type: object.

ChatData Class Methods

The following is ChatData class method.

addKeyValue

Syntax

```
addKeyValue(key, value )
```

Description

Add key value to the chatdata object.

Parameters

Parameter	*Description*
key	The variable name.
value	The value of the variable.

Email Class Constructor

The Email class extends the Task class.

See Task Class Constructor.

The following is the email class constructor.

Email

Syntax

Email(*event*)

Description

The Email object describes the task information for tasks associated with the EmailAddress.

Parameters

Parameter	***Description***
event	The notification event.

Returns

Returns an Email object.

Email Class Fields

The Chat class inherits the following fields from Task class:

- caseid
- cost
- customerid
- group
- id
- onStat
- priority
- type

The task type is equal to Task.TYPE_EMAIL.

- urlAbs
- urlRel

See Task Class Fields.

The following are the additional Email class fields.

address

Description

The address containing the A2AChat.

agentId

Description

The agent's ID.

appData

Description

The application data that is provided to the client with the notification event.

Type: AppData object.

Related Links

Task Class Fields

customerName

Description

The customer username.

Type: string.

emailconnection

Description

The emailconnection object that associated with this email.

Type: line object.

emailId

Description

The unique email id generated from an Enqueue. It identifies the email stored in the database ie. it serves as a key to the data stored in the database.

Type: string.

groupId

Description

The group id.

Type: string.

question

Description

The question for this email.

Type: string.

statistics

Description

The statistics object that is associated with this email.

Type: object.

subject

Description

The subject of this email.

Type: string.

userData

Description

The user data that is associated with this email.

Type: object.

Email Class Method

The following are the Email class methods:

gettpUrl

Syntax

gettpUrl(*defaultUrl*)

Description

Returns the URL for the given task for third-party routing.

Parameters

Parameter	***Description***
defaultUrl	If not null, this value will override the generated base URL.

Returns

The URL associated with this task.

getUrl

Syntax

getURL(*defaultUrl*)

Description

Returns the URL for the given task.

Parameters

Parameter	*Description*
defaultUrl	If not null, this value will override the generated base URL.

Returns

The URL associated with this task.

EmailAddress Class Constructor

The EmailAddress class extends the _UQAddress class.

See _UQAddress Class Constructor.

The following is the EmailAddress class constructor.

EmailAddress

Syntax

```
EmailAddress()
```

Description

Handles the creation of email tasks.

Parameters

None.

Returns

An EmailAddress object.

EmailAddress Class Fields

The EmailAddress class inherits the following fields from the _Address class:

- id

  The address's ID is equal to Task.TYPE_EMAIL.

- caps

See _Address Class Fields.

The EmailAddress class inherits the tasks field from the _UQAddress class.

See _UQAddress Class Fields.

The following are EmailAddress class fields.

agent

Description

The agent id of this email address.

emailconnections

Description

Describes the EmailConnection object.

EmailAddress Class Methods

The EmailAddress class inherits the following fields from the _UQAddress class:

- acceptTask
- dequeueTask

See _UQAddress Methods.

getEmailconnectionByConnectionId

Syntax

`getEmailconnectionByConnectionId`(*`connectionId`*)

Description

Get the emailconnection object with the emailconnection id.

Parameters

Parameter	***Description***
connectionId	The connection id associated with a email.

Returns

Returns line object, null if there is no connection with that emailconnection id.

getEmailconnectionindexByConnectionId

Syntax

getEmailconnectionindexByConnectionId(*connectionId*)

Description

Get the emailconnection index with the emailconnection id.

Parameters

Parameter	*Description*
connectionId	The connection id associated with a email.

Returns

Returns index, -1 if there is no connection with that emailconnection id.

getFreeEmailconnection

Syntax

getFreeEmailconnection()

Description

Get the free email connection object. A connection is free if there is no activity on this line.

Returns

Returns emailconnection object, null if there is no free line.

EmailAddress Callback Event Methods

The EmailAddress class inherits the onError callback event method from the _Address class.

See _Address Class Callback Event Method.

The EmailAddress class inherits the following callback event methods from the _UQAddress class:

- onAccepted
- onAcceptingTask
- onDequeueingTask
- onNotify
- onTaskAdded
- onTaskRemoved
- onUnassigned

See _UQAddress Class Callback Event Methods.

EmailConnection Class Constructor

The following is EmailConnection class constructor.

EmailConnection

Syntax

```
EmailConnection()
```

Description

Handles email connection.

EmailConnection Class Fields

The following are EmailConnection class fields.

caps

Description

The capabilities of the EmailConnection.

connectionId

Description

The email connection Id.

Type: string

email

Description

Email task id associated with this email connection.

Type: string with following constants:

id

Description

The id of the EmailConnection.

Type: string

state

Description

The connection state.

Type: string

Value	*Description*
ST_DIALING	Dialing state.
ST_DROPPED	Dropped state.
ST_IDLE	Idle state.
ST_INCOMING	Incoming state.
ST_TALKING	Talking state.

EmailConnection Class Methods

The following are the EmailConnection class methods:

abandon

Syntax

abandon(*reason*)

Description

Abandon email task without completing the task .

Parameters

Parameter	*Description*
Reason	The reason code to abandon.

Returns

Request number.

answer

Syntax

answer(*reason*)

Description

Answer/Accept the email assignment.

Parameters

Parameter	*Description*
Reason	The reason code to answer.

Returns

Request number.

attachUserData

Syntax

attachUserData(*userData*,*reason*)

Description

Attach user data for the email connection.

Parameters

Parameter	*Description*
userData	User data of the email connection.
reason	The reason code for the answer request.

Returns

Request number.

complete

Syntax

complete(*reason*)

Description

Send task completion notification.

Parameters

Parameter	*Description*
Reason	The reason code to complete.

Returns

Request number.

forward

Syntax

forward(*fromagentId*,*toagentId*, *qid*, *userdata*, *emaildata*, *reason*)

Description

Forward email to another agent or to another queue.

Parameters

Parameter	*Description*
fromagentid	The agent id from whom task is forwarded.
toagentid	The target agent id to whom task is forwarded.
qid	The target queue to forward the task.
userdata	The user data.
emaildata	The email data.
Reason	The reason code to forward the task.

Returns

Request number.

reject

Syntax

reject(*reason*)

Description

Reject email assignment.

Parameters

Parameter	*Description*
Reason	The reason code to reject.

Returns

Request number.

withdraw_RES

Syntax

```
withdraw_RES
(reason)
```

Description

Send response for the request of (Withdraw of Email) by routing server. This is the response from the agent to the routing system.

Parameters

Parameter	*Description*
Reason	The reason code to withdraw.

Returns

Request number.

EmailConnection Class Callback Event Methods

The following are EmailConnection class callback event methods:

onAnswering

Syntax

```
onAnswering(event)
```

Description

Fires when email task is getting answered.

onCapabilitiesChanged

Syntax

```
onCapabilitiesChanged(event)
```

Description

Fires when there is a change in connection capabilities.

onCompleted

Syntax

onCompleted(*event*)

Description

Fires when EmailConnection state is changed to COMPLETED.

onDropped

Syntax

onDropped(*event*)

Description

Fires when EmailConnection state is changed to DROPPED.

onEmaildataChanged

Syntax

onEmaildataChanged(*event*)

Description

Fires when there is a change in email data.

onError

Syntax

onError(*event*)

Description

Fires when a there is a email connection error.

onForwarded

Syntax

onForwarded(*event*)

Description

Fires when email is forwarded to another queue or to another agent.

onForwardError

Syntax

onForwardError(*event*)

Description

Fires when there is error in forwarding the task.

onForwarding

Syntax

onForwarding(*event*)

Description

Fires when forwarding the email.

onIncoming

Syntax

onIncoming(*event*)

Description

Fires when EmailConnection state is changed to INCOMING.

onProcessing

Syntax

onProcessing(*event*)

Description

Fires when EmailConnection state is changed to PROCESSING.

onRejected

Syntax

onRejected(*event*)

Description

Fires when email task is rejected by agent and the routing system acknowledges the same.

onRevoked

Syntax

onRevoked(*event*)

Description

Fires when email is revoked by routing system before agent accepts that.

onUserdataChanged

Syntax

onUserdataChanged(*event*)

Description

Fires when there is a change in user data.

onWithdraw_REQ

Syntax

onWithdraw_REQ(*event*)

Description

Fires when there is a request for email withdraw from routing server. This is a request from routing server to the agent.

EmailConnectionCaps Class Constructor

The following is EmailConnectionCaps class constructor:

EmailConnectionCaps

Syntax

EmailConnectionCaps(*strCaps*)

Description

Describes the EmailConnection capabilities.

Parameters

Parameter	*Description*
strCaps	The capability of the connection

Returns

Returns EmailConnection object.

EmailConnectionCaps Class Fields

The following are the EmailConnectionCaps class fields.

canAnswer

Description

Answer/Accept an email capability.

Type: boolean

canComplete

Description

Capability to complete an assigned email.

Type: boolean

canForward

Description

Capability to forward an email.

Type: boolean

canReject

Description

Capability to reject an email.

Type: boolean

EmailData Class Constructor

The following is EmailConnectionCaps class constructor.

EmailData

Syntax

```
EmailData()
```

Description

The EmailData object describes the key-value pairs of the email data with the Email object.

Returns

Returns EmailData object.

EmailData Class Fields

The following is EmailData class field.

data

Description

Key value pairs that includes email data.

Type: object.

EmailData Class Methods

The following is EmailData class method.

addKeyValue

Syntax

addKeyValue(*key, value*)

Description

Add key value to the emaildata object.

Parameters

Parameter	*Description*
key	The variable name.
value	The value of the variable.

Extension Class Constructor

The Extension class extends the _Address class.

See _Address Class Constructor.

The following is the Extension class constructor.

Extension

Syntax

Extension(*numOfLines*)

Description

The Extension object describes the CTI address.

Parameters

Parameter	*Description*
numOfLines	Enter the number of lines for this extension.

Returns

An Extension object.

Extension Class Fields

The Extension class inherits the following fields from the _Address class:

- caps
- id

See _Address Class Fields.

The following are the Extension class fields.

forwardMode

Description

The forward mode of the extension.

Type: forwardMode object.

Related Links

ForwardMode Class Constructor

isDnd

Description

Flag for Do Not Disturb.

Type: boolean.

lines

Description

List of line objects associated with this extension.

Type: list.

numOfLines

Description

The number of lines for this extension.

Type: number.

Extension Class Methods

The Extension class inherits the onError method from the Address class.

The following are the Extension class methods.

cancelDnd

Syntax

cancelDnd(*reason*)

Description

Cancel the DND (do not disturb).

Parameters

Parameter	***Description***
reason	The reason for DND cancellation.

Returns

Request number.

cancelForwardSet

Syntax

cancelForwardSet(*reason*)

Description

Cancel the forward.

Parameters

Parameter	*Description*
reason	The reason for cancelling the forward.

Returns

Request number.

forwardSet

Syntax

forwardSet(*number*, *mode*, *reason*)

Description

Forward the call to another number/extension.

Parameters

Parameter	*Description*
mode	The forward mode.
number	The number to which all calls will be forwarded.
reason	The reason to forward.

Returns

Request number.

getDialingLine

Syntax

getDialingLine()

Description

Get the dialing line object.

Parameters

None

Returns

Returns a line object. Null if there is no dialing line.

Related Links

Line Class Constructor

getFreeLine

Syntax

getFreeLine()

Description

Get the free line object. A line is free if there is no activity on this line.

Parameters

None

Returns

Returns a line object. It returns null if there is no free line.

getLineByConnectionId

Syntax

getLineByConnectionId(*connectionID*)

Description

Get the line object with the call.

Parameters

Parameter	***Description***
connectionId	The connection ID associated with a call.

Returns

Returns a Line object. Returns null if there is no line with that call.

Related Links

Line Class Constructor

getOffHookLine

Syntax

getOffHookLine()

Description

Get the offhook line object.

Parameters

None.

Returns

Returns a Line object. Returns null if there is no offhook line.

Related Links

Line Class Constructor

setDnd

Syntax

setDnd(*reason*)

Description

Fires when setting DND (do not disturb).

Parameters

Parameter	***Description***
reason	The reason to set the DND.

Returns

Request number.

Extension Class Callback Event Methods

The following are the Extension class callback event methods.

onCancelingDnd

Syntax

onCancelingDnd(*event*)

Description

Fires when canceling DND (do not disturb).

onCancelingForward

Syntax

onCancelingForward(*event*)

Description

Fires when canceling forward.

onDnd

Syntax

onDnd(*event*)

Description

Fires when DND (Do Not Disturb) is requested and processed.

onDndCanceled

Syntax

onDndCanceled(*event*)

Description

Fires when DND is cancelled.

Parameters

Parameter	*Description*
event	The event object.

Returns

None.

onForwardCanceled

Syntax

onForwardCanceled(*event*)

Description

Fires when forward is canceled.

onForwarded

Syntax

onForwarded(*event*)

Description

Fires when call is forwarded.

onForwarding

Syntax

onForwarding(*event*)

Description

Fires when forwarding the call.

onSettingDnd

Syntax

onSettingDnd(*event*)

Description

Fires when setting DND (do not disturb).

ExtensionCaps Class Constructor

The following is the Extension class constructor.

ExtensionCaps

Syntax

ExtensionCaps(*strCaps*)

Description

Describes the extension's capabilities.

Parameters

Parameter	***Description***
strCaps	A string comprising the extension's capabilities.

Returns

An ExtensionCaps object.

ExtensionCaps Class Fields

The following are the Extension class fields.

canCancelDnd

Description

This extension can cancel DND (do not disturb).

Type: boolean.

canDial

Description

This extension can dial out.

Type: boolean.

canFwdBusy

Description

This extension can forward calls if busy.

Type: boolean.

canFwdBusyNoAnswer

Description

This extension can forward calls if busy/no answer.

Type: boolean.

canFwdCancelForward

Description

This extension can cancel forward.

Type: boolean.

canFwdDefault

Description

This extension can forward.

Type: boolean.

canFwdNoAnswer

Description

This extension can forward if no answer.

Type: boolean.

canFwdUnconditional

Description

This extension can forward unconditionally.

Type: boolean.

canRefreshState

Description

This extension can refresh state.

Type: boolean.

canSetDnd

Description

This extension can set DND (do not disturb).

Type: boolean.

ForwardMode Class Constructor

The following is the ForwardMode class constructor.

ForwardMode

Syntax

```
ForwardMode()
```

Description

Describes various forward modes that can be used while setting the forwarding mode for an Address.

Parameters

None.

Returns

A ForwardMode object.

ForwardMode Class Field

The following is the ForwardMode class field.

mode

Description

The current forwarding mode.

Type: string of the following constants.

Value	*Description*
BUSY	The BUSY forwarding mode.
BUSYNOANSWER	The BUSYNOANSWER forwarding mode.
DEFAULT	The DEFAULT forwarding mode.
NOANSWER	The NOANSWER forwarding mode.
NONE	No forwarding mode.
UNCONDITIONAL	The UNCONDITIONAL forwarding mode.

GenericAddress Class Constructor

The GenericAddress class extends the _UQAddress class.

The following is the GenericAddress class constructor.

GenericAddress

Syntax

```
GenericAddress()
```


Description

Handles the creation of generic tasks.

Parameters

None.

Returns

A GenericAddress object.

GenericAddress Class Fields

The GenericAddress class inherits the following fields from the _Address class.

- caps
- id

  The address's ID is equal to Task.TYPE_GENERIC.

See _Address Class Fields.

The GenericAddress class inherits the tasks field from the _UQAddress class.

See _UQAddress Class Fields.

The following are GenericAddress class fields:

agent

Description

The agent id of this generic address owner.

Type: object

genericconnections

Description

List of GenericConnections.

Type: genericconnectionobject

GenericAddress Class Methods

The GenericAddress class inherits the following methods from the _UQAddress class:

- acceptTask
- dequeueTask

See _UQAddress Methods.

The following are GenericAddress own class methods.

getFreeGenericconnection

Syntax

```
getFreeGenericconnectio()
```

Description

Get the free generic connection object. A connection is free if there is no activity on this line.

Parameters

None.

Returns

Returns genericconnection object, null if there is no free line.

getGenericconnectionByConnectionId

Syntax

```
getFreeGenericconnectio(connectionId)
```

Description

Get the genericconnection object with the genericconnection id.

Parameters

Parameter	*Description*
connectionId	The connection id associated with a generic.

Returns

Returns line object, null if there is no connection with that genericconnection id.

getGenericconnectionindexByConnectionId

Syntax

getGenericconnectionindexByConnectionId(*connectionId*)

Description

Get the genericconnection index with the genericconnection id.

Parameters

Parameter	*Description*
connectionId	The connection id associated with a generic.

Returns

Returns index, -1 if there is no connection with that genericconnection id.

GenericAddress Class Callback Event Methods

The GenericAddress class inherits the onError callback event method from the _Address class.

See _Address Class Callback Event Method.

The GenericAddress class inherits the following callback event methods from the _UQAddress class:

- onAcceptingTask
- onDequeueingTask
- onTaskAdded
- onTaskRemoved
- onNotify
- onAccepted
- onUnassigned

See _UQAddress Class Callback Event Methods.

GenericConnection Class Constructor

The following is the GenericConnection class constructor.

GenericConnection

Syntax

```
GenericConnection()
```

Description

Handles Generic connection.

Parameters

None.

Returns

A genericconnection object.

GenericConnection Class Fields

The following are GenericConnection class fields:

caps

Description

The capabilities of the GenericConnection.

connectionId

Description

The generic connection Id.

Type: string

generic

Description

The generic task id associated with this generic connection.

Type: string

id

Description

The id of the GenericConnection.

Type: string

state

Description

The connection state.

Type: string with following constants:

Value	*Description*
ST_DIALING	Dialing state.
ST_DROPPED	Dropped state.
ST_IDLE	Idle state.
ST_INCOMING	Incoming state.
ST_TALKING	Talking state.

GenericConnection Class Methods

The following are the GenericConnection class methods:

abandon

Syntax

abandon(*reason*)

Description

Abandon generic task.

Parameters

Parameter	*Description*
Reason	The reason code to abandon.

Returns

Request number.

answer

Syntax

answer(*reason*)

Description

Answer/Accept the generic task assignment.

Parameters

Parameter	*Description*
Reason	The reason code to answer.

Returns

Request number.

attachUserData

Syntax

attachUserData(*userData*,*reason*)

Description

Attach user data for the generic connection.

Parameters

Parameter	*Description*
userData	User data of the generic connection.
reason	The reason code for the answer request.

Returns

Request number.

complete

Syntax

complete(*reason*)

Description

Send task completion notification.

Parameters

Parameter	*Description*
Reason	The reason code to complete.

Returns

Request number.

forward

Syntax

forward(*fromagentId*,*toagentId*, *qid*, *userdata*, *genericdata*, *reason*)

Description

Forward generic task to another Agent or to another queue.

Parameters

Parameter	*Description*
fromagentid	The agent id from whom task is forwarded.
toagentid	The target agent id to whom task is forwarded.
qid	The target queue to forward the task.
userdata	The user data.
emaildata	The email data.
Reason	The reason code to forward the task.

Returns

Request number.

reject

Syntax

reject(*reason*)

Description

Reject generic task assignment.

Parameters

Parameter	*Description*
Reason	The reason code to reject.

Returns

Request number.

withdraw_RES

Syntax

withdraw_RES
(*reason*)

Description

Send response for the request of (Withdraw of Generic task) by routing server. This is the response from the agent to the routing system.

Parameters

Parameter	*Description*
Reason	The reason code to withdraw.

Returns

Request number.

GenericConnection Class Callback Event Methods

The following are ChatConnection class callback event methods:

onCapabilitiesChanged

Syntax

onCapabilitiesChanged(*event*)

Description

Fires when there is a change in GenericData.

onCompleted

Syntax

onCompleted(*event*)

Description

Fires when GenericConnection state is changed to COMPLETED.

onDropped

Syntax

onDropped(*event*)

Description

Fires when GenericConnection state is changed to DROPPED.

onError

Syntax

onError(*event*)

Description

Fires when a there is a generic connection error.

onForwarded

Syntax

onForwarded(*event*)

Description

Fires when generic task is forwarded.

onForwardError

Syntax

onForwardError(*event*)

Description

Fires when there is error in forwarding the generic task.

onForwarding

Syntax

onForwarding(*event*)

Description

Fires when forwarding the generic task.

onGenericdataChanged

Syntax

onGenericdataChanged(*event*)

Description

Fires when there is a change in GenericData.

onIncoming

Syntax

onIncoming(*event*)

Description

Fires when GenericConnection state is changed to INCOMING.

onProcessing

Syntax

onProcessing(*event*)

Description

Fires when GenericConnection state is changed to PROCESSING.

onRejected

Syntax

onRejected(*event*)

Description

Fires when generic task is rejected by agent and the routing system acknowledges the same.

onRevoked

Syntax

onRevoked(*event*)

Description

Fires when generic task is revoked by routing system before agent accepts that.

onUserdataChanged

Syntax

onUserdataChanged(*event*)

Description

Fires when there is a change in user data.

onWithdraw_REQ

Syntax

onWithdraw_REQ(*event*)

Description

Fires when there is a request for generic task withdraw from routing server. This is a request from routing server to the agent.

GenericConnectionCaps Class Constructor

The following is GenericConnectionCaps class constructor.

GenericConnectionCaps

Syntax

GenericConnectionCaps(*strCaps*)

Description

Describes the GenericConnection capabilities.

Parameters

Parameter	*Description*
strCaps	The capability of the connection.

Returns

Returns GenericConnection object.

GenericConnectionCaps Class Fields

The following are the GenericConnectionCaps class fields

canAnswer

Description

Answer/Accept a generic task capability.

Type: boolean

canComplete

Description

Complete an assigned generic task capability.

Type: boolean

canForward

Description

Forward a generic task capability.

Type: boolean

canReject

Description

Reject a generic task capability.

Type: boolean

GenericData Class Constructor

The following is GenericConnectionCaps class constructor:

GenericData

Syntax

```
GenericData()
```

Description

The GenericData object describes the key-value pairs of the generic data with the generic object

Returns

Returns GenericData object.

GenericData Class Fields

The following is GenericData class field:

data

Description

Key value pairs that includes data.

Type: object.

GenericData Class Methods

The following is GenericData class method:

addKeyValue

Syntax

```
addKeyValue(key, value )
```

Description

Add key value to the genericdata object.

Parameters

Parameter	*Description*
key	The variable name.
value	The value of the variable.

GenericTask Class Constructor

The GenericTask class extends the Task class.

The following is the GenericTask class constructor

GenericTask

Syntax

GenericTask(*event*)

Description

The GenericTask object describes the generic task information for tasks associated with the GenericAddress object.

Parameters

Parameter	*Description*
event	The triggering event.

Returns

A GenericTask object.

GenericTask Class Fields

The GenericTask class inherits the following fields from the Task class:

- caseid
- cost
- customerid

- group
- id
- onStat
- priority
- type

  The task type for generic task that is Task.TYPE_GENERIC.
- urlAbs
- urlRel

See Task Class Fields.

The following are the additional GenericTask class fields.

address

Description

The address containing the A2AChat.

agentId

Description

The agent id to whom this task is assigned.

Returns a string.

appData

Description

The application data that is provided to the client with the notification event.

Type: AppData object.

Related Links

AppData Class Constructor

customerName

Description

The customer username.

Type: string.

genericconnection

Description

The generic connection object that associated with this Generic task.

Type: line

genericId

Description

The unique generic id generated from an Enqueue. It identifies the data in the database associated with the generic task.

Type: string.

groupId

Description

The group id for which task is initiated.

Type: string.

question

Description

The question of this generic task which is raised by customer.

Type: string.

statistics

Description

Task statistics of the generic task.

Type: object.

subject

Description

The subject of the generic task.

Type: string.

userdata

Description

User data object that is associated with this generic task.

Type: object.

GenericTask Class Method

The following is the GenericTask class method.

gettpUrl

Syntax

gettpUrl(*defaultUrl*)

Description

Returns the URL for the given task for the third-party routing.

Parameters

Parameter	***Description***
defaultUrl	If not null, this value will override the generated base URL.

Returns

Returns the URL for the given task.

getUrl

Syntax

getUrl(*defaultUrl*)

Description

Returns the URL for the given task.

Parameters

Parameter	*Description*
defaultUrl	If not null, this value will override the generated base URL.

Returns

Returns the URL for the given task.

GLOBALS Class Fields

The following are global fields, which may be accessed without instantiating an object. They are accessed as shown.

A2AChat.PS_JR

Description

The Journal Routing constant.

Example

A2AChat.PS_JR = "ps_jr";

A2AChat.TYPE_ANSWER

Description

The constant representing an A2AChat type of answer.

This type is generated when a different user wants this user to answer an A2AChat.

Example

A2AChat.TYPE_ANSWER = "answer";

A2AChat.TYPE_CONSULT

Description

The constant representing an A2AChat type of consult.

This type is generated when this user wants to consult a different user.

Example

A2AChat.TYPE_CONSULT = "consult";

Server.TYPE_CTI

Description

The constant representing a CTI server.

Example

Server.TYPE_CTI = "CTI";

Server.TYPE_UQ

Description

The UQ server type.

Example

Server.TYPE_UQ = "UQ";

Task.TYPE_A2ACHAT

Description

The constant representing an A2AChat.

Example

Task.TYPE_A2ACHAT = "A2ACHAT";

Task.TYPE_CHAT

Description

The constant representing a chat task.

Example

Task.TYPE_CHAT = "CHAT";

Task.TYPE_CTI

Description

The constant representing a CTI task.

Example

Task.TYPE_CTI = "CTI";

Task.TYPE_EMAIL

Description

The constant representing an email task.

Example

Task.TYPE_EMAIL = "EMAIL";

Task.TYPE_GENERIC

Description

The constant representing a generic task.

Example

Task.TYPE_GENERIC = "GENERIC";

GLOBALS Class Methods

The following are the GLOBALS class methods.

initJSMCAPI

Syntax

```
initJSMCAPI()
```

Description

The initialization function of the JSMCAPI. This is the first function should be called by application.

Parameters

None.

Returns

None. Initializes JSMCAPI.

isValid

Syntax

isValid(*obj*)

Description

Returns true if an object is not undefined and not null. *Obj* can be any JavaScript object or literal. This method is a convenience method to check that an object is both not null and not undefined.

Parameters

Parameter	***Description***
obj	*Obj* can be any JavaScript object or literal.

Returns

Returns a boolean.

Returns true if an object is not undefined and not null.

MCFBroadcast

Syntax

MCFBroadcast(*cluster*, *queue*, *task*, *state*, *presence*, *message*, *securitylevel*, *importa*⇒ *ncelevel*, *senderid*, *NameValuePairString*)

Description

Broadcast a message to any queue, cluster, or only agents.

Parameters

Parameter	*Description*
cluster	Cluster Id to which we need to send broadcast message.
queue	Denotes the queue Id in the cluster.
task	Denotes the task, such as email, chat, voice, or generic.
state	Denotes the state, such as LoggedIn and NotLoggedIn.
presence	Denotes whether the agent is ready or not ready.
Message	Enter the message to broadcast.
security level	Security level defined by application developers.
importance level	Importance level defined by application developers.
sender Id	The sender's user id
namevaluepairs	Any extra data formed as name-value pair string.

Returns

None

Group Class Constructor

The following is the Group class constructor.

Group

Syntax

```
Group()
```

Description

The Group object describes the group information.

Parameters

None.

Returns

A Group object.

Group Class Fields

The following are the Group class fields.

id

Description

The group ID.

Type: string.

name

Description

The group name.

Type: string.

registered

Description

True if the Group is registered on the server.

Type: boolean.

statistics

Description

The group statistics for CTI.

Type: GroupStatistics object

Related Links

GroupStatistics1 Class Constructor

statistics1

Description

The group statistics for the queue server.

Type: GroupStatistics1 object.

Related Links

GroupStatistics1 Class Constructor

statistics2

Description

The group statistics for the queue server.

Type: GroupStatistics2 object.

Related Links

GroupStatistics2 Class Constructor

Group Class Callback Event Methods

The following are the Group class callback event methods.

onStat

Syntax

onStat(*event*)

Description

Fires when there is new statistics going to this group.

onStat1

Syntax

onStat1(*event*)

Description

Fires when statistics1 is received.

onStat2

Syntax

onStat2(*event*)

Description

Fires when statistics2 is received.

onTaskAdded

Syntax

onTaskAdded(*event*)

Description

Fires when a task added to this group.

onTaskRemoved

Syntax

onTaskRemoved(*event*)

Description

Fires when a task is removed from this group.

GroupStatistics Constructor

The following is the GroupStatistics class constructor.

GroupStatistics

Syntax

GroupStatistics()

Description

The group statistics information.

Parameters

None.

Returns

A GroupStatistics object.

GroupStatistics Fields

The following are the GroupStatistics class fields.

data

Description

Key value pairs that include all statistics.

listOfTasksInTheQueueByTaskType

Description

Denotes the task type, task state and the time for which the task is in the system.

maxTaskCompletionTime

Description

Longest wait time for a task in the queue.

newestTask

Description

Time elapsed for the most recent task.

newestTaskCompletionTime

Description

Difference between queue time and the time when task is done.

numberOfAbandoned

Description

Number of tasks that are abandoned.

numberOfLoggedIn

Description

Number of agents that are logged in the queue.

numberOfQueued

Description

Number of tasks that are queued.

numUnassignedTasks

Description

Number of unassigned tasks.

queuedWaitTime

Description

The average wait time, in seconds, of a queued task.

queueUpTime

Description

Time since the queue is available on the system (startup/boot time).

relativeQueueLoad

Description

Relative queue load.

timeElapsedOldestTask

Description

Time elapsed for the oldest task (difference between current time and en-queue time).

GroupStatistics1 Class Constructor

The following is the GroupStatistics1 class constructor.

GroupStatistics1

Syntax

GroupStatistics1(*data*)

Description

UQ Group statistics sent on group refresh 1.

Parameters

Parameter	*Description*
data	Statistical data from the UQ server.

Returns

A GroupStatistics1 object.

GroupStatistics1 Class Fields

The following are the GroupStatistics1 class fields.

mostRecentTaskDone

Description

The most recently done task in the group.

Type: string.

mostRecentTaskDoneData

Description

The data for the most recent task done.

Type: string.

mostRecentTaskEnqueued

Description

The most recently enqueued task in the group.

Type: string.

mostRecentTaskEnqueuedData

Description

The data for the most recently enqueued task.

Type: string.

numAgentsAvailable

Description

Number of agents available in the group.

Type: string.

numAgentsLoggedIn

Description

Number of agents logged in on the group.

Type: string.

numEscalation

Description

Number of escalated tasks in the group.

Type: string.

numOverflow

Description

Number of overflowed tasks in the group.

Type: string.

numTaskAccepted

Description

Number of tasks accepted in the group.

Type: string.

numTaskDone

Description

Number of tasks done in the group.

Type: string.

numTaskQueued

Description

Number of tasks queued in the group.

Type: string.

reasonFlag

Description

The reason flag for this GroupStatistics event.

Type: string.

taskTotalTimeInSystem

Description

The total time in the system.

Type: string.

timeSinceStart

Description

The time since start.

Type: string.

GroupStatistics2 Class Constructor

The following is the GroupStatistics2 class constructor.

GroupStatistics2

Syntax

```
GroupStatistics2(data)
```

Description

UQ Group statistics sent on group refresh 2.

Parameters

Parameter	*Description*
data	UQ server statistical data.

Returns

A GroupStatistics2 object.

GroupStatistics2 Class Fields

The following are the GroupStatistics2 class fields.

averageTaskDuration

Description

The average task duration in the group.

Type: string.

averageWaitTime

Description

The average wait time in the group.

Type: string.

oldestTask

Description

The oldest task in the group.

Type: string.

recentTask

Description

The most recent task in the group.

Type: string.

timeElapsedOldestTask

Description

The time elapsed for the oldest task in the group.

Type: string.

timeElapsedRecentTask

Description

The time elapsed for the most recent task in the group.

Line Class Constructor

The following is the Line class constructor.

Line

Syntax

```
Line()
```

Description

The Line object describes the line of the extension. JSMCAPI only supports one extension with two lines and two extensions with one line in each.

Parameters

None.

Returns

A Line object.

Line Class Fields

The following are the Line class fields.

call

Description

The Call object on the line.

Type: Call object.

Related Links

Call Class Constructor

caps

Description

The capabilities of the line.

Type: LineCaps object.

Related Links

LineCaps Class Constructor

connectionid

Description

The call connection ID on the line.

Type: string.

id

Description

The line ID.

Type: string.

isMuted

Description

Flag to see whether extension is muted or not.

Type: string.

state

Description

The line state.

Type: string with the following constants:

Value	***Description***
ST_DIALING	The dialing state.
ST_DROPPED	The dropped state.
ST_HELD	The held state.
ST_IDLE	The idle state.
ST_OFFHOOK	The offhook state.
ST_RINGING	The ringing state.

Value	*Description*
ST_TALKING	The talking state.

Line Class Methods

The following are the Line class methods.

alternate

Syntax

alternate(*reason*)

Description

Alternate a call.

Parameters

Parameter	*Description*
reason	The reason to alternate a call.

Returns

Request number.

answer

Syntax

answer(*reason*)

Description

Answer a call.

Parameters

Parameter	*Description*
reason	The reason to answer a call.

Returns

Request number.

attachUserData

Syntax

attachUserDate(*userdata, reason*)

Description

Attach user data to a call.

Parameters

Parameter	*Description*
userdata	The user data to attach to the call.
reason	The reason to attach the user data.

Returns

Request number.

clear

Syntax

clear(*reason*)

Description

Clear a conference call.

Parameters

Parameter	*Description*
reason	The reason to clear the conference.

Returns

Request number.

complete

Syntax

complete(*reason*)

Description

Complete the two-step transfer/conference.

Parameters

Parameter	*Description*
reason	The reason to complete the two-step transfer/conference.

Returns

Request number.

conference

Syntax

conference(*destination*, *reason*, *userdata*, *calldata*)

Description

Conference a call.

Parameters

Parameter	*Description*
destination	The destination being invited into the conference.
reason	The reason to conference.
userdata	The user data to be attached.
calldata	The call data to be modified.

Returns

Request number.

conferenceSingle

Syntax

conferenceSingle(*destination*, *reason*, *userdata*, *calldata*)

Description

Single-step conference a call.

Parameters

Parameter	*Description*
destination	The destination being invited into the conference.
reason	The reason to conference.
userdata	The user data to be attached.
calldata	The call data to be modified.

Returns

Request number.

dial

Syntax

dial(*number*, *reason*, *userdata*)

Description

Dial out.

Parameters

Parameter	*Description*
destination	The destination being invited into the conference.
reason	The reason to conference.
userdata	The user data to be attached.

Returns

Request number.

dropParty

Syntax

dropParty(*destination*, *reason*)

Description

Drop a party in conference.

Parameters

Parameter	*Description*
destination	The destination being dropped from the conference.
reason	The reason to drop the party.

Returns

Request number.

getAni

Syntax

```
getAni()
```

Description

Get the ANI.

Parameters

None.

Returns

Returns a string representing the ANI.

getDescr

Syntax

```
getDescr()
```

Description

Get the description attached to the call on the line.

Parameters

None.

Returns

Returns a string.

getDnis

Syntax

```
getDnis()
```

Description

Get the DNIS.

Parameters

None.

Returns

Returns a string.

getPadvalue

Syntax

getPadvalue(*key*)

Description

Get the PAD value.

Returns

A string representing PAD value.

getReferenceId

Syntax

getReferenceId()

Description

Get the reference ID attached to the call on the line.

Parameters

None.

Returns

Returns a string.

getUrl

Syntax

getUrl(*defaultUrl*)

Description

Get the URL for screen pop-up.

Parameters

Parameter	*Description*
defaultUrl	The default pop-up URL.

Returns

Returns a string.

grabCall

Syntax

grabCall(*destination, reason*)

Description

Grab a call from the queue.

Parameters

Parameter	*Description*
destination	The destination to be grabbed.
reason	The reason for the grab.

Returns

Request number.

hold

Syntax

hold(*reason*)

Description

Hold a call.

Parameters

Parameter	*Description*
reason	The reason to hold the call.

Returns

Request number.

join

Syntax

join(*reason, conferenceId*)

Description

Join an existing call/conference.

Parameters

Parameter	*Description*
reason	The reason to join the call.
conferenceId	Call id.

Returns

Request number.

mute

Syntax

mute(*reason*)

Description

Mute an extension.

Parameters

Parameter	*Description*
reason	The reason to mute the call.

Returns

Request number.

park

Syntax

park(*destination, reason*)

Description

Park a call.

Parameters

Parameter	*Description*
destination	The destination on which the call will be parked.
reason	The reason to park the call.

Returns

Request number.

reconnect

Syntax

reconnect(*reason*)

Description

Reconnect to the original party during two-step transfer/conference.

Parameters

Parameter	*Description*
reason	The reason to reconnect.

Returns

Request number.

reject

Syntax

reject(*reason*)

Description

Reject an incoming call.

Parameters

Parameter	*Description*
reason	The reason to reject.

Returns

Request number.

release

Syntax

release(*reason*)

Description

Release a call.

Parameters

Parameter	*Description*
reason	The reason to release the call.

Returns

Request number.

retrieve

Syntax

retrieve(*reason*)

Description

Retrieve a held call.

Parameters

Parameter	*Description*
reason	The reason to retrieve the call.

Returns

Request number.

sendDTMF

Syntax

Syntax

sendDTMF(*reason*,*stringDTMF*)

Description

Send DTMF tones to the switch.

Parameters

Parameter	*Description*
reason	The reason for requesting DTMF.
stringDTMF	DTMF tones string (0-9,*,#).

Returns

Request number.

setcallresult

Syntax

setcallresult(*result*)

Description

Set call result.

Parameters

Parameter	*Description*
result	Result of the call.

Returns

Request number.

setcallresultDNC

Syntax

setcallresultDNC(*number, reason*)

Description

Do not call.

Parameters

Parameter	*Description*
number	The number not to be disturbed.
reason	The reason not to disturb.

Returns

Request number.

setcallresultReschedule

Syntax

setcallresultReschedule(*hour, minutes, day, month,year*)

Description

Reschedule a call.

Parameters

Parameter	*Description*
hour	Hours
Minutes	Minutes
Day	Day
Month	Month
Year	Year

Returns

Request number.

transfer

Syntax

transfer(*destination*, *reason*, *userdata*, *calldata*)

Description

Transfer a call.

Parameters

Parameter	*Description*
destination	The destination to which the call will be transferred.
reason	The reason to transfer.
userdata	The user data to be attached.
calldata	The call data to be modified.

Returns

Request number.

transferMute

Syntax

transferMute(*destination*, *reason*, *userdata*, *calldata*)

Description

Transfer mute a call.

Parameters

Parameter	*Description*
destination	The destination to which the call will be transferred.
reason	The reason to transfer.

Parameter	*Description*
userdata	The user data to be attached.
calldata	The call data to be modified.

Returns

Request number.

unmute

Syntax

unmute(*reason*)

Description

Unmute a call.

Parameters

Parameter	*Description*
reason	The reason to unmute a call.

Returns

Request number.

updateCallData

Syntax

updateCallData(*calldata, reason*)

Description

Update call data to a call.

Parameters

Parameter	*Description*
calldata	The call data to update.
reason	The reason to update.

Returns

Request number.

Line Class Callback Event Methods

The following are the Line class callback event methods.

onAlternating

Syntax

onAlternating(*event*)

Description

Fires when alternating the call.

onAnswering

Syntax

onAnswering(*event*)

Description

Fires when answering the call.

onAttachingUD

Syntax

onAttachingUD(*event*)

Description

Fires when attaching user data.

onCallDataChanged

Syntax

onCallDataChanged(*event*)

Description

Fires when the call data on the call is changed.

onCapabilitiesChanged

Syntax

onCapabilitiesChanged(*event*)

Description

Fires when the capabilities changed.

onClearing

Syntax

onClearing(*event*)

Description

Fires when clearing the conference.

onCompleting

Syntax

onCompleting(*event*)

Description

Fires when completing the two-step transfer/conference.

onConferencing

Syntax

onConferencing(*event*)

Description

Fires when conferencing the call.

onDialing

Syntax

onDialing(*event*)

Description

Fires when there is an outgoing call.

onDropped

Syntax

onDropped(*event*)

Description

Fires when the call is released.

onError

Syntax

onError(*event*)

Description

Fires when there is error.

onGrabbing

Syntax

onGrabbing(*event*)

Description

Fires when grabbing a call from a queue.

onHeld

Syntax

onHeld(*event*)

Description

Fires when the call is on hold.

onHolding

Syntax

onHolding(*event*)

Description

Fires when holding the call.

onJoining

Syntax

onJoining(*event*)

Description

Fires when joining a call or a conference.

onMuted

Syntax

onMuted(*event*)

Description

Fires when a call is muted.

onOffHook

Syntax

onOffHook(*event*)

Description

Fires when the line is off hook.

onOnHook

Syntax

onOnHook(*event*)

Description

Fires when the line is on hook.

onParking

Syntax

onParking(*event*)

Description

Fires when parking the call.

onPartyAdded

Syntax

onPartyAdded(*event*)

Description

Fires when a new call party coming.

onPartyChanged

Syntax

onPartyChanged(*event*)

Description

Fires when the call party has changed.

onPartyRemoved

Syntax

onPartyRemoved(*event*)

Description

Fires when a call party is removed.

onReconnecting

Syntax

onReconnecting(*event*)

Description

Fires when reconnecting the call.

onRejected

Syntax

onRejected(*event*)

Description

Fires when call is rejected.

onRejecting

Syntax

onRejecting(*event*)

Description

Fires agent rejects an incoming call.

onReleasing

Syntax

onReleasing(*event*)

Description

Fires when releasing the call.

onRetrieving

Syntax

onRetrieving(*event*)

Description

Fires when retrieving the call.

onRinging

Syntax

onRinging(*event*)

Description

Fires when there is an incoming call.

onSetcallresult

Syntax

onSetcallresult(*event*)

Description

Fires when there is a request for setting call result.

onSetcallresultDNC

Syntax

onSetcallresultDNC(*event*)

Description

Fires when there is a request for do not call.

onSetcallresultReschedule

Syntax

onSetcallresultReschedule(*event*)

Description

Fires when there is a request to reschedule a call.

onTalking

Syntax

onTalking(*event*)

Description

Fires when the call is established.

onTransfering

Syntax

onTransfering(*event*)

Description

Fires when transferring the call.

onUnmuted

Syntax

onUnmuted(*event*)

Description

Fires when call is unmuted.

onUpdatingCD

Syntax

onUpdatingCD(*event*)

Description

Fires when updating the call data.

onUserDataChanged

Syntax

onUserDataChanged(*event*)

Description

Fires when the user data attached on the call is changed.

LineCaps Class Constructor

The following is the LineCaps class constructor.

LineCaps

Syntax

LineCaps(*strCaps*)

Description

Describes the line's capabilities.

Parameters

Parameter	***Description***
strCaps	The line's capabilities.

Returns

A LineCaps object.

LineCaps Class Fields

The following are the LineCaps class fields.

canAlternate

Description

The line has alternate capability.

Type: boolean.

canAnswer

Description

The line has answer capability.

Type: boolean.

canAttachUserDate

Description

The line has attach user data capability.

Type: boolean.

canClear

Description

The line has clear capability.

Type: boolean.

canComplete

Description

The line has complete capability.

Type: boolean.

canConference

Description

The line has conference capability.

Type: boolean.

canConferenceSingle

Description

The line has conference single capability.

Type: boolean.

canDropParty

Description

The line has drop party capability.

Type: boolean.

canHold

Description

The line has hold capability.

Type: boolean.

canMute

Description

The line has mute capability.

Type: boolean.

canPark

Description

The line has park capability.

Type: boolean.

canReconnect

Description

The line has reconnect capability.

Type: boolean.

canReject

Description

The line has reject capability.

Type: boolean.

canRelease

Description

The line has release capability.

Type: boolean.

canRetrieve

Description

The line has retrieve capability.

Type: boolean.

canSendDTMF

Description

The line has DTMF capability.

Type: boolean.

canSetcallresult

Description

The line has set call result capability.

Type: boolean.

canSetcallresultDNC

Description

The line has do not call capability.

Type: boolean.

canSetcallresultReschedule

Description

The line has reschedule capability.

Type: boolean.

canTransfer

Description

The line has transfer capability.

Type: boolean.

canTransferMute

Description

The line has transfer mute capability.

Type: boolean.

canUnmute

Description

The line has unmute capability.

Type: boolean.

canUpdateCallData

Description

The line has update call data capability.

Type: boolean.

MCEvent Class Constructor

The following is the MCEvent class constructor.

MCEvent

Syntax

```
MCEvent()
```

Description

The MCEvent will be passed to the application event handler.

Parameters

None.

Returns

An MCEvent object.

MCEvent Class Fields

The following are the MCEvent class fields.

extension

Description

The extension associated with the event.

Type: Extension object.

Related Links

Extension Class Constructor

group

Description

The group associated with the event.

Type: Group object.

Related Links

Group Class Constructor

reason

Description

The reason associated with the event.

Type: Reason object.

Related Links

Reason Class Constructor

user

Description

The user associated with the event.

Type: User object.

Related Links

User Class Constructor

MediaType Class Constructor

The following is the MediaType class constructor.

MediaType

Syntax

```
MediaType()
```

Description

Distinguishes between email and CTI media types.

Parameters

None.

Returns

A MediaType object.

MediaType Class Field

The following is the MediaType field.

type

Description

The media type.

Type: string with the following constants:

Value	*Description*
MT_CHAT	The chat media type.
MT_EMAIL	The email media type.
MT_GENERIC	The generic media type.
MT_VOICE	The voice media type (CTI).

PSMC Class Constructor

The following is the PSMC class constructor.

PSMC

Syntax

PSMC ( )

Description

The global object that the application can access.

Parameters

None.

Returns

A PSMC object.

PSMC Class Fields

The following are the PSMC class fields.

renserver

Description

The PSMC instance of the RenServer object.

Type: RenServer object.

Related Links

RenServer Class Constructor

servers

Description

An associative array the different server objects, indexed by server ID.

Type: associative array.

Related Links

Server Class Constructor

sessions

Description

An associative array that holds the different session objects, indexed by server ID.

Type: associative array.

Related Links

Session Class Constructor

PSMC Class Methods

The following are the PSMC class methods.

closeSession

Syntax

closeSession(*serverId*)

Description

Closes the specified session and server object and deletes them.

Parameters

Parameter	*Description*
serverId	The ID of the REN server cluster for a UQ server, or the server ID for a CTI server.

Returns

None.

getCallById

Syntax

getCallById(*id*)

Description

Retrieve the call object by the call ID.

Parameters

Parameter	*Description*
id	The call ID.

Returns

Type: Call object.

Related Links

Call Class Constructor

getChatById

Syntax

getChatById(*id*)

Description

Retrieve the chat object by the task id.

Parameters

Parameter	*Description*
id	The task ID.

Returns

Type: chat object.

getEmailById

Syntax

getEmailById(*id*)

Description

Retrieve the email object by the task id.

Parameters

Parameter	*Description*
id	The task ID.

Returns

Type: email object.

getGenericTaskById

Syntax

getGenericTaskById(*id*)

Description

Retrieve the Generic object by the task id.

Parameters

Parameter	*Description*
id	The task ID.

Returns

Type: generic object.

getLineById

Syntax

getLineById(*id*)

Description

Retrieve the line object by the line ID.

Parameters

Parameter	*Description*
id	The line ID.

Returns

Type: Line object.

Related Links

Line Class Constructor

openSession

Syntax

openSession(*id*)

Description

Creates the session and server objects for the PSMC.

Parameters

Parameter	*Description*
id	The ID of the REN server cluster for a UQ server, or the server ID for a CTI server.

Returns

None.

start

Syntax

```
start()
```

Description

Start the JSMCAPI.

Parameters

None.

Returns

None.

stop

Syntax

```
stop()
```

Description

Stop the JSMCAPI.

Parameters

None.

Returns

None.

Reason Class Constructor

The following is the Reason class constructor.

Reason

Syntax

reason(*code*, *desc*, *reasondata1*, *reasondata2*, *reasondata3*)

Description

The Reason object carries the reason code and reason description.

Parameters

Parameter	***Description***
code	The reason code.
desc	The reason description.
reasondata1	The reason data1.
reasondata2	The reason data2.
reasondata3	The reason data3

Returns

Type: Reason object.

Reason Class Fields

The following are the Reason class fields.

code

Description

The reason code.

Type: string.

desc

Description

The reason description.

Type: string.

reasonData1

Description

Place holder for extra data.

Type: string.

reasonData2

Description

Place holder for extra data.

Type: string.

reasonData3

Description

Place holder for extra data.

Type: string.

RenServer Class Constructor

The following is the RenServer class constructor.

RenServer

Syntax

```
RenServer()
```

Description

The RenServer object describes the REN server cluster information.

Parameters

None.

Returns

A RenServer object.

RenServer Class Fields

The following are the RenServer class fields.

isRunning

Description

Flag containing the state of the REN server connection.

Type: boolean.

url

Description

The REN server URL.

Type: string.

RenServer Class Callback Event Methods

The following are the RenServer class callback event methods.

onDown

Syntax

onDown(*event*)

Description

Fires when the REN server is down.

onUp

Syntax

onUp(*event*)

Description

Fires when the REN server is up.

Server Class Constructor

The following is the Server class constructor.

Server

Syntax

Server(*serverId*)

Description

The Server object describes the server information.

Parameters

Parameter	***Description***
serverId	The ID of the REN server cluster for a UQ server, or the server ID for a CTI server.

Returns

A Server object.

Server Class Fields

The following are the Server class fields.

id

Description

The server ID.

Type: string.

info

Description

The server information string.

Type: string.

state

Description

The server state.

Type: string with the following constants.

Value	*Description*
ST_INSERVICE	The server is in service.
ST_OUTOFSERVICE	The server is out of service.

type

Description

The type of server, either CTI or UQ.

Type: string with the following constants.

Value	*Description*
TYPE_CTI	The server is a CTI server.
TYPE_MCS	The server is a third-party MultiChannel server.
TYPE_UQ	The server is a queue server.

Example

The TYPE_* fields are global fields, which may be accessed without instantiating an object. They are accessed as shown in this example:

```
Server.TYPE_UQ = "UQ";
```

Related Links

GLOBALS Class Fields

Server Class Callback Event Methods

The following are the Server class callback event methods.

onBroadcast

Syntax

onBroadcast(*event*)

Description

Fires when there is broadcast message.

onHbLost

Syntax

onHbLost(*event*)

Description

Fires when the server's heartbeat is lost.

onHbRecovered

Syntax

onHbRecovered(*event*)

Description

Fires when the server's heartbeat is recovered.

onInService

Syntax

onInService(*event*)

Description

Fires when the server changes state to in service.

onOutOfService

Syntax

onOutOfService(*event*)

Description

Fires when the server changes state to out of service.

onRestart

Syntax

onRestart(*event*)

Description

Fires when server restarts.

Session Class Constructor

The following is the Session class constructor.

Session

Syntax

Session(*serverId*)

Description

The Session object describes the session with the server.

Parameters

Parameter	*Description*
serverId	The ID of the REN server cluster for a UQ server, or the server ID for a CTI server.

Returns

Returns a Session object.

Session Class Fields

The following are the Session class fields.

addresses

Description

The associative array of addresses that the session register.

Type: associative array.

buddies

Description

The buddy hash table in this session.

Type: associative array.

groups

Description

The group hash table in this session.

Type: associative array.

id

Description

The session ID generated by the server.

Type: string.

intervalBetweenReqs

Description

Interval in milliseconds between requests.

Type: number

numberRegsPerBulkReq

Description

Number of users or groups to register for one bulk register request.

Type: number

serverId

Description

The serverId will be unique for every session. It is used to lookup the protocol objects.

Type: string.

state

Description

The session state.

Type: string with the following constants.

Value	*Description*
ST_ACTIVE	The session is in the active state.
ST_CLOSED	The session is in the closed state.
ST_CLOSING	The session is in the closing state.
ST_IDLE	The session is in the idle state.

user

Description

The current user for the session.

Type: User object.

Related Links

User Class Constructor

Session Class Methods

The following are the Session class methods.

broadcastSubscribe

Syntax

broadcastSubscribe(*cluster*, *queue*, *task*, *state*, *presence*, *method*)

Description

Subscribe to broadcast messages.

Returns

Object.

broadcastUnsubscribe

Syntax

broadcastunsubscribe(*type*)

Description

Unsubscribe to broadcast messages.

Returns

None.

close

Syntax

```
close()
```

Description

Close the session with the server.

Parameters

None.

Returns

Request number.

open

Syntax

```
open()
```

Description

Open the session with the server.

Parameters

None.

Returns

Request number.

registerAddress

Syntax

registerAddress(*address*)

Description

Register the address to the server so that the server will report all events that occur on this address.

Parameters

Parameter	*Description*
address	The address to be registered.

Returns

Request number.

registerBuddy

Syntax

registerBuddy(*buddy*)

Description

Register the buddy to the JSMCAPI so that the JSMCAPI will report all state events that occur on this buddy.

Parameters

Parameter	*Description*
buddy	The buddy to be registered.

Returns

Request number.

registerBuddiesBulk

Syntax

registerBuddiesBulk(*buddies*, *numRegsPerBulkReq*, *intervalBetweenReqs*)

Description

Register multiple buddies to the JSMCAPI so that the JSMCAPI will report state events that occur for the buddies.

Parameters

Parameter	***Description***
buddies	An array of buddies
numRegPerBulkReq	Number of buddies to register in one bulk registration request.
intervalBetweenReqs	Interval in milliseconds between requests.

Returns

Request number.

registerGroup

Syntax

registerGroup(*group*)

Description

Register the group to the server so that the server will report all events that occur on this group.

Parameters

Parameter	***Description***
group	The group to be registered.

Returns

Request number.

registerGroupsBulk

Syntax

registerGroupsBulk(*groups*, *numRegsPerBulkReq*, *intervalBetweenReqs*)

Description

Register multiple groups to the server so that the server will report all events that occur on the groups.

Parameters

Parameter	***Description***
groups	An array of groups.
numRegPerBulkReq	Number of groups to register in one bulk registration request.
intervalBetweenReqs	Interval in milliseconds between requests.

Returns

Request number.

registerUser

Syntax

registerUser(*user*)

Description

Register the user to the server so that the server will report all those events that happen on this user.

Parameters

Parameter	***Description***
user	The user to be registered.

Returns

Request number.

setAutoRecovery

Syntax

setAutoRecovery(*autorecover*)

Description

Sets the auto recovery feature of the queue server connection to off.

Parameters

Parameter	*Description*
autoRecover	Recover.

statPublish

Syntax

statPublish(*userStat, groupStat*)

Description

Sets the statistics publishing levels for the queue server.

Parameters

Parameter	*Description*
userStat	Either 0, 1, or 2 for UserStatistics publishing. Enter 0 for no statistics publishing, 1 for Statistics1 publishing, and 2 for Statistics2 and Statistic1 publishing.
groupStat	Either 0, 1, or 2 for GroupStatistics publishing. Enter 0 for no statistics publishing, 1 for Statistics1 publishing, and 2 for Statistics2 and Statistic1 publishing.

Returns

Request number.

unregisterAddress

Syntax

unRegisterAddress(*address*)

Description

Unregister the address from the server so that the server will not report any events about this address.

Parameters

Parameter	*Description*
address	The address to be unregistered.

Returns

Request number.

unregisterBuddy

Syntax

unregisterBuddy(*buddy*)

Description

Unregister the buddy from the JSMCAPI so that the JSMCAPI will not report any events about the buddy state change.

Parameters

Parameter	*Description*
buddy	The buddy to be unregistered.

Returns

Request number.

unregisterGroup

Syntax

unregisterGroup(*group*)

Description

Unregister the group from the server so that the server will not report any events about this group.

Parameters

Parameter	*Description*
group	The group to be unregistered.

Returns

Request number.

unregisterUser

Syntax

unregisterUser(*user*)

Description

Unregister the user from the server so that the server will not report any events to this user.

Parameters

Parameter	*Description*
user	The user to be unregistered.

Returns

Request number.

Session Class Callback Event Methods

The following are the Session class callback event methods.

onAddressRegistered

Syntax

onAddressRegistered(*event*)

Description

Fires when the address is registered by the session.

onAddressUnregistered

Syntax

onAddressUnregistered(*event*)

Description

Fires when the address is unregistered by the session.

onBuddyRegistered

Syntax

onBuddyRegistered(*event*)

Description

Fires when the buddy is registered by the session.

onBuddyUnregistered

Syntax

onBuddyUnregistered(*event*)

Description

Fires when the buddy is unregistered by the session.

onClosed

Syntax

onClosed(*event*)

Description

Fires when the session is closed.

onError

Syntax

onError(*event*)

Description

Fires when there is a session error.

onGroupRegistered

Syntax

onGroupRegistered(*event*)

Description

Fires when a group is registered by the session.

onGroupUnregistered

Syntax

onGroupUnregistered(*event*)

Description

Fires when a group is unregistered by the session.

onInfo

Syntax

onInfo(*event*)

Description

Fires when there is a session information event, such as the user is already logged in.

onOpened

Syntax

onOpened(*event*)

Description

Fires when the session is opened.

onUserRegistered

Syntax

onUserRegistered(*event*)

Description

Fires when the user is registered by the session.

onUserUnregistered

Syntax

onUserUnregistered(*event*)

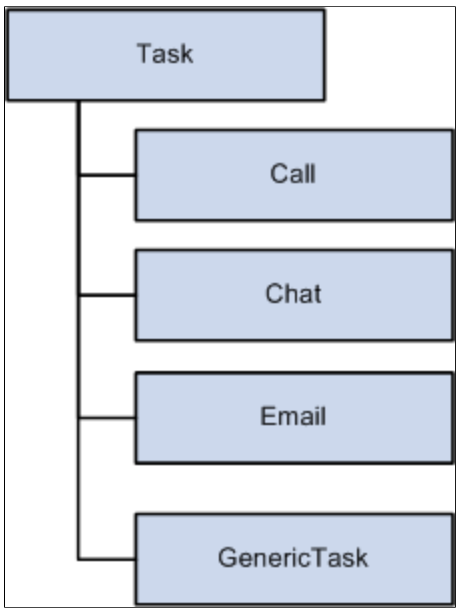
Description

Fires when the user gets unregistered by the session.

Task Class Hierarchy

The Task class can be extended by other subclasses.

The following flow chart shows the different subclasses of the Task class and how they interrelate.



Related Links

Call Class Constructor
Chat Class Constructor
Email Class Constructor
GenericAddress Class Constructor

Task Class Constructor

The following is the Task class constructor.

Task

Syntax

```
Task()
```

Description

Task is an abstract base class.

Parameters

None.

Returns

Type: Task object.

Task Class Fields

The following are the Task class fields.

caseid

Description

The associated case ID.

Type: string.

cost

Description

The associated cost.

Type: string.

customerid

Description

The associated customer ID.

Type: string.

group

Description

The group for which task is assigned.

Type: string.

id

Description

The task ID.

Type: string.

onStat

Description

Method that will be triggered when there is statistics published regarding this task.

Type: string.

priority

Description

The priority of the task.

Type: string.

type

Description

The type of the task.

Type: string, one of the following:

Value	*Description*
TYPE_A2ACHAT	The A2AChat type.
TYPE_CHAT	The chat type.
TYPE_CTI	The CTI type.
TYPE_EMAIL	The email type.
TYPE_GENERIC	The generic type.

Example

The TYPE_* fields are global fields, which may be accessed without instantiating an object. They are accessed as shown in this example:

```
Task.TYPE_GENERIC = "GENERIC";
```

Related Links

GLOBALS Class Fields

urlAbs

Description

The absolute URL that is useful in constructing URL for new window.

Type: urlAbs object.

urlRel

Description

The relative URL that is useful in constructing URL for new window.

Type: urlRel object.

TaskStatistics Class Constructor

The following is the TaskStatistics class constructor.

TaskStatistics

Syntax

```
TaskStatistics()
```

Description

Describes the task statistics information.

Returns

Task statistics object.

TaskStatistics Class Fields

The following are the TaskStatistics class fields.

data

Description

Key value pairs that includes all statistics.

Type: collection.

holdTime

Description

Time duration that the task is on hold.

Type: number

queueTime

Description

Time duration in the queue for this task.

Type: number

talkTime

Description

Time duration that the task is established.

Type: number

User Class Constructor

The User class extends the _User class.

See _User Class Constructor.

The following is the User class constructor.

User

Syntax

```
User(id, name, agentId, agentPassword)
```

Description

Describes the user/agent.

Parameters

Parameter	*Description*
id	The user ID.
name	The user name.
agentId	The agent ID.
agentPassword	The agent's password.

Returns

A User object.

User Class Fields

The User class inherits the following fields from the _User class:

- agentId
- caps
- id
- name
- presences
- ST_LOGGEDIN
- ST_LOGGEDOUT
- ST_NOTREADY
- ST_READY
- ST_UNKNOWN
- ST_WORKNOTREADY
- ST_WORKREADY
- states
- statistics
- statistics1

- statistics2

The following are the User class fields.

addresses

Description

The addresses associated with this agent.

Type: associative array.

agentPassword

Description

The password for this agent.

Type: string.

language

Description

The language for this agent.

Type: string.

registeredAddresses

Description

The registered addresses for this user.

Type: associative array.

statesUq

Description

States of the user for the queue server indexed by group number

Type: object.

User Class Methods

The following are the User class methods.

ctiBusyUq

Syntax

ctiBusyUq(*busy*, *subtractcost*, *task*)

Description

Flip ctiBusy flag, reduce cost of a CTI task from an agent's workload, or both. If the task is the same as the agent's assigned task, then the task's cost will be used. In case they do not match, the task type is checked and the default cost of the task is used to recalculate the workload.

Parameters

Parameter	***Description***
busy	The ctibusy flag that can have a value of 0 or 1.
subtractcost	Flag to reduce cost of the cti task from the agent's workload. It can be set to 0 or 1.
task	The task object.

Returns

Request number.

disableMedia

Syntax

disableMedia(*group*, *mediaType*, *extension*, *reason*)

Description

Disable a media channel for an agent on a group/queue.

Parameters

Parameter	***Description***
group	The group (queue) on which to disable the specified media type.
mediaType	The media type to disable.

Parameter	*Description*
extension	The extension on which to disable the specified media type.
reason	The reason for disabling this media type.

Returns

Request number.

enableMedia

Syntax

enableMedia(*group*, *mediaType*, *extension*, *reason*)

Description

Enable a media channel for an agent on a group/queue.

Parameters

Parameter	*Description*
group	The group (queue) on which to enable the specified media type.
mediaType	The media type to enable.
extension	The extension on which to enable the specified media type.
reason	The reason for enabling this media type.

Returns

Request number.

isMediaEnabled

Syntax

isMediaEnabled(*mediaType*, *group*)

Description

Determines if the specified media type is enabled.

Parameters

Parameter	*Description*
mediaType	Enter the media type.
group	Enter the group.

Returns

Returns true if the media type is enabled.

login

Syntax

login(*group*, *extension*, *reason*)

Description

Log the user in to a queue.

Parameters

Parameter	*Description*
group	The group to which the user will log in.
extension	The extension.
reason	The reason for the login.

Returns

Request number.

loginUq

Syntax

loginUq(*group*)

Description

Log a user in to a UQ server queue.

Parameters

Parameter	*Description*
group	The group (queue) to which the user will login.

Returns

Request number.

logout

Syntax

logout(*group*, *extension*, *reason*)

Description

Log the user out of the specified group (queue).

Parameters

Parameter	*Description*
group	The group from which the user will log out.
extension	The extension.
reason	The reason for the log out.

Returns

Request number.

logoutUq

Syntax

logoutUq(*group*)

Description

Log the user out from the UQ server queue.

Parameters

Parameter	*Description*
group	The group (queue) from which to log the user out.

Returns

Request number.

register

Syntax

register(*extension*, *reason*)

Description

Register an extension for the user.

Parameters

Parameter	*Description*
extension	The extension to be registered.
reason	The reason to register the extension.

Returns

Request number.

setNotReady

Syntax

setNotReady(*group*, *presence*, *extension*, *reason*)

Description

Set the user state as not ready.

Parameters

Parameter	*Description*
group	The group on which the user is changing state.
presence	The presence in the new state.
extension	The extension with which the user is changing state.
reason	The reason to set not ready.

Returns

Request number.

setPresence

Syntax

setPresence(*group*, *presence*, *extension*, *reason*)

Description

Set the presence for the current state.

Parameters

Parameter	*Description*
group	The group on which the user is changing presence.
presence	The presence for the state.
extension	The extension with which the user is changing presence.
reason	The reason to set the presence.

Returns

Request number.

setPresenceUq

Syntax

setPresenceUq(*group*, *presenceText*, *reason*)

Description

Set the Presence for UQ servers.

Parameters

Parameter	*Description*
group	The group (queue) on which the user is changing his or her presence.
presenceText	The new presence.
reason	The reason to set the new presence.

Returns

Request number.

setReady

Syntax

setReady(*group*, *presence*, *extension*, *reason*)

Description

Set the user state as ready.

Parameters

Parameter	*Description*
group	The group on which the user is changing state.
presence	The presence for the state.
extension	The extension with which the user is changing state.

Parameter	*Description*
reason	The reason to set the state.

Returns

Request number.

setWorkNotReady

Syntax

setWorkNotReady(*group*, *presence*, *extension*, *reason*)

Description

Set the user state as work not ready.

Parameters

Parameter	*Description*
group	The group on which the user is changing state.
presence	The presence for the state.
extension	The extension with which the user is changing state.
reason	The reason to set work not ready.

Returns

Request number.

setWorkReady

Syntax

setWorkReady(*group*, *presence*, *extension*, *reason*)

Description

Set the user state as work ready.

Parameters

Parameter	*Description*
group	The group on which the user is changing state.
presence	The presence for the state.
extension	The extension with which the user is changing state.
reason	The reason to set work ready.

Returns

Request number.

unregister

Syntax

unregister(*extension*, *reason*)

Description

Unregister an extension for the user.

Parameters

Parameter	*Description*
extension	The extension to be unregistered.
reason	The reason to unregister the extension.

Returns

Request number.

User Class Callback Event Methods

The following are the User class callback event methods.

onCapabilitiesChanged

Syntax

onCapabilitiesChanged(*event*)

Description

Fires when capabilities changed.

onCtiBusy

Syntax

onCtiBusy(*event*)

Description

Fires when the UQ server is notified that the CTI is busy.

onCTIBUSYUq

Syntax

onCTIBUSYUq(*event*)

Description

Fires when the UQ server is notified that the CTI is busy.

onCtiClear

Syntax

onCtiClear(*event*)

Description

Fires when the UQ server is notified that the CTI is clear.

onDropped

Syntax

onDropped(*event*)

Description

Fires when a user is dropped.

onError

Syntax

onError(*event*)

Description

Fires when an error occurs.

onInfo

Syntax

onInfo(*event*)

Description

Fires when an information event occurs. Currently only used on UQ server.

onLoggedIn

Syntax

onLoggedIn(*event*)

Description

Fires when the user is logged in.

onLoggedOut

Syntax

onLoggedOut(*event*)

Description

Fires when the user is logged out.

onLoggingIn

Syntax

onLoggingIn(*event*)

Description

Fires when the user is logging in.

onLoggingOut

Syntax

onLoggingOut(*event*)

Description

Fires when the user is logging out.

onMediaDisabled

Syntax

onMediaDisabled(*event*)

Description

Fires when a media type is disabled.

onMediaEnabled

Syntax

onMediaEnabled(*event*)

Description

Fires when a media type is enabled.

onNotReady

Syntax

onNotReady(*event*)

Description

Fires when a user state changes to not ready.

onPresenceChanged

Syntax

onPresenceChanged(*event*)

Description

Fires when the user's presence is changed.

onReady

Syntax

onReady(*event*)

Description

Fires when the user's state changes to ready.

onRegistered

Syntax

onRegistered(*event*)

Description

Fires when the address is registered.

onRegistering

Syntax

onRegistering(*event*)

Description

Fires when registering the address.

onSettingNotReady

Syntax

onSettingNotReady(*event*)

Description

Fires when setting the user's state to not ready.

onSettingPresence

Syntax

(*event*)

onSettingPresence

Description

Fires when setting presence for the current state.

onSettingReady

Syntax

onSettingReady(*event*)

Description

Fires when setting the user's state to ready.

onSettingWorkNotReady

Syntax

onSettingWorkNotReady(*event*)

Description

Fires when setting the user's state to work not ready.

onSettingWorkReady

Syntax

onSettingWorkReady(*event*)

Description

Fires when setting the user's state to work ready.

onStat

Syntax

onStat(*event*)

Description

Fires when statistics are received.

onStat1

Syntax

onStat1(*event*)

Description

Fires when statistics1 is received.

onStat2

Syntax

onStat2(*event*)

Description

Fires when statistics2 received.

onUnknown

Syntax

onUnknown(*event*)

Description

Fires when the user's state changes to unknown.

onUnregistered

Syntax

onUnregistered(*event*)

Description

Fires when the address gets unregistered.

onUnregistering

Syntax

onUnregistering(*event*)

Description

Fires when unregistering the address.

onWorkNotReady

Syntax

onWorkNotReady(*event*)

Description

Fires when the user's state changes to work not ready.

onWorkReady

Syntax

onWorkReady(*event*)

Description

Fires when the user's state changes to work ready.

UserCaps Class Constructor

The following is the UserCaps class constructor.

UserCaps

Syntax

UserCaps(*strCaps*)

Description

The UserCaps object describes the user capabilities.

Parameters

Parameter	*Description*
strCaps	The user capabilities.

Returns

A UserCaps object.

UserCaps Class Fields

The following are the UserCaps class fields

canHandleChat

Description

Chat task handling capability of the user.

Type: boolean.

canHandleEmail

Description

Email task handling capability of the user.

Type: boolean.

canHandleGeneric

Description

Generic task handling capability of the user.

Type: boolean.

canHandleVoice

Description

Voice task handling capability of the user.

Type: boolean.

canLogin

Description

Login capability of the user.

Type: boolean.

canLogout

Description

Logout capability of the user.

Type: boolean.

canRefreshState

Description

The refreshState capability of the user.

Type: boolean.

canSetNotReady

Description

The setNotReady capability of the user.

Type: boolean.

canSetPresence

Description

The setPresence capability of the user.

Type: boolean.

canSetReady

Description

The setReady capability of the user.

Type: boolean.

canSetWorkNotReady

Description

The setWorkNotReady capability of the user.

Type: boolean.

canSetWorkReady

Description

The setWorkReady capability of the user.

Type: boolean.

UserData Class Constructor

The following is the UserData class constructor.

UserData

Syntax

```
UserData()
```

Description

The UserData object describes the key-value pairs of attached user data.

Parameters

None.

Returns

A UserData object.

UserData Class Fields

The following are the UserData class fields constants.

UserData Class Field Constants

Description

UserData class uses the following constants.

Value	*Description*
COMPONENT	The component.
DESCR	The description
ICSCRIPTPROGRAMNAME	The IC script program name.
ICTYPE	The IC type.
MARKET	The market.
MENU	The menu.
REFERENCEID	The reference ID
TARGET	The target.
URLABS	The absolute URL.
URLREL	The relative URL.

UserData Class Method

The following is the UserData class method.

addKeyValue

Syntax

addKeyValue(*key, value*)

Description

Add the key-value pair to the UserData object.

Parameters

Parameter	*Description*
key	The key.
value	The value.

Returns

None.

UserStatistics1 Class Constructor

The following is the UseStatistics1 class constructor.

UserStatistics1

Syntax

UserStatistics1()

Description

UQ User statistics sent on user refresh 1.

Parameters

None.

Returns

A UserStatistics1 object.

UserStatistics1 Class Fields

The following are the UserStatistics1 class fields.

availableCost

Description

The amount of unused work.

ctiBusy

Description

Flag indicating that a user is engaged on CTI task.

currentQueue

Description

Current queue/group the User last logged into.

Type: string.

mostRecentTaskData

Description

Most recent task's data.

Type: string.

mostRecentTaskId

Description

Most recent task's ID.

Type: string.

numTaskAccepted

Description

The number of tasks accepted by the user.

Type: string.

numTasksDone

Description

The number of tasks done by the user.

numTasksUnassigned

Description

The number of tasks unassigned by the user.

Type: string.

presenceText

Description

The presence text for the user.

Type: string.

reasonFlag

Description

The reason flag for this event.

Type: string.

state

Description

Current state of the user.

Type: string.

timeInCurrentState

Description

Time in current state for the user.

Type: string.

timeSinceLoggedIn

Description

The time since the user logged in.

Type: string.

UserStatistics2 Class Constructor

The following is the UserStatistics2 class constructor.

UserStatistics2

Syntax

```
UserStatistics2()
```

Description

UQ User statistics sent on user refresh 2 event.

Parameters

None.

Returns

A UserStatistics2 object.

UserStatistics2 Class Fields

The following are the UserStatistics2 class fields.

currentQueue

Description

The current queue or group for the user.

Type: string.

timeIdle

Description

Idle time for the user.

Type: string.

timeInCurrentState

Description

The time in the current state for the user.

Type: string.

timeNotReady

Description

The time during which the user was in the not ready state.

Type: string.

timeSinceLogin

Description

The time since the user logged in.

Type: string.

totalTimeAvailable

Description

The total time during which the user has been in the available state.

Type: string.

totalTimeUnavailable

Description

The total time during which the user has been in the unavailable state.

Chapter 13

Configuring the Email Channel

Understanding the Email Channel

The email channel requires configuration in Integration Broker in addition to settings in PeopleSoft MultiChannel Framework configuration pages. For information on the configuration of email channel in the PeopleSoft Integration Broker see, Configuring Email Channel in PeopleSoft Integration Broker.

PeopleSoft MultiChannel Framework (MCF) provides a framework for:

- Fetching emails from a mail server.
- Storing and retrieving email parts in a database.
- Managing attachments.
- Queueing of emails by the queue server.

Application developers use the PeopleCode application package PT_MCF_MAIL to develop PeopleSoft Application Engine programs and PeopleSoft Pure Internet Architecture components for email processing.

See “Understanding PeopleSoft MultiChannel Framework Mail Classes” (PeopleCode API Reference).

PeopleSoft MultiChannel Framework email channel features include:

- The GETMAILTARGET target connector.
- A PeopleCode application package, PT_MCF_MAIL, to retrieve, store, and delete email.
- Support for both Post Office Protocol 3 (POP3) and Internet Message Access Protocol 4 (IMAP4) email protocols.
- Support for HTTPS through a Microsoft Azure connection. See “Understanding OAuth 2.0” (Security Administration).
- Support for SSL connections.
- Support for Transport Layer Security (TLS) connections.
- Support for NT LAN Manager (NTLM) and Simple Authentication and Security Layer (SASL) authentication mechanisms.
- Support for email attachments, including:
    - An attachments repository running on the same web server as PeopleSoft Integration Broker.
    - URL access to email attachments.

  - Storage and retrieval of email attachments from database.
  - A relative addressing scheme to enable flexibility of repository location.
  - User- and role-based security to control access to attachments.

- The option to access email headers, attachments, and body from the mail server.
- Support for multiple message size thresholds to control distribution of email between the database, the attachment repository, and the mail server.
- Support for UTF8 Unicode as supported by the Sun Java Runtime Environment (JRE).

  See http://java.sun.com.
- Time zone conversions to support global service-level agreements.
- Sample email application pages demonstrating the functionality of the PT_MCF_MAIL application package.
- Ability to enable logging for MCFOutbound email by setting SMTP trace in psappsrv.cfg. The log file is created in the log directory defined in psappsrv.cfg.

  See “SMTPTrace” (System and Server Administration).

Working With Emails using PeopleSoft Multichannel Framework

This section describes some factors to consider when handling email using PeopleSoft MultiChannel Framework.

POP3 Versus IMAP4

Most mail servers support simultaneous client connections through both POP3 and IMAP4. Using IMAP4 for your MCF email connection provides significant benefit over using POP3. IMAP4 allows for the use of email folders, which allows malformed emails to be set aside for separate processing. Quarantining malformed emails allows system administrators to remove or correct invalid emails without interrupting normal processing.

Note: Using IMAP4 for your MCF email connection does not preclude the use of POP3 for any other client connections to your mail server.

Note: POP3 mail servers are typically read-only, which can create issues if emails are required to be deleted after being read.

Time Zone Offsets

Set the values of the email sent time zone offset (in minutes) and the receive time zone offset (in minutes) whenever possible. The default value is 800, which indicates time zone information is not available. When available, the values range from +720 to −720.

Connector Determination of Email Size

The connector sometimes cannot determine the size of the message. In such cases the size is set to 0 and an error message is written to the gateway error log.

Malformed Emails

Emails with syntax that does not comply to the latest standard definitions may not be processed successfully by MCF. MCF will return as much as possible of noncompliant emails, however, at times, invalid parts will be ignored. For example, email addresses with invalid characters will not be displayed, such as two @ symbols. Also, if the header information is not valid, such as a malformed content type, the associated email part may not appear.

A common example of a malformed email is one that contains 8-bit encoded information in a header, such as the To, Cc or Subject field, or in an attachment file name. According to the standard definitions, this information must be encoded using 7-bit encoding, as described in RFC 2045 and others. This encoding is usually done by the email sender.

Note: The PeopleSoft email reader only supports email messages that follow the SMTP specification. Many email clients and mail servers have their own proprietary formats; therefore, be aware that PeopleSoft MCF will interpret these emails according to the RFC specification. For example, some mail servers and clients allow 8-bit encoded data in email headers, which the PeopleSoft email reader does not allow.

Domain Validation

Application developers can use the method IsDomainNameValid to validate email address domains.

Note: The number of retries for domain validation is set by the parameter SMTPDNSTimeoutRetries in psappsrv.cfg.

See “SMTP Settings” (System and Server Administration), “IsDomainNameValid” (PeopleCode API Reference).

Email Address Syntax Validation

The ValidateAddress method checks the email address syntax against RFC standards.

See “ValidateAddress” (PeopleCode API Reference).

Character Sets

If you wish to attach files which are named using characters outside of the character set recognized by your application server or process scheduler operating system, you should update the locale or regional settings of your operating system to allow those characters to be recognized.

SSL/TLS Connections

Certificates are an important component in SSL communication for authentication of the parties involved. In this case, one party is the email server and the other party is PeopleSoft applications.

To configure SSL/TLS for inbound emails:

1. Add the property MCF_UseTLS to the GETMAILTARGET target connector of your node, for example, MCF_GETMAIL, with a value of *Y*. This enables a TLS connection to the server, requiring the use of STARTTLS.

   Note: For a direct SSL/TLS connection, use the MCF_UseSSL property instead of MCF_UseTLS.

2. Obtain the email server certificate and import that in the pskey key store using *pskeymanager* utility.

3. Update the Integration Broker Gateway properties file with the path to this key store and the key store password, as described here:

   - `secureFileKeystorePath`: Enter the full path and file name of the gateway keystore file, which is located in the web server directory structure. For example: <PIA_HOME>\webserv \<DOMAIN>\piaconfig\keystore.

   - `secureFileKeystorePasswd`: Enter the keystore password. This password must be encrypted.

For the communication to work, you must also import the trusted CA root certificates of the CA which signed the email server certificate into the Integration Broker Gateway truststore. If a CA issues a chain of certificates, you must add the entire certificate chain from the signer of the email server certificate to the Root CA certificate.

By default, the gateway truststore may be the pskey peoplesoft keystore stored on the web server or it may be the cacerts keystore for your Java installation. The path to the gateway truststore can be defined by the SSL_KEY_STORE_PATH variable in the setenv.cmd file.

See “Implementing WebLogic SSL Keys and Certificates” (System and Server Administration)

Connectivity information for outgoing emails can be configured in psappsrv.cfg or provided by the application using application package PT_MCF_MAIL.

See “Understanding PeopleSoft MultiChannel Framework Mail Classes” (PeopleCode API Reference)

NTLM and SASL Authentication Mechanisms

MCF supports the NT LAN Manager (NTLM) and the Simple Authentication and Security Layer (SASL) authentication mechanisms for outgoing emails. To enable these authentication mechanisms, you must add and configure the NTLM and the SASL parameters in psappsrv.cfg and psprcs.cfg for a primary server and if required, for a backup server. MCF does not set default values for the NTLM and the SASL parameters in the configuration files.

Note: NTLM and SASL log will be available in the SMTP log because NTLM and SASL are part of SMTP sessions. That is, separate log files are not created for NTLM and SASL.

See “SMTP Settings” (System and Server Administration).

Virus Scanning

You can enable virus scanning for inbound MCF email attachments and outbound file attachments.

Using IDDA Logging

Use the PeopleSoft Instrumented Development Diagnostic Aid (IDDA) logger to gather information on email attachment viewing.

See “Enabling IDDA Logging” (System and Server Administration).

Configuring Email Channel in PeopleSoft Integration Broker

Before you configure the email channel for PeopleSoft Multichannel Framework, you must successfully install and configure the PeopleSoft Integration Broker.

See “Installing PeopleSoft Integration Broker” (Integration Broker Administration).

For the email channel to work, you must configure the following in PeopleSoft Integration Broker:

- Define a gateway that will be used with the MCF_GETMAIL connector node.

  See “Defining Integration Gateways and Loading Connectors” (Integration Broker Administration).

- Define a connector for the MCF_GETMAIL node and specify the gateway.

  See “Specifying Gateways and Connectors” (Integration Broker Administration).

- Configure the GETMAILTARGET connector properties.

  See “Editing Connector Properties” (Integration Broker Administration).

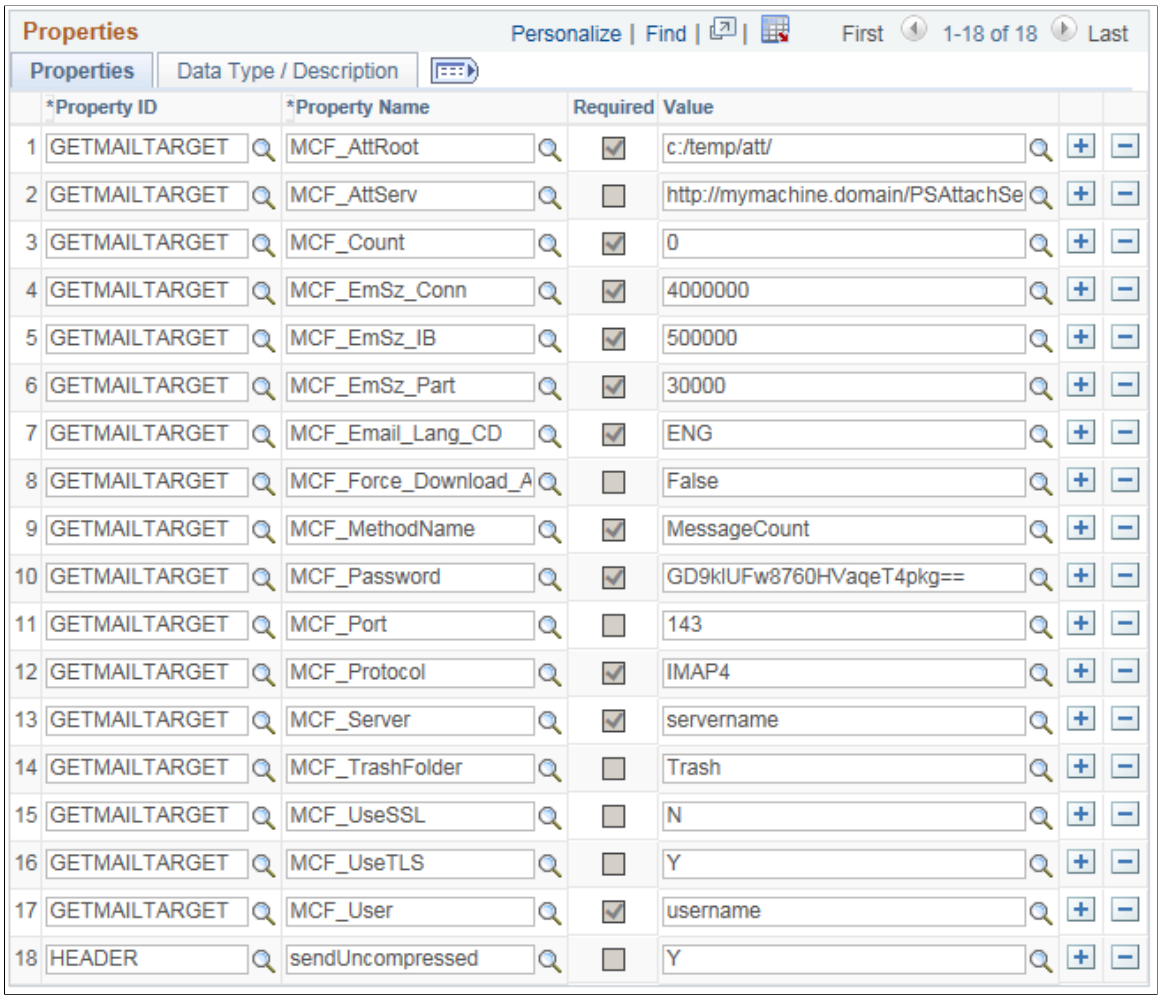
Note: To use MCF email in UNIX and LINUX systems, add `-Djava.awt.headless=true` in the JAVA_OPTIONS variable of the setEnv.sh file.
Use this option only if the operating system is set up as *headless* (no graphics option).

Configuring GETMAILTARGET Connector Properties

Access the Connector Properties page using the following navigation path:

PeopleTools > **Integration Broker** > **Integration Setup** > **Nodes**. The Node Definition search page opens. Open a Node Definition and click the Connectors tab to open the Connectors page.

This example illustrates the Data Type/Properties tab on the Connectors page where you can set the Property Name of a Connector ID.



Properties — Personalize | Find | First 1-18 of 18 Last

Properties | Data Type / Description

	*Property ID	*Property Name	Required	Value
1	GETMAILTARGET	MCF_AttRoot	☑	c:/temp/att/
2	GETMAILTARGET	MCF_AttServ	☐	http://mymachine.domain/PSAttachSe
3	GETMAILTARGET	MCF_Count	☑	0
4	GETMAILTARGET	MCF_EmSz_Conn	☑	4000000
5	GETMAILTARGET	MCF_EmSz_IB	☑	500000
6	GETMAILTARGET	MCF_EmSz_Part	☑	30000
7	GETMAILTARGET	MCF_Email_Lang_CD	☑	ENG
8	GETMAILTARGET	MCF_Force_Download_A	☐	False
9	GETMAILTARGET	MCF_MethodName	☑	MessageCount
10	GETMAILTARGET	MCF_Password	☑	GD9klUFw8760HVaqeT4pkg==
11	GETMAILTARGET	MCF_Port	☐	143
12	GETMAILTARGET	MCF_Protocol	☑	IMAP4
13	GETMAILTARGET	MCF_Server	☑	servername
14	GETMAILTARGET	MCF_TrashFolder	☐	Trash
15	GETMAILTARGET	MCF_UseSSL	☐	N
16	GETMAILTARGET	MCF_UseTLS	☐	Y
17	GETMAILTARGET	MCF_User	☑	username
18	HEADER	sendUncompressed	☐	Y

Configure GETMAILTARGET connector properties on the MCF_GETMAIL node. You can also set threshold values for email routed to the attachment repository.

Note: The threshold parameters MCF_EmSz_IB, MCF_EmSz_Part, and MCF_FetchSize, are not used. If the size of the email reads exceed the resources available, tuning needs to be done either by reducing the number of emails read at each time, or by limiting the size of emails, particularly attachments, at the mail server level.

The following table lists node properties:

Property Name	*Default Value*	*Description*
MCF_AttRoot	*c:/temp/att*	Defines the location of the attachment repository on the gateway. For example, if the attachment root is c:/temp/att, then attachments for emails addressed to a mail box *support* are downloaded to c:/temp/att/<mail server host name>/<mailbox>. **Note:** If the attachments are to be stored by a gateway that is remote to the webserver specified in the MCF_AttServ property, be sure that the file system permissions are set so that the directory may be accessed by both. The user permissions for the directory specified by MCF_AttRoot should be set such that the node (such as the local node) that will retrieve the attachments has read permissions. **Note:** The user name or mailbox portion of the attachment root path may include encoded characters. All email attachments, and email parts that are not of the content type specified for the MCF_ContentTypes property, are always written to the attachment repository. A relative URL is returned to the application that allows the application to access the attachment. The attachment retrieval depends on the file system of the operating system. For Windows NTFS/FAT 16 file system, the folder length or maximum characters of the file extension is 256/512. For Unix and Solaris UFS file system, the length is 256. Attachment security is managed at a user and role level using the PT_MCF_EMAIL application package. Users can configure attachment folder paths using metastrings to enable multiple folders based on several variables: • *%OPRID%:* PeopleSoft user ID. • *%DBNAME%:* database name. • *%CURRDATE%:* current date. • *%CURRHOUR%:* current hour. MCF_AttRoot is only used for file-based attachments. Database attachments ignore this parameter. When using the *HTTPS* email protocol, this directory is used as a temporary staging

Property Name	Default Value	Description
		area even in the case of database attachment storage
MCF_AttServ	*http://<machine_name>.<domain_name>/PSAttachServlet/ps/*	Defines the location of the MCF attachment repository servlet, located on the gateway web server. Attachment relative URLs are appended to this address to create a fully qualified URL to reference an attachment in the repository. **Note:** When you are setting a remote gateway to access email attachments stored on the remote machine, the database that the gateway/psattachservlet accesses should have the permissions to view the attachments. **Note:** If your web server is installed with a PeopleSoft Pure Internet Architecture site name other than *ps,* include the site name in the specified path.
MCF_ClientCertAlias	ClientCert	Provide the alias of the client certificate. **Note:** The client certificate must be imported in the pskey keystore. **Note:** The MCF_ClientCertAlias property must be set only if the Email Server is set for client authentication, that is, the Email Server is set to request the client's certificate for authentication. Additionally, you must ensure that the trusted certificates and the certificate chain (if any) of the client certificate are loaded in the Email Server. See "Installing Integration Gateway-Based Digital Certificates" (Integration Broker Administration).

Property Name	*Default Value*	*Description*
MCF_ClientCertPass	password	Enter the password of the private key that you provided when you created the client certificate. **Note:** The alias and the password must match the alias and password of the imported client certificate in the pskey keystore. **Note:** The MCF_ClientCertPass property must be set only if the Email Server is set to request the client's certificate for authentication. Additionally, you must ensure that the trusted certificates and the certificate chain (if any) of the client certificate are loaded in the Email Server.
MCF_ContentTypes	*text/plain*	Lists those email content types (separated by commas) that are to be accessible by using the Text property of the MCFBodyPart class. The content type, text/plain, is always an implicit member of this list. Content types that are not members of this list will be stored in the attachment repository and, as a result, instead be accessible by using the AttachmentURL property of the MCFBodyPart class. For example, you may specify text/html or text/xml as a member of this list, and, as a result, it will be accessible through the Text property of the MCFBodyPart class. Since a binary content type is automatically converted to base64-encoded text by the email server, if it is specified as a member of the content types list, it will be stored in the PeopleSoft database in this form. So, do not specify binary content types in this list unless the application is prepared to convert this base64-encoded text back to its original binary form as needed. **Note:** If the value of the MCF_Force_Download_Attachments property is True, then the value of the MCF_ContentTypes property is entirely ignored.
MCF_Count	*0*	Defines the default number of emails retrieved from the mail server unless otherwise specified in the SyncRequest. The PeopleSoft MultiChannel Framework email application package always specifies the number of emails to be retrieved and does not reference this property.

Property Name	Default Value	Description
MCF_DBLengthType	*N*	Specifies the double byte length type of the database. The default value is N, which denotes that the type is Unicode. The other supported types are: • *M* : SJIS • *A* : Ansi • *D* : DB2EBCDIC
MCF_DisableAuthPlain	*N*	A value of *Y* indicates that MCFInboundEmail is prevented from attempting to connect to a mail server using PLAIN authentication. The default value is *N*. This property is not a required parameter.
MCF_EmSz_Conn	*4000000*	Defines the connector threshold, in bytes. This parameter ensures that emails retrieved from a POP3 mail server do not exceed the available memory on the gateway server. Email content retrieved from IMAP4 servers can be streamed to a file on the attachment repository, but content retrieved from POP3 servers must first be read into memory before being written to a file. Therefore, the MCF_EmSz_Conn threshold can be set very high for IMAP4 mail servers. PeopleSoft MultiChannel Framework does not process an email if its size is greater than this value. The status returned to the application is *1*. The triggering email is neither downloaded nor deleted from the mail server. Only the email header is returned to the application. When using the *HTTPS* email protocol, determines the quantity of emails that can be retrieved in a request. See “Understanding PeopleSoft MultiChannel Framework Mail Classes” (PeopleCode API Reference).

Property Name	*Default Value*	*Description*
MCF_EmSz_IB	*500000*	Defines the PeopleSoft Integration Broker threshold, in bytes. This is the maximum size message that the GETMAILTARGET connector can send through PeopleSoft Integration Broker. The purpose of this threshold is to ensure that retrieved emails do not exceed the available memory on the gateway server and that the server running the application that requested the emails. A PeopleSoft Integration Broker message can contain one or more emails, depending on the number of emails the application retrieves per message. As soon as this threshold is reached, the remainder of the triggering email is written to the repository, and no more emails are retrieved. Emails fetched prior to the triggering email are returned normally. The triggering email is returned with a return status of *2,* indicating that some of its parts were written to the repository. See “Understanding PeopleSoft MultiChannel Framework Mail Classes” (PeopleCode API Reference).
MCF_EmSz_Part	*30000*	Defines the text part threshold, in bytes. If any text part of an email exceeds this value, the part is routed to the attachment repository. This ensures that excessively large objects of text are not written to the database. For databases with limitations on the maximum size of rows or long text field lengths, this value must be lower than that limit to avoid SQL errors when saving emails. **Note:** The value of MCF_EmSz_Part divided by 3 is the character threshold for moving email body text into the attachment repository for HTTPS email. You can adjust this threshold according to your database needs and characters sets you use.

Property Name	Default Value	Description
MCF_Email_Lang_CD	*ENG*	Defines the expected language of emails downloaded by this node. This code is included in the email header returned to the application. Configure a node for each language you expect to handle, because Simple Mail Transfer Protocol (SMTP) mail servers and clients do not guarantee correct identification of the email's language when creating an email.
MCF_FetchSize	*819200*	Determines the number of bytes retrieved for an IMAP email in a single fetch. It also controls the size of the buffer (in bytes) used to write email attachments to the repository. Setting MCF_FetchSize to a higher number uses more memory, but it may also increase the speed of download for larger emails and attachments.
MCF_Force_Download_Attachments	*False*	A value of True indicates that all email parts, including text/plain, irrespective of their size or content type, are to be downloaded to the attachment repository. This property may be used to facilitate the viewing of an email body containing multi-byte characters. A value of False indicates that email parts are evaluated individually on the basis of their size, content type and the value of the MCF_ ContentTypes property. **Note:** If the value of the MCF_Force_ Download_Attachments property is True, then the value of the MCF_ContentTypes property is entirely ignored.
MCF_MethodName	*MessageCount*	Defines the methods associated with the application class package. The defined method is the default method used unless otherwise specified in the SyncRequest. The email application package always specifies the method to be used and does not reference this property. See “Understanding PeopleSoft MultiChannel Framework Mail Classes” (PeopleCode API Reference).

Property Name	***Default Value***	***Description***
MCF_Password	None	Defines the default password for the mailbox unless otherwise specified in the SyncRequest. This property is not applicable to the HTTPS connection protocol. See “Understanding PeopleSoft MultiChannel Framework Mail Classes” (PeopleCode API Reference).
MCF_Port	*143*	Defines the mail server port to be used. By default, POP3 servers use port 110 and IMAP4 servers use port 143. Confirm the port number with your system administrator and set it accordingly here. This property is not applicable to the HTTPS connection protocol.
MCF_Protocol	*IMAP4*	Defines the protocol used by the connector to access emails on the mail server. Supported values are *POP3* , *IMAP4* and *HTTPS*. Configuration of *HTTPS* will also require OAuth 2.0. See “Understanding OAuth 2.0” (Security Administration)
MCF_Quarantine	*Quarantine*	Defines a quarantine folder for email meeting the following criteria: • Connector size overflow. • Unsupported encoding. • Unknown Java exception. • JavaMail content error. • No attachment repository. • Mail parse exception. The quarantine folder is created for each user account and must be managed by each email user, not by the system administrators. **Note:** Folders are supported only on IMAP4 mail servers. This property is not applicable to the HTTPS connection protocol.

Property Name	*Default Value*	*Description*
MCF_RepositoryType	*FILE*	Specifies the location for storing email attachments. Email attachments can be stored either in the attachment repository or in the database. The default value is FILE. If you select DATABASE, the email attachments are stored in the database. Also, when you select DATABASE, the value in the MCF_AttRoot property is ignored. If you select FILE, the email attachments are stored in the attachment repository.
MCF_Server	None	Defines the fully qualified host name of the mail server. This is the default fully qualified host name of the mail server, unless otherwise specified in the SyncRequest. For example, bigserver.example.com. When using the *HTTPS* email protocol, MCF_Server will refer to the application name. See “Understanding PeopleSoft MultiChannel Framework Mail Classes” (PeopleCode API Reference).
MCF_SMIMEErrorHandling	1	Defines how signed S/MIME emails are treated if the signature cannot be verified. If the digital signature of retrieved email cannot be verified, email retrieval APIs respond in the following ways: • By default, or if the parameter is set to 1, an error is returned and logged. The API does not return unverified data, but returns data outside of the signature block, such as header information, if available. • If the parameter is set to 2, no error is returned. The unverified data is returned and the subject line of the email is prefixed with "[UNVERIFIED]". • If the parameter is set to 3, an error is logged but not returned. Also, all of the data, including unverified data is returned.

Property Name	*Default Value*	*Description*
MCF_TrashFolder	*Trash*	Determines that the deleted mails are transferred to a trash folder and not permanently expunged from the system. You need to provide a mail server that supports IMAP4 folders. This feature does not support POP3 protocol because local folders are not supported.
MCF_User	None	Defines the user name for the email account (mailbox) being accessed. This is the default user name used unless otherwise specified in the SyncRequest. For example, if emails are addressed to support@example.com, the user name is *support.* This property will be used to hold the application name from the configuration page.
MCF_UseSSL	*N*	A value of *Y* indicates that SSL/TLS must be used when communicating to the mail server. The SSL settings will be loaded from the GETMAILTARGET connector, including client certificate settings. For information on SSL connections, see Working With Emails using PeopleSoft Multichannel Framework, SSL Connections. This property is not applicable to the HTTPS connection protocol.
MCF_UseTLS	*N*	If the property value is set to *Y*, then it requires an SSL/TLS connection for the inbound email system using STARTTLS. This property is not applicable to the HTTPS connection protocol. For information on TLS connections, see Working With Emails using PeopleSoft Multichannel Framework, Transport Layer Security Configuration.

Note: The POP3 and IMAP4 email protocols calculate message size differently. POP3 does not include header size, but IMAP4 does. As a result, message sizes affecting thresholds behave differently depending on the protocol used.
Multipurpose Internet Mail Extensions (MIME), is an Internet standard for representing multipart and multimedia data in email so that they can be exchanged between different email systems. The parts may be nested. PeopleSoft embeds the Sun JavaMail API to implement support for the MIME standard. However, all parts of an email are represented in PeopleSoft as level 1 rowsets, regardless of whatever hierarchy existed in the original email. Email clients determine the MIME format of the emails sent from them, so identical email content sent from different email clients may have a different MIME structures.

Setting Java Truststore Properties for the GETMAILTARGET Connector

You may set the Java truststore path and password for the GETMAILTARGET connector.

Use these properties to set the Java truststore path and password in the IntegrationGateway.properties file:

```
secureTruststorePath=<fileLocation>

secureTruststorePasswd=<password>
```

Use the supplied encryption utility to provide an encrypted password for these entries.

Note: If the truststore path is not specified, it will be undefined for email retrieval, and trusted root certificates may not be honored.

Related Links

“Understanding Managing Integration Gateways” (Integration Broker Administration)

Setting Up MCF Email Using Azure

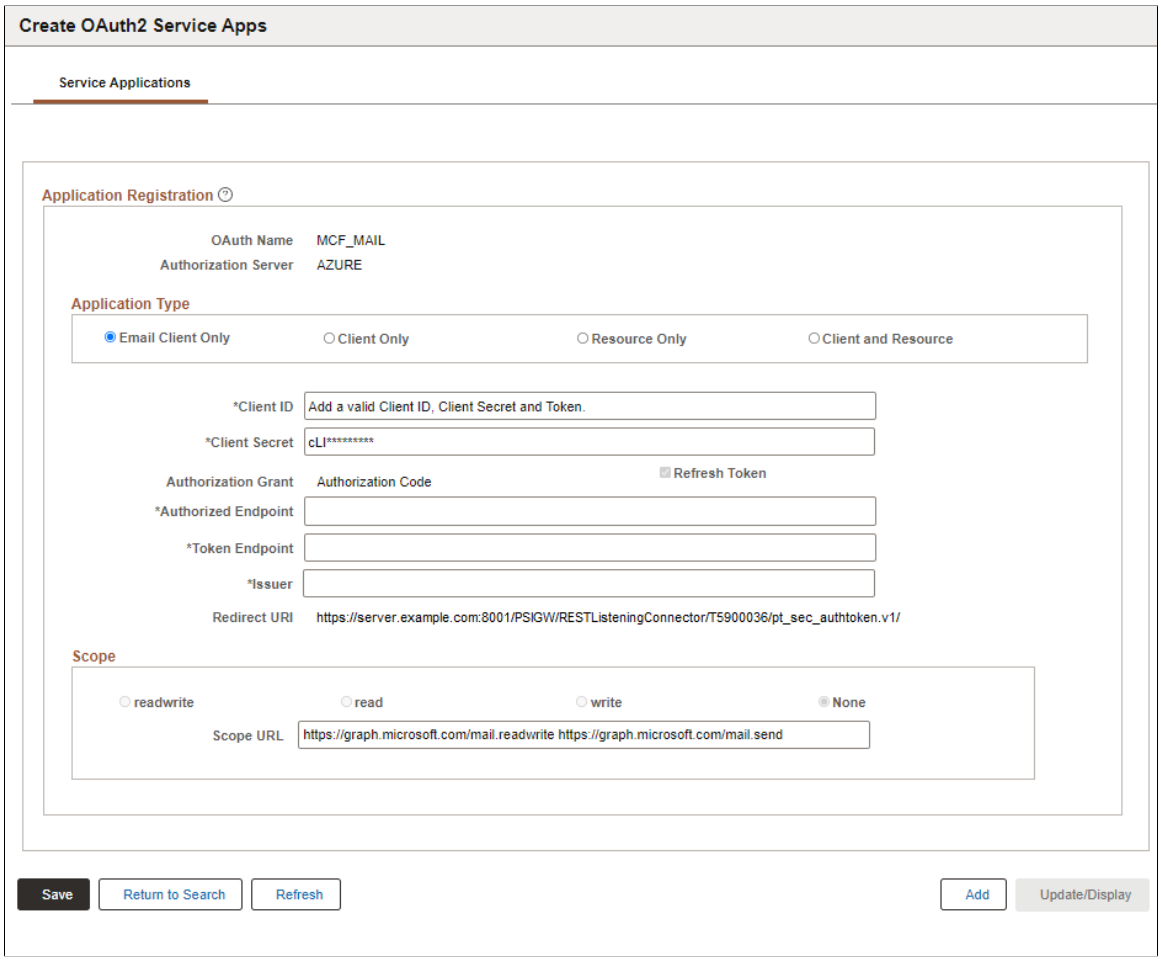
You can set up MCF inbound and outbound emails using OAuth2 authentication through Azure using the Create OAuth2 Service Apps page.

To create your own OAuth2 service application or open an existing application, open the Create OAuth2 Service Apps page using this navigation path:

Navigation:

PeopleTools > **Security** > **OAuth2 Administration** > **Create OAuth2 Service Apps**.

This example illustrates the fields and controls on the Example to set up MCF Emails using OAuth2 Authentication with Azure.



Example

This example shows the suggested values to be used for MCF emails:

Field or Control	***Description***
Application Type	Select Email Client.
Client ID	Enter Azure application client ID.
Client Secret	Enter Azure application secret.
Authorized Endpoint	https://login.microsoftonline.com/<*tenant ID*>/*oauth2/v2.0/authorize*

Field or Control	*Description*
Token Endpoint	https://login.microsoftonline.com/<*tenant ID*>/*oauth2/v2.0/token*
Issuer	https://login.microsoftonline.com
Scope	For email configurations, scope is set to *None* and grayed-out.
Scope URL	For email configurations, this defaults to https://graph.microsoft.com/mail.readwrite.
Refresh Token	This option is selected and grayed-out for email configurations.

For more information on OAuth2 support for non-IDCS service providers, see “Understanding OAuth 2.0” (Security Administration).

There are several SMTP parameters that you must modify for OAuth configuration. See “SMTP Settings” (System and Server Administration).

For more information on certificates for SSL communication, see How To Configure SMTP SSL on Application Servers and Process Schedulers on My Oracle Support (Doc ID 1612188.1). For more information on setting up client certificate authentication, see How to Setup SMTP Client Certificate Authentication in PeopleSoft (Doc ID 2876697.1).

Enabling Virus Scanning

Virus scanning can be enabled for inbound IMAP and POP3 MCF Email, and AddAttachment, by configuring the virusscan.xml file for your virus scan engine.

See “Enabling Virus Scanning for Web Servers” (Security Administration)

Demonstrating the Email Channel

To demonstrate the email channel, use the Email Sample Pages (MCFEM_DEMOERMS_CMP) component.

The PeopleSoft system provides email sample pages to demonstrate the functionality of the PT_MCF_MAIL application package.

Note: The email sample pages are not intended for any purpose other than demonstration and troubleshooting. They should not be used in production.

The email sample pages can be also used to test the configuration of your integration gateway and MCF_GETMAIL node.

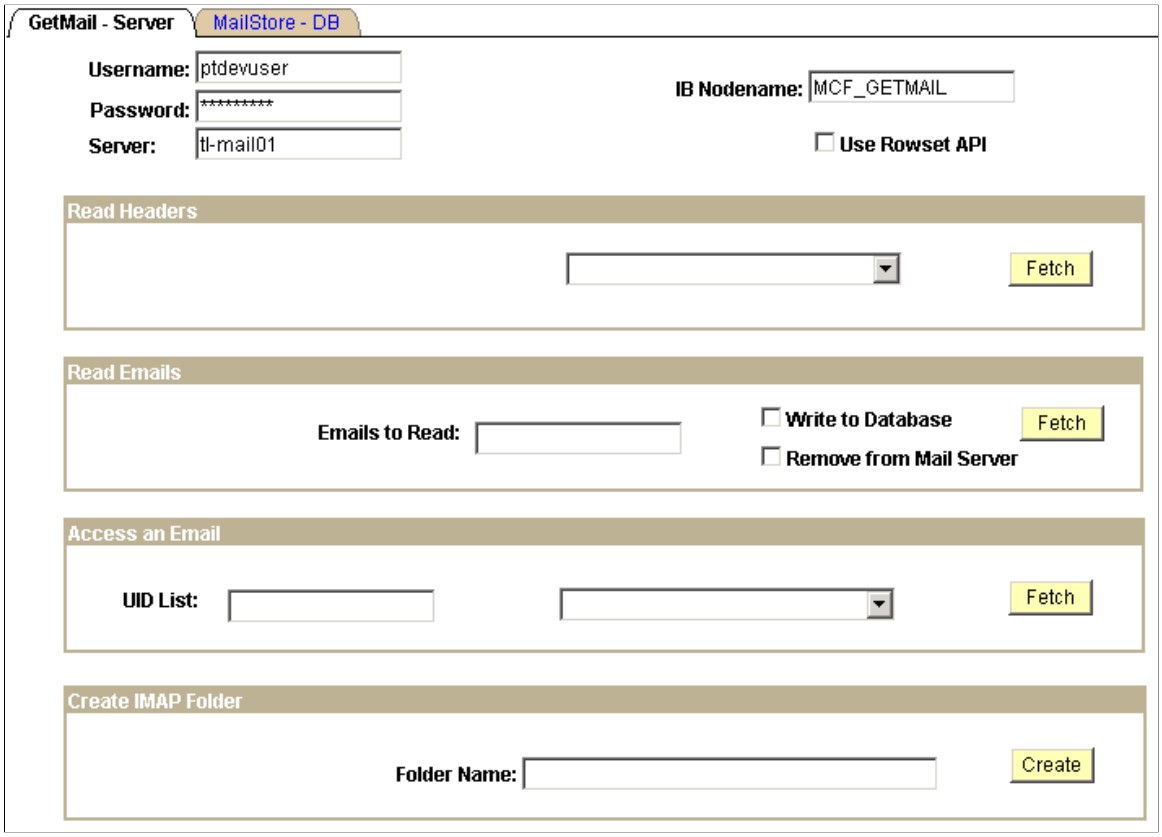
Note: Using the email sample pages requires access to a mail server.

Using the GetMail - Server Page

Access the GetMail - Server page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Email** > **Sample Email Pages** > **GetMail - Server**

This example illustrates the fields and controls on the GetMail - Server page. You can find definitions for the fields and controls later on this page.



This page demonstrates reading or deleting emails from the mail server. It also demonstrates the storage and retrieval of emails to and from the database tables. Output of the response to each of the requests is written to a file at %PS_SERVDIR%/files/mcfdata.out on the application server. Check this file after each test for the output data.

To demonstrate email functionality, send an email to the user name and server that you enter on the GetMail - Server page.

Field or Control	*Description*
Username and Password	Enter a valid username and password for an account on the mail server.

Field or Control	*Description*
Server	Enter the mail server.
IB Nodename (Integration Broker node name)	Enter the PeopleSoft Integration Broker node used to access email from the mail server. The default is MCF_GETMAIL
Use Rowset API	Select to use the rowset-based GetMail API. Deselect to use the new Mail Classes API.

Read Headers

Select a method to be called from the drop-down list box, then click **Fetch.** Select from:

Field or Control	*Description*
Message Count:	Writes the number of emails in the specified mailbox to the output file mcfdata.out.
ReadHeadersWithAttach:	Writes headers and attachment information for all emails in the specified mailbox to the output file mcfdata.out.

Read Emails

Enter the number of emails to read, select other parameters, then click **Fetch** to read emails.

Field or Control	*Description*
Emails to Read	Enter the number of emails to retrieve from the specified mailbox and write to the output file mcfdata.out.
Write to Database	Select to have the retrieved emails written to the database as well as to the output file mcfdata.out.
Remove from Mail Server	Select to delete emails from the mail server after they have been retrieved .

Access an Email

Enter an email UID list, select a method from the drop-down list box, and then click **Fetch** to access the selected emails.

Field or Control	*Description*
UID List (unique identifier list)	Enter the unique ID of an email to be retrieved or deleted. To delete, you can enter a comma-delimited list of unique email IDs (UID) to be deleted. Each email has a unique number (the UID) associated with it that is permanently guaranteed not to refer to any other email in the mailbox. To find the UID of an email, run the *ReadHeadersWith Attach* option first, which returns the UID for each email. If an invalid UID is specified, one empty row is returned.
Read Email w/ attachment	Retrieves the specified email from the specified mailbox and writes it to the output file mcfdata.out.
Remove Message	Deletes the specified email from the specified mailbox.

Create IMAP Folder

Create an email folder; only valid for IMAP4 mail servers.

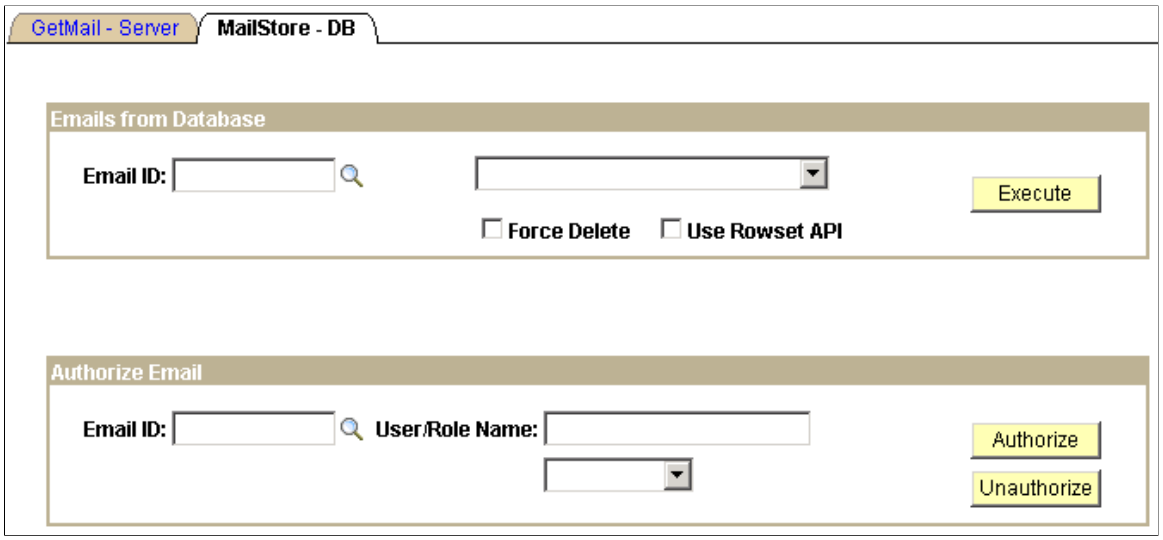
Field or Control	*Description*
Folder Name	Enter the name of the folder to create.

Using the MailStore - DB Page

Access the MailStore - DB page using the following navigation path:

PeopleTools > **MultiChannel Framework** > **Email** > **Samples Pages** > **MailStore - DB**

This example illustrates the fields and controls on the MailStore - DB page. You can find definitions for the fields and controls later on this page.



View an example of how to access emails from the database (how to retrieve and delete email) and how to authorize email attachments for viewing or deletion using the PT_MCF_MAIL application package provided with PeopleSoft MCF.

Field or Control	*Description*
Email ID	Enter the PeopleSoft email ID (not to be confused with the UID) with which the email was saved to the database.

Emails from Database

Enter an email ID, select an action from the drop-down list box, and then click **Execute** to perform the selected action.

Field or Control	*Description*
Delete Email from Database	Deletes the specified email from the database and any corresponding attachments from the repository.
Retrieve Email	Retrieves the specified email from the database and writes it to the output file mcfdata.out.
Force Delete	Select after entering *Delete Email from Database* to force the deletion even if an error occurs when deleting associated attachments from the repository.

Field or Control	*Description*
Use Rowset API	Select to use the rowset-based GetMail API. Deselect to use the new Mail Classes API.

Authorize Email

Enter an email ID, a user role or name, and click **Authorize** or **Unauthorize** to perform the specified action.

Field or Control	*Description*
Email ID	Enter the PeopleSoft email ID (not to be confused with the UID) with which the email was saved to the database.
User/Role Name	Enter a PeopleSoft user ID or role to be authorized or unauthorized to view the attachment associated with the selected email. After entering an ID or role, select *User* or *Role,* as appropriate, from the drop-down list box.

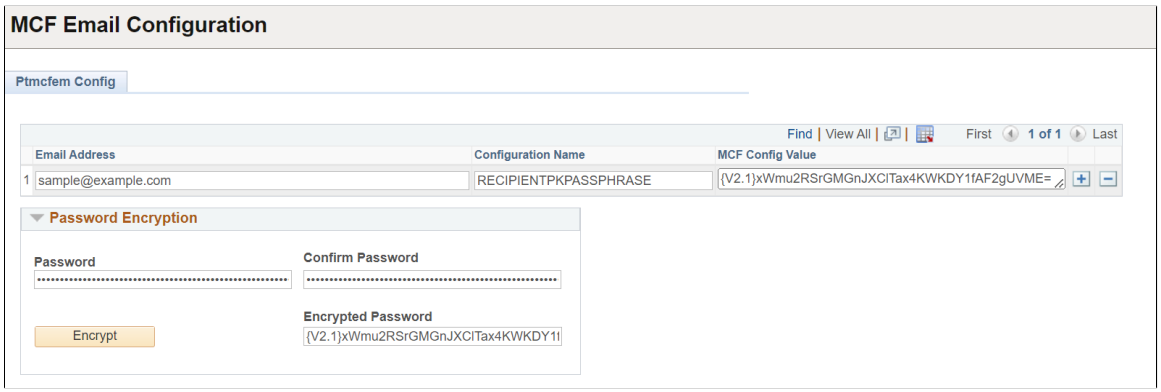
Related Links

Using the Email Sample Page

Configuring Signed and Encrypted Emails

To facilitate the decryption of signed and encrypted emails in PeopleSoft, you must configure the existing PeopleSoft pages by adding public certificates, private keys, and private key passphrases to the database.

This example illustrates the fields and controls on the MCF Email Configuration page. You can find definitions for the fields and controls later on this page.



Field or Control	*Description*
Email Address	Enter the email address of the recipient.
Configuration Name	Enter the configuration name. For inbound encrypted emails, this is always RECIPIENTPKPASSPHRASE. Private keys that are used with outbound emails are filed under the configuration name SENDERPKPASSPHRASE.
Password	Enter the password for encryption.
Confirm Password	Confirm the password for encryption.
Encrypted Password	Click Encrypt button to generate the encrypted password.
MCF Config Value	Copy and paste the encrypted password.

Configuring Algorithms for Inbound Emails

To verify and decode signed inbound emails, you must configure the smime_signed_verify algorithm keyset. If an email may be both signed and encrypted, you must configure the smime_signandencrypt_decryptandverify algorithm. For encrypted, but not signed emails, the smime_encrypted_decrypt algorithm must be configured.

To configure the smime_signed_verify algorithm, add the email sender's public certificate to the corresponding keyset after ensuring that the keyset ID is the email address of the email sender.

To configure the smime_encrypted_decrypt algorithm and the smime_signandencrypt_decryptandverify algorithm:

1. Add the email sender's public certificate, the email receiver's public certificate, and the email receiver's private key to the corresponding keyset.

   Note: Ensure that the keyset ID for the certificates is the associated email address. Prefix the greater than (>) sign to the email address for the recipient's private key.

   See “Understanding PeopleSoft Encryption Technology” (Security Administration).

2. Add the email recipient's private key passphrase to the MCF configuration page.

   For all private keys entered into the keyset algorithms mentioned in this section, you must enter the passphrases for the recipient’s private keys into the MCF Email Configuration page. To access the configuration page for private key passphrases, select **PeopleTools** > **MultiChannel Framework** > **Email** > **MCF Email Configuration.**

Configuring Algorithms for Outbound Emails

To verify and decode signed outbound emails, you must configure the following parameters in the psappsrv.cfg or psprcs.cfg files, using the SMTP Settings section that already contains outbound email settings:

- SMTPSMIMEEncryption
- SMTPSMIMESignature
- SMTPSMIMEHandling
- SMTPSMIMEHandling1
- SMTPSMIMEEncryption1
- SMTPSMIMESignature1

To configure outbound signed emails:

1. Configure the smime_signed_sign algorithm.

   See Emails — smime_signed_sign section in “Understanding the Supported Algorithms” (Security Administration).

2. Add the email sender's private key passphrase to the MCF configuration page.

   Note: You must not configure the private key passphrase in the keyset.

To configure outbound encrypted emails:

1. Configure the smime_encrypted_encrypt algorithm.

   See Emails — smime_encrypted_encrypt section in “Understanding the Supported Algorithms” (Security Administration).

2. Add the email receiver's private key passphrase to the MCF configuration page.

   Note: You must not configure the private key passphrase in the keyset.

To configure outbound emails that are both signed and encrypted:

1. Configure the smime_signandencrypt_signandencrypt algorithm.

   See Emails — smime_signandencrypt_signandencrypt section in “Understanding the Supported Algorithms” (Security Administration).

2. Add the email sender's private key passphrase to the MCF configuration page.

   Note: You must not configure the private key passphrase in the keyset.

Chapter 14

JSMCAPI Quick Reference

JSMCAPI Classes

This section lists the JSMCAPI classes in alphabetical order, along with the constructor, fields, methods, callback event methods, and returns associated with each class.

_Address

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	***Returns***
_Address()	_Address object

Fields

Field	***Type***
caps	Object
id	String

Callback Event Methods

Callback Event Method	***Returns***
onError(*event*)	None

_UQAddress

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

The _UQAddress class extends the _Address class.

See _Address.

Constructor

Constructor	***Returns***
_UQAddress()	_UQAddress object

Fields

Field	***Type***
tasks	Associative array

Methods

Method	***Returns***
acceptTask(*task*, *reason*)	None
dequeueTask(*task*)	Object

Callback Event Methods

The _UQAddress class inherits the onError callback event method from the _Address class.

See _Address.

Callback Event Method	***Returns***
onAccepted(*event*)	None
onAcceptingTask(*event*)	None
onDequeueingTask(*event*)	None
onNotify(*event*)	None
onTaskAdded(*event*)	None
onTaskRemoved(*event*)	None

Callback Event Method	*Returns*
onUnassigned(*event*)	None

_User

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
_User()	_User object

Fields

Field	*Type*
agentId	String
caps	Associative array
id	String
name	String
presences	Associative array
states	Associative array of the constants beginning with ST_*.
ST_LOGGEDIN	String
ST_LOGGEDOUT	String
ST_NOTREADY	String
ST_READY	String
ST_UNKNOWN	String

Field	*Type*
ST_WORKNOTREADY	String
ST_WORKREADY	String
statistics	Object
statistics1	Object
statistics2	Object

A2AChat

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

The A2AChat class extends the _Address class.

See _Address.

Constructor

Constructor	*Returns*
A2AChat(*event*, *address*, *chatType*)	A2AChat object

Fields

Field	*Type*
address	Object
agentId	String
agentName	String
appData	Object
chatType	String

Field	*Type*
customerName	Object
id	String
isConference	Object
jr	String of the PS_JR constant.
type	String of the following constants: • TYPE_ANSWER • TYPE_CONSULT
uniqueueId	String

Methods

Method	*Returns*
getUrl(*defaultUrl*)	String

A2AChatAddress

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
A2AChatAddress()	A2AChatAddress object

Fields

The A2AChatAddress class inherits the following fields from the _Address class:

- id
- caps

See _Address.

Field	*Type*
id	String
tasks	Associative array

Methods

Method	*Returns*
initiateChat(*agentId*)	None

Callback Event Methods

The A2AChatAddress class inherits the onError callback event method from the _Address class.

See _Address.

Callback Event Method	*Returns*
onChatEnded(*event*)	None
onInitiatingChat(*event*)	None
onNotify(*event*)	None

AgentStatistics

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
AgentStatistics()	AgentStatistics object

Fields

Field	*Type*
averageCallDuration	Number
averageHoldDuration	Number
callsHandled	Number
data	Associative array
percentIdleTime	Number
percentTimeAvailable	Number
percentTimeInCurrentState	Number
percentTimeUnavailable	Number
taskAcceptedCurrentLogin	Number
timeCurrentLogin	Number
timeWorking	Number
totalTaskAcceptedLogin	Number
totalTaskDoneLogin	Number
totalTaskDoneLogin	Number
unavailableDuration	Number
waitDuration	Number

AppData

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
AppData()	AppData object

Fields

Field	*Type*
data	Associative array
groupId	String
jr	String
question	String
strData	String
subject	String
uniqueId	String
url	String of the constant URL
userId	String
username	String
wizUrl	String

Methods

Method	*Returns*
addKeyValue(*key*, *value*)	None

Buddy

This section lists the constructor, methods, and callback event methods associated with this class.

The Buddy class extends the _User class.

See _User.

Constructor

Constructor	***Returns***
Buddy()	Buddy object

Fields

The Buddy class inherits the following fields from the _User class:

- agentID
- caps
- id
- name
- presences
- states
- statistics
- statistics1
- statistics2

See _User.

Callback Event Methods

Callback Event Method	***Returns***
onStat1(*event*)	None
onStat2(*event*)	None
onState(*event*)	None

Call

This section lists the constructor, fields, and returns (if applicable) associated with this class.

The Call class extends the Task class.

See Task.

Constructor

Constructor	***Returns***
Call(*strCall*)	Call object

Fields

The Call class inherits the following fields from the Task class:

- caseid
- cost
- customerid
- group
- id
- onStat
- priority
- type
- urlAbs
- urlRel

See Task.

Field	***Type***
line	Object
statistics	Object
userData	String

CallData

This section lists the constructor, fields, methods, and returns (if applicable) associated with this class.

Constructor

Constructor	***Returns***
CallData()	CallData object

Fields

Field	***Type***
ani	String
callId	String
callType	String
data	String
dnis	String

Methods

Method	***Returns***
addKeyValue(*key , value*)	None

CallStatistics

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	***Returns***
CallStatistics()	CallStatistics object

Fields

Field	*Type*
data	Associative array
holdTime	String
queueTime	String
talkTime	String

Chat

This section lists the constructor, fields, methods, and returns (if applicable) associated with this class.

The Chat class extends the Task class.

See Task.

Constructor

Constructor	*Returns*
Chat(*event*, *address*)	Chat object

Fields

The Chat class inherits the following fields from the Task class:

- caseid
- cost
- customerid
- group
- id
- onStat
- priority
- type
- urlAbs

- urlRel

See Task.

Field	*Type*
address	Object
agentId	String
appData	Object
chatconnection	Object
chatType	String
customerName	String
groupId	String
question	String
statistics	Object
subject	String
userData	Object

Methods

Method	*Returns*
gettpUrl(*defaultUrl*)	String
getUrl(*defaultUrl*)	String

ChatAddress

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

The ChatAddress class extends the _UQAddress class.

See _UQAddress.

Constructor

Constructor	***Returns***
ChatAddress()	ChatAddress object

Fields

The ChatAddress class inherits the following fields from the _Address class:

- id
- caps

Field	***Type***
chatconnections	Object
tasks	Object

See _Address.

The ChatAddress class inherits the tasks field from the _UQAddress class.

See _UQAddress.

Methods

The ChatAddress class inherits the following methods from the _UQAddress class:

- acceptTask
- dequeueTask

Method	***Returns***
chat (*agentId*, *buddyId*, *reason*)	None
getChatconnectionByConnectionId (*connectionId*)	Object
getChatconnectionindexByConnectionId (*connectionId*)	Index
getFreeChatconnection	Object

Method	*Returns*
getFreeChatconnectionIndex	Object

See _UQAddress.

Callback Event Methods

The ChatAddress class inherits the onError callback event method from the _Address class.

See _UQAddress.

The ChatAddress class inherits the following callback event methods from the _UQAddress class:

- onAccepted
- onAcceptingTask
- onDequeueingTask
- onNotify
- onTaskAdded
- onTaskRemoved
- onUnassigned

See _UQAddress.

Callback Event Method	*Returns*
onCapabilitiesChanged (*event*)	None

ChatConnection

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
ChatConnection(*event*)	ChatConnection object

Fields

Field	***Returns***
caps	Object
chat	String
connectionId	String
id	String
state	String

Methods

Method	***Returns***
answer (*agentid*, *reason*)	None
conference (*agentid*, *reason*)	None
forward (*fromagentid*, *toagentid*, *qid*, *userdata*, *chatdata*, *reason*)	None
gethistory (*agentid*, *noofLines*, *reason*)	None
getUrl (*deafultURL*)	String
message (*agentid*, *message*, *reason*)	None
pushURL(*URL*)	None
reject(*agentid*, *reason*)	None
release(*agentid*, *reason*)	None
typing (*agentid*, *reason*)	None

Callback Event Methods

Callback Event Method	*Returns*
onAccepted (*event*)	None
onAnswering (*event*)	None
onCapabilitiesChanged (*event*)	None
onChatdataChanged (*event*)	None
onConferencing (*event*)	None
onDialing (*event*)	None
onDropped (*event*)	None
onError (*event*)	None
onForwarded (*event*)	None
onForwardError (*event*)	None
onForwarding (*event*)	None
onHistory (*event*)	None
onIncomingChat (*event*)	None
onMessage (*event*)	None
onPartyAdded (*event*)	None
onPartyChanged (*event*)	None
onPartyRemoved (*event*)	None
onProperties (*event*)	None

Callback Event Method	*Returns*
onPushURL (*event*)	None
onRejected (*event*)	None
onReleased (*event*)	None
onReleasing (*event*)	None
onRevoked (*event*)	None
onTalking (*event*)	None
onTyping (*event*)	None
onUserdataChanged (*event*)	None

ChatConnectionCaps

This section lists the constructor, fields, methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
ChatConnectionCaps(*event*)	ChatConnectionCaps object

Fields

Field	*Type*
canAnswer	Boolean
canConference	Boolean
canConferenceSingle	Boolean
canForward	Boolean

Field	*Type*
canIndicateTyping	Boolean
canPushURL	Boolean
canReject	Boolean
canSendMessage	Boolean

ChatData

This section lists the constructor, fields, methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
ChatData(*event*)	ChatData object

Fields

Field	*Type*
data	Object

Methods

Method	*Returns*
addKeyValue(*key*, *value*)	None

Connection

This section lists the constructor, methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
Connection (*event*)	ConnectionEvent object

Methods

Method	*Returns*
eventUserDataChanged(*event*)	None

ConnectionListener

This section lists the constructor, methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
ConnectionListener (*event*)	ConnectionListener object

Methods

Method	*Returns*
requestAttachUserdata(*object*, *request*)	None

ConnectionRequest

This section lists the constructor, methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
ConnectionRequest (*event*)	ConnectionRequest object

Methods

Method	*Returns*
getUserData( )	UserData

Email

This section lists the constructor, fields, methods, and returns (if applicable) associated with this class.

The Email class extends the Task class.

See Task.

Constructor

Constructor	*Returns*
Email(*event*)	Email object

Fields

The Email class inherits the following fields from the Task class:

- caseid
- cost
- customerid
- group
- id
- onStat
- priority
- type
- urlAbs
- urlRel

See Task.

Field	*Type*
address	Object

Field	*Type*
agentId	Object
appData	Object
customerName	String
emailconnection	Object
emailId	String
groupId	String
question	String
statistics	Object
subject	String
userData	Object

Methods

Method	*Returns*
gettpUrl(*defaultUrl*)	Object
getUrl(*defaultUrl*)	String

EmailAddress

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

The EmailAddress class extends the _UQAddress class.

See _UQAddress.

Constructor

Constructor	*Returns*
EmailAddress()	EmailAddress object

Fields

The EmailAddress class inherits the following fields from the _Address class:

- id
- caps

See _Address.

The EmailAddress class inherits the tasks field from the _UQAddress class.

See _UQAddress.

Field	*Type*
agent	Object
emailconnections	Object

Methods

The EmailAddress class inherits the following methods from the _UQAddress class:

- acceptTask
- dequeueTask

See _UQAddress.

Method	*Returns*
getEmailconnectionByConnectionId (*connectionId*)	Object
getEmailconnectionindexByConnectionId (*connectionId*)	Index
getFreeEmailconnection	Object

Callback Event Methods

The EmailAddress class inherits the onError callback event method from the _Address class.

See _Address.

The EmailAddress class inherits the following callback event methods from the _UQAddress class:

- onAccepted
- onAcceptingTask
- onDequeueingTask
- onNotify
- onTaskAdded
- onTaskRemoved
- onUnassigned

See _Address.

EmailConnection

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	***Returns***
EmailConnection(*event*)	EmailConnection object

Fields

Field	***Type***
caps	Object
connectionId	String
email	String
id	String
state	String

Methods

Method	***Returns***
abandon (*reason*)	None
answer (*reason*)	None
complete (*reason*)	None
forward (*fromagentid*, *toagentid*, *qid*, *userdata*, *Emaildata*, *reason*)	None
reject(*agentid*, *reason*)	None
withdraw_RES (*reason*)	None

Callback Event Methods

Callback Event Method	***Returns***
onAnswering (*event*)	None
onCapabilitiesChanged (*event*)	None
onCompleted (*event*)	None
onDropped (*event*)	None
onEmaildataChanged (*event*)	None
onError (*event*)	None
onForwarded (*event*)	None
onForwardError (*event*)	None
onForwarding (*event*)	None
onIncoming (*event*)	None

Callback Event Method	*Returns*
onProcessing (*event*)	None
onRejected (*event*)	None
onRevoked (*event*)	None
onUserdataChanged (*event*)	None
onWithdraw_REQ (*event*)	None

EmailConnectionCaps

This section lists the constructor, fields, methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
EmailConnectionCaps(*event*)	EmailConnectionCaps object

Fields

Field	*Type*
canAnswer	Boolean
canComplete	Boolean
canForward	Boolean
canReject	Boolean

EmailData

This section lists the constructor, fields, methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
EmailData(*event*)	EmailData object

Fields

Fields	*Type*
data	Object

Methods

Method	*Returns*
addKeyValue(*key*, *value*)	None

Extension

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

The Extension class extends the _Address class.

See _Address.

Constructor

Constructor	*Returns*
Extension(*numOfLines*)	Extension object

Fields

The Extension class inherits the following fields from the _Address class:

- id
- caps

See _Address.

Field	*Type*
forwardMode	Object
isDnd	Boolean
lines	Associative array
numOfLines	Number

Methods

Method	*Returns*
cancelDnd(*reason*)	None
cancelForwardSet(*reason*)	None
forwardSet(*number*, *mode*, *reason*)	None
getDialingLine()	Object
getFreeLine()	Object
getLineByConnectionId(*connectionId*)	Object
getOffHookLine()	Object
setDnd(*reason*)	None

Callback Event Methods

The Extension class inherits the onError callback event method from the _Address class.

See _Address.

Callback Event Method	*Returns*
onCancelingDnd(*event*)	None

Callback Event Method	*Returns*
onCancelingForward(*event*)	None
onDnd(*event*)	None
onDndCanceled(*event*)	None
onForwardCanceled(*event*)	None
onForwarded(*event*)	None
onForwarding(*event*)	None
onSettingDnd(*event*)	None

ExtensionCaps

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
ExtensionCaps(*strCaps*)	ExtensionCaps object

Fields

Field	*Type*
canCancelDnd	Boolean
canDial	Boolean
canFwdBusy	Boolean
canFwdBusyNoAnswer	Boolean
canFwdCancelForward	Boolean

Field	*Type*
canFwdDefault	Boolean
canFwdNoAnswer	Boolean
canFwdUnconditional	Boolean
canRefreshState	Boolean
canSetDnd	Boolean

ForwardMode

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
ForwardMode()	ForwardMode object

Fields

Field	*Type*
mode	String of the following constants: • BUSY • BUSYNOANSWER • DEFAULT • NOANSWER • NONE • UNCONDITIONAL

GenericAddress

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

The GenericAddress class extends the _UQAddress class.

See _UQAddress.

Constructor

Constructor	***Returns***
GenericAddress()	GenericAddress object

Fields

The GenericAddress class inherits the following fields from the _Address class:

- id
- caps

See _Address.

The GenericAddress class inherits the tasks field from the _UQAddress class.

See _UQAddress.

Field	***Type***
agent	Object
genericconnections	Object

Methods

The GenericAddress class inherits the following methods from the _UQAddress class:

- acceptTask
- dequeueTask

See _UQAddress.

Method	***Returns***
getGenericconnectionByConnectionId (*connectionId*)	Object
getGenericconnectionindexByConnectionId (*connectionId*)	Index
getFreeGenericconnection	Object

Callback Event Methods

The GenericAddress class inherits the onError callback event method from the _Address class.

See _Address.

The GenericAddress class inherits the following callback event methods from the _UQAddress class:

- onAccepted
- onAcceptingTask
- onDequeueingTask
- onNotify
- onTaskAdded
- onTaskRemoved
- onUnassigned

See _UQAddress.

GenericConnection

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	***Returns***
GenericConnection(*event*)	GenericConnection object

Fields

Field	***Type***
caps	Object
connectionId	String
generic	String
id	String
state	String

Methods

Method	***Returns***
abandon (*reason*)	None
answer (*reason*)	None
complete (*reason*)	None
forward (*fromagentid*, *toagentid*, *qid*, *userdata*, *Genericdata*, *reason*)	None
reject(*agentid*, *reason*)	None
withdraw_RES (*reason*)	None

Callback Event Methods

Callback Event Method	***Returns***
onCapabilitiesChanged (*event*)	None
onCompleted (*event*)	None
onDropped (*event*)	None
onError (*event*)	None
onForwarded (*event*)	None
onForwardError (*event*)	None
onForwarding (*event*)	None
onGenericdataChanged (*event*)	None
onIncoming (*event*)	None
onProcessing (*event*)	None

Callback Event Method	*Returns*
onRejected (*event*)	None
onRevoked (*event*)	None
onUserdataChanged (*event*)	None
onWithdraw_REQ (*event*)	

GenericConnectionCaps

This section lists the constructor, fields, methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
GenericConnectionCaps(*event*)	GenericConnectionCaps object

Fields

Field	*Type*
canAnswer	Boolean
canComplete	Boolean
canForward	Boolean
canReject	Boolean

GenericData

This section lists the constructor, fields, methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
GenericData(*event*)	GenericData object

Fields

Field	*Type*
data	Object

Methods

Method	*Returns*
addKeyValue(*key*, *value*)	None

GenericTask

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

The GenericTask class extends the Task class.

See Task.

Constructor

Constructor	*Returns*
GenericTask(*event*)	GenericTask object

Fields

The GenericTask class inherits the following fields from the Task class:

- caseid
- cost
- customerid
- group

- id
- onStat
- priority
- type
- urlAbs
- urlRel

See Task.

Field	***Type***
address	String
agentId	String
appData	Object
customerName	String
genericconnection	Object
genericId	String
groupId	String
question	String
subject	String
userData	Object

Methods

Method	***Returns***
gettpUrl(*defaultUrl*)	Object
getUrl(*defaultUrl*)	String

GLOBALS

This section lists the fields and methods associated with this class.

Fields

Field	*Type*
A2AChat.PS_JR	Constant
A2AChat.TYPE_ANSWER	Constant
A2AChat.TYPE_CONSULT	Constant
Server.TYPE.CTI	Constant
Server.TYPE_UQ	Constant
Task.TYPE_A2ACHAT	Constant
Task.TYPE_CHAT	Constant
Task.TYPE_CTI	Constant
Task.TYPE_EMAIL	Constant
Task.TYPE_GENERIC	Constant

Methods

Method	*Returns*
initJSMCAPI()	None
isValid(*obj*)	Boolean
MCFBroadcast(*cluster*, *cluster*, *cluster*, *cluster*, *cluster*, *cluster*, *cluster*, *cluster*, *cluster*, *cluster*)	None

Group

This section lists the constructor, fields, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	***Returns***
Group()	Group object

Fields

Field	***Type***
id	String
name	String
registered	Boolean
statistics	Object
statistics1	Object
statistics2	Object

Callback Event Methods

Callback Event Methods	***Returns***
onStat(*event*)	None
onStat1(*event*)	None
onStat2(*event*)	None
onTaskAdded(*event*)	None
onTaskRemoved(*event*)	None

GroupStatistics

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
GroupStatistics()	GroupStatistics object

Fields

Field	*Type*
data	Associative array
listOfTasksInTheQueueByTaskType	Number
maxTaskCompletionTime	Number
newestTask	Number
newestTaskCompletionTime	Number
numberOfAbandoned **Note:** This field is only updated for voice tasks, and only if the CTI adapter supports this calculation.	String
numberOfLoggedIn	String
numberOfQueued	String
numUnassignedTasks	Number
queuedWaitTime	String
queueUpTime	Number
relativeQueueLoad	String

Field	*Type*
timeElapsedOldestTask	Number

GroupStatistics1

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
GroupStatistics1(*data*)	GroupStatistics1 object

Fields

Field	*Type*
mostRecentTaskDone	String
mostRecentTaskDoneData	String
mostRecentTaskEnqueued	String
mostRecentTaskEnqueuedData	String
numAgentsAvailable	String
numAgentsLoggedIn	String
numEscalation	String
numOverflow	String
numTaskAccepted	String
numTaskDone	String
numTaskQueued	String

Field	*Type*
reasonFlag	String
taskTotalTimeInSystem	String
timeSinceStart	String

GroupStatistics2

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
GroupStatistics2(*data*)	GroupStatistics2 object

Fields

Field	*Type*
averageTaskDuration	String
averageWaitTime	String
oldestTask	String
recentTask	String
timeElapsedOldestTask	String
timeElapsedRecentTask	String

Line

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
Line()	Line object

Fields

Field	*Type*
call	Object
caps	Object
connectionId	String
id	String
isMuted	Boolean
state	String of the following constants: • ST_DIALING • ST_DROPPED • ST_HELD • ST_IDLE • ST_OFFHOOK • ST_RINGING • ST_TALKING

Methods

Method	*Returns*
alternate(*reason*)	None
answer(*reason*)	None

Method	***Returns***
attachUserData(*userdata*, *reason*)	None
clear(*reason*)	None
complete(*reason*)	None
conference(*destination*, *reason*, *userdata*, *calldata*)	None
conferenceSingle(*destination*, *reason*, *userdata*, *calldata*)	None
dial(*number*, *reason*, *userdata*)	None
dropParty(*destination*, *reason*)	None
getAni()	String
getDescr()	String
getDnis()	String
getReferenceId()	String
getUrl(*defaultUrl*)	String
grabCall(*destination*, *reason*)	String
hold(*reason*)	String
join (*reason*, *conferenceId*)	None
mute (*reason*)	None
park(*destination*, *reason*)	None
reconnect(*reason*)	None
reject(*reason*)	None

Method	*Returns*
release(*reason*)	None
retrieve(*reason*)	None
sendDTMF(*reason*, stringDTMF)	None
setcallresult(*result*)	None
setcallresultDNC(*reason*, *number*)	None
setcallresultReschedule(*hours*, *minutes*,*day*, *month*,*year*)	None
transfer(*destination*, *reason*, *userdata*, *calldata*)	None
transferMute(*destination*, *reason*, *userdata*, *calldata*)	None
unmute(*reason*)	None
updateCallData(*calldata*, *reason*)	None

Callback Event Methods

Callback Event Method	*Returns*
onAlternating(*event*)	None
onAnswering(*event*)	None
onAttachingUD(*event*)	None
onCallDataChanged(*event*)	None
onCapabilitiesChanged(*event*)	None
onClearing(*event*)	None
onCompleting(*event*)	None

Callback Event Method	***Returns***
onConferencing(*event*)	None
onDialing(*event*)	None
onDropped(*event*)	None
onError(*event*)	None
onGrabbing(*event*)	None
onHeld(*event*)	None
onHolding(*event*)	None
onJoining(*event*)	None
onMuted(*event*)	None
onOffHook(*event*)	None
onOnHook(*event*)	None
onParking(*event*)	None
onPartyAdded(*event*)	None
onPartyChanged(*event*)	None
onPartyRemoved(*event*)	None
onReconnecting(*event*)	None
onRejected(*event*)	None
onRejecting(*event*)	None
onReleasing(*event*)	None

Callback Event Method	*Returns*
onRetrieving(*event*)	None
onRinging(*event*)	None
onSetcallresult(*event*)	None
onSetcallresultDNC(*event*)	None
onSetcallresultReschedule(*event*)	None
onTalking(*event*)	None
onTransfering(*event*)	None
onUnmuted(*event*)	None
onUpdatingCD(*event*)	None
onUserDataChanged(*event*)	None

LineCaps

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
LineCaps(*strCaps*)	LineCaps object

Fields

Field	*Type*
canAlternate	Boolean
canAnswer	Boolean

Field	*Type*
canAttachUserData	Boolean
canClear	Boolean
canComplete	Boolean
canConference	Boolean
canConferenceSingle	Boolean
canDropParty	Boolean
canHold	Boolean
canMute	Boolean
canPark	Boolean
canReconnect	Boolean
canReject	Boolean
canRelease	Boolean
canRetrieve	Boolean
canSendDTMF	Boolean
canSetcallresult	Boolean
canSetcallresultDNC	Boolean
canSetcallresultReschedule	Boolean
canTransfer	Boolean
canTransferMute	Boolean

Field	*Type*
canUnmute	Boolean
canUpdateCallData	Boolean

MCEvent

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
MCEvent()	MCEvent object

Fields

Field	*Type*
extension	Object
group	Object
reason	Object
user	Object

MediaType

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
MediaType()	MediaType object

Fields

Field	*Type*
type	String of the following constants: • MT_EMAIL • MT_VOICE

PSMC

This section lists the constructor, fields, methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
PSMC()	PSMC object

Fields

Field	*Type*
renServer	Object
servers	Associative array
sessions	Associative array

Methods

Method	*Returns*
closeSession(*serverId*)	None
getCallById(*id*)	Object
getChatById(*id*)	Object
getEmailById(*id*)	Object

Method	*Returns*
getGenericTaskById(*id*)	Object
getLineById(*id*)	Object
openSession(*serverId*)	None
start()	None
stop()	None

Reason

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
Reason(*code*, *desc*)	Reason object

Fields

Field	*Type*
code	String
description	String
reasonData1	String
reasonData2	String
reasonData3	String

RenServer

This section lists the constructor, fields, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
RenServer()	RenServer object

Fields

Field	*Type*
isRunning	Boolean
url	String

Callback Event Methods

Callback Event Method	*Returns*
onDown(*event*)	None
onUp(*event*)	None

Server

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
Server(*serverID*)	Server object

Fields

Field	*Type*
id	String
info	String

Field	*Type*
state	String of the following constants: • ST_INSERVICE • ST_OUTOFSERVICE
type	String of the following constants: • TYPE_CTI • TYPE_UQ

Callback Event Methods

Callback Event Method	*Returns*
onBroadcast(*event*)	None
onHbLost(*event*)	None
onHbRecovered(*event*)	None
onInService(*event*)	None
onOutOfService(*event*)	None
onRestart(*event*)	None

Session

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
Session(*serverID*)	Session object

Fields

Field	*Type*
addresses	Associative array
buddies	Associative array
groups	Associative array
id	String
intervalBetweenReqs	Number
numberRegsPerBulkReq	Number
serverID	String
state	String of the following constants: • ST_ACTIVE • ST_CLOSED • ST_CLOSING • ST_IDLE
user	User object

Methods

Method	*Returns*
broadcastSubscribe(*cluster*, *queue*, *task*, *state*, *presence*, *method*)	None
broadcastUnsubscribe(*type*)	None
close()	None
open()	None

Method	*Returns*
registerAddress(*address*)	None
registerBuddy(*buddy*)	None
registerBuddiesBulk(*buddies*, *numRegsPerBulkReq*, *intervalBetweenReqs*)	None
registerGroup(*group*)	None
registerGroupsBulk(*groups*, *numRegsPerBulkReq*, *intervalBetweenReqs*)	None
registerUser(*user*)	None
setAutoRecovery(*autoRecover*)	None
statPublish(*userStat*, *userStat*)	None
unregisterAddress(*address*)	None
unregisterBuddy(*buddy*)	None
unregisterGroup(*group*)	None
unregisterUser(*user*)	None

Callback Event Methods

Callback Event Method	*Returns*
onAddressRegistered(*event*)	None
onAddressUnregistered(*event*)	None
onBuddyRegistered(*event*)	None
onBuddyUnregistered(*event*)	None

Callback Event Method	*Returns*
onClosed(*event*)	None
onError(*event*)	None
onGroupRegistered(*event*)	None
onGroupUnregistered(*event*)	None
onInfo(*event*)	None
onOpened(*event*)	None
onUserRegistered(*event*)	None
onUserUnregistered(*event*)	None

Task

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
Task()	Task object

Fields

Field	*Type*
caseid	String
cost	String
customerid	String
group	String

Field	*Type*
id	String
onStat	String
priority	String
type	String; one of the following: • TYPE_A2ACHAT • TYPE_CHAT • TYPE_CTI • TYPE_EMAIL • TYPE_GENERIC
urlAbs	Object
urlRel	Object

Methods

Method	*Returns*
setUserData(*UserData data*)	None
getUserData( )	UserData

TaskStatistics

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
TaskStatistics()	TaskStatistics object

Fields

Field	*Type*
data	Associative array
holdTime	String
queueTime	String
talkTime	String

User

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

The User class extends the _User class.

See _User.

Constructor

Constructor	*Returns*
User(*id*, *name*, *agentId*, *agentPassword*)	User object

Fields

The User class inherits the following fields from the _User class:

- agentId
- caps
- id
- name
- presences
- states
- statistics
- statistics1
- statistics2

See _User.

Field	*Type*
addresses	Associative array
agentPassword	String
language	String
registeredAddresses	Associative array
statesUq	Object

Methods

Method	*Returns*
disableMedia(*group*, *mediaType*, *extension*, *reason*)	None
enableMedia(*group*, *mediaType*, *extension*, *reason*)	None
isMediaEnabled(*mediaType group*)	None
login(*group*, *extension*, *reason*)	None
loginUq(*group*)	None
logout(*group*, *extension*, *reason*)	None
logoutUq(*group*)	None
register(*extension*, *reason*)	None
setNotReady(*group*, *presence*, *extension*, *reason*)	None
setPresence(*group*, *presence*, *extension*, *reason*)	None
setPresenceUq(*group*, *presenceText*, *reason*)	None

Method	*Returns*
setReady(*group*, *presence*, *extension*, *reason*)	None
setWorkNotReady(*group*, *presence*, *extension*, *reason*)	None
setWorkReady(*group*, *presence*, *extension*, *reason*)	None
unRegister(*extension*, *reason*)	None

Callback Event Methods

Callback Event Method	*Returns*
onCapabilitiesChanged(*event*)	None
onCtiBusy(*event*)	None
onCtiClear(*event*)	None
onDropped(*event*)	None
onError(*event*)	None
onInfo(*event*)	None
onLoggedIn(*event*)	None
onLoggedOut(*event*)	None
onLoggingIn(*event*)	None
onLoggingOut(*event*)	None
onMediaDisabled(*event*)	None
onMediaEnabled(*event*)	None
onNotReady(*event*)	None

Callback Event Method	***Returns***
onPresenceChanged(*event*)	None
onReady(*event*)	None
onRegistered(*event*)	None
onRegistering(*event*)	None
onSettingNotReady(*event*)	None
onSettingPresence(*event*)	None
onSettingReady(*event*)	None
onSettingWorkNotReady(*event*)	None
onSettingWorkReady(*event*)	None
onStat(*event*)	None
onStat1(*event*)	None
onStat2(*event*)	None
onUnkown(*event*)	None
onUnregistered(*event*)	None
onUnregistering(*event*)	None
onWorkNotReady(*event*)	None
onWorkReady(*event*)	None

UserCaps

This section lists the constructor, fields, method, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
UserCaps()	UserCaps object

Fields

Field	*Type*
canHandleChat	Boolean
canHandleEmail	Boolean
canHandleGeneric	Boolean
canHandleVoice	Boolean
canLogin	Boolean
canLogout	Boolean
canRefreshState	Boolean
canSetNotReady	Boolean
canSetPresence	Boolean
canSetReady	Boolean
canSetWorkNotReady	Boolean
canSetWorkReady	Boolean

UserData

This section lists the constructor, constants, methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
UserData()	UserData object

Constants

The UserData class uses the following constants:

- COMPONENT
- DESCR
- ISCRIPTPROGRAMNAME
- ICTYPE
- MARKET
- MENU
- REFERENCEID
- TARGET
- URLABS
- URLREL

Methods

Method	*Returns*
add(*key*, *value*)	None
addKeyValue(*key*, *value*)	None
getValue(String key)	String
remove(*key*)	None
getKeys( )	String array

UserStatistics1

This section lists the constructor, fields, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
UserStatistics1()	UserStatistics1 object

Fields

Field	*Type*
availableCost	String
ctiBusy	String
currentQueue	String
mostRecentTaskData	String
mostRecentTaskId	String
numTaskAccepted	String
numTasksDone	String
numTasksUnassigned	String
presenceText	String
reasonFlag	String
state	String
timeInCurrentState	String
timeSinceLoggedIn	String

UserStatistics2

This section lists the constructor, fields, methods, callback event methods, and returns (if applicable) associated with this class.

Constructor

Constructor	*Returns*
UserStatistics2()	UserStatistics2 object

Fields

Field	*Type*
currentQueue	String
timeIdle	String
timeInCurrentState	String
timeNotReady	String
timeSinceLogin	String
totalTimeAvailable	String
totalTimeUnavailable	String

Chapter 15

Installing Digital Certificates for REN SSL

Installing Digital Certificates

Digital certificates are required to provide client and server authentication in Real-time Event Notification (REN) server using SSL. A digital certificate is an electronic means of establishing your credentials for web or business transactions that are issued by a certification authority (CA). The CA is a trusted third party who signs and issues the certificates for users after verifying their authentication using secure means.

The Installing for REN SSL topic presents a sample way of installing and configuring REN using SSL. PeopleSoft customers may have their own means of obtaining and installing for REN server with SSL enabled.

This section outlines the basic steps to install. Before installing, you must create the application server domain.

Note: The application server domain must have write permissions. All certificates are stored under <PS_HOME>/appserv/<domain name>. The cacerts file has write permissions under <PS_HOME>/JRE/lib/security.

The following overview lists the steps that are required to install. The subsequent sections describe each step in detail.

To install and configure REN SSL:

1. Install the CA server certificate.

   See Installing the CA Server Certificate.

2. Install the REN server certificate.

   See Installing the REN Server Certificate.

3. Configure .

   See Configuring Digital Certificates.

4. Import certificates into the Java keystore for any REN Java Clients.

   See Importing Certificates in Java Keystore for REN Java Clients.

5. Configure the REN server.

   See Configuring the REN Server.

6. Configure REN clusters.

See Configuring REN Clusters.

7. Install certificates for local node.

   See Installing Certificates for Local Node.

8. Generate the client certificate.

   See Generating the Browser Client Certificate

9. Install PSMCAPI certificates.

   See Installing PSMCAPI Certificates.

10. Configure an external keystore.

    See Configuring External Keystore in REN Server.

11. Configuring UQSRV and MCFLOG logs.

    See Configuring UQSRV and MCFLOG logs.

Installing the CA Server Certificate

To install the CA server certificate:

1. Generate the RSA private key for the certificate authority.
2. Generate the Certificate Signing Request (CSR) for the certificate authority.
3. Generate the PEM file.

   Note: If a CA certificate already exists in PEM format, the preceding three steps can be omitted.

4. Import the CA Certificate in PEM format using **PeopleTools** > **Security** > **Security Objects** > **Manage Digital Certificates.**

The preceding steps are explained in detail in the section Configuring Digital Certificates.

See “Configuring Digital Certificates” (Security Administration), “Implementing Client Authentication” (Security Administration).

Installing the REN Server Certificate

To install the REN server certificate:

1. Generate REN server CSR using **PeopleTools** > **Security** > **Security Objects** > **Manage Digital Certificates.**
2. Get the CSR signed by a CA.

   Note: The certificate must be in PEM format.

3. Import the certificate into **PeopleTools** > **Security** > **Security Objects** > **Manage Digital Certificates.**

The preceding steps are explained in detail in the section Configuring Digital Certificates.

See Configuring Digital Certificates.

Configuring Digital Certificates

Before configuring digital certificates, you must generate the private keys, CSR, and PEM file.

To configure digital certificates:

1. Select **PeopleTools** > **Security** > **Security Objects** > **Manage Digital Certificates.**
2. Click **+.**
3. Select *ROOTCA* from the **Type** drop-down list box.
4. Enter an alias name for the CA in **Alias,** and click **Add Root.**

   The Add Root Certificate dialog box appears.
5. Open the ca.pem file.

   The root CA certificate is generated.
6. Copy the contents of the ca.pem file, paste them into the Add Root Certificate dialog box, and click **OK.**
7. Click **+.**
8. Select *Cert* from the **Type** drop-down list box.
9. Enter an alias name in **Alias,** such as PSFTCA.
10. Click **Add Root**.
11. Select the CA certificate alias of step 4 from the **Issuer Alias** lookup button.
12. Click **Request.**

    The Request New Certificate dialog box appears.
13. Complete the **Common Name, Org Unit (organization unit), Organization, Locality, State/ Province, Country, Algorithm, Key Size, Email Address,** and **Challenge Pswd**fields.

    Note: The common name must be the machine name of the REN server machine, for example, PTA112.example.com, where PTA112 is the machine name and .example.com is the domain name.

    ---
14. Click **OK.**

    The Certificates Signing Request dialog box appears.
15. Copy and paste the text from the Certificates Signing Request dialog box, and save the text in a file named ren.csr in <PS_HOME>\appserv\<domain name>\.

16. Click **OK.**

    The **Import** link appears.

17. Submit ren.csr to the CA that issued the selected root certificate.

    The CA may send you the signed public key certificate by email or require you to download it from a specified web page.

18. Open the saved certificate file in a text editor, and then highlight and copy its entire contents.

19. Select **PeopleTools** > **Security** > **Security Objects** > **Manage Digital Certificates.**

20. Click **Import.**

    The Import Certificate page appears.

21. Paste the copied certificate content into the long edit box, and click **OK.**

See “Implementing Node Authentication” (Integration Broker Administration), “Implementing Nonrepudiation” (Integration Broker Administration).

Importing Certificates in Java Keystore for REN Java Clients

In order for REN Java clients, such as a queue server, MCF Log server, or CTI adapter, to communicate with an SSL-enabled REN server, these clients need to have SSL certificates available in a Java keystore. You must import the REN server certificates into the keystore used by each REN Java client. For the Queue servers and MCF Log servers, this will be the keystore used by the JRE in the PeopleSoft domain where the queue server and MCF Log server are running. For CTI adapters, this will be the JRE used by the adapter.

The following example shows how to install the REN server certificates into the JRE trust store for a queue server and MCF log server. You will need to repeat this procedure for every queue server domain being used. You will also need to perform a similar procedure for CTI adapters or any other REN Java client, if you are using them.

Note: If you have configured client authentication, you will need to import client certificates into the JRE keystore.

To import certificates in Java keystore:

1. Open a command prompt.

2. Enter the following command:

```
<PS_HOME>\jre\bin\keytool -import -trustcacerts -alias <alias-name>
-file <CA Certificate Pem file> -keystore <full path to keystore used by the R⇒

EN Java client>
-storepass <password>
```

   Example:

```
<PS_HOME>\jre\bin\keytool -import -trustcacerts -alias PSFTCA -file ca.pem
-keystore <PS_HOME>\jre\lib\security\cacerts -storepass changeit
```


Note: You will get an error message, `sslv3 alert certificate unknown`, if the certificate is not imported correctly.

See “Implementing Client Authentication” (Security Administration).

Configuring the REN Server

To configure the REN server for SSL:

1. Select **PeopleTools** > **REN Server Configuration** > **REN Server Definition.**

   The REN Server Definition page appears.

2. Select the **SSL Only** check box.
3. Select your REN Server certificate from the **Certificate Alias** drop-down list box.
4. Click *Save.*

Related Links

Defining REN Servers

Configuring REN Clusters

To configure REN clusters:

1. Select **PeopleTools** > **REN Server Configuration** > **REN Server Cluster.**

   The REN Server Cluster page appears.

2. Update the REN server cluster URL using https and the SSL port.
3. Update the REN server browser URL using https and the SSL port.
4. Click **Save.**

Related Links

Defining a REN Server Cluster

Installing Certificates for Local Node

Apart from the CA and REN server certificates, client authentication requires local node certificates, a client certificate for the browser, and a PSMCAPI certificate.

To install certificates for the local node:

1. Select **PeopleTools** > **Security** > **Security Objects** > **Manage Digital Certificates.**
2. Click **+.**
3. Select *Local Node* from the **Type** drop-down list box.

4. Enter the local node name in **Alias,** and click **Add Root.**
5. Select the alias of the CA certificate from the **Issuer Alias** lookup button.
6. Click **Request.**
7. Complete the **Common Name, Org Unit, Organization, Locality, State/Province, Country, Algorithm, Key Size, Email Address,** and **Challenge Pswd** fields.

   Note: The common name must be the machine name of the REN server machine, for example, PTA112.example.com, where PTA112 is the machine name and .example.com is the domain name.
8. Click **OK.**

   The Certificates Signing Request dialog box appears.
9. Copy and paste the text from the Certificates Signing Request dialog box, and save the text in a file.
10. Click **OK.**

    The **Import** link appears.
11. To obtain your local node certificate, submit the certificate request text to the CA that issued the selected root certificate.

    The process of obtaining digital certificates varies, depending on the CA. Typically, a CA requires you to paste the content of the PEM-formatted CSR into a form that you submit online. The CA may send you the signed public key certificate by email or require you to download it from a specified web page.
12. Open the saved certificate file in a text editor, and then highlight and copy its entire contents.
13. Select **PeopleTools** > **Security** > **Security Objects** > **Manage Digital Certificates.**
14. Click the **Import** link.

    The Import Certificate page appears.
15. Paste the copied certificate content into the long edit box, and click **OK.**

See “Implementing Node Authentication” (Integration Broker Administration), “Implementing Nonrepudiation” (Integration Broker Administration).

Generating the Browser Client Certificate

You can generate the client certificate by openssl or keytool in P12 format and import it in the browser. Importing the certificates depends on the browser.

The following steps are an example of generating a client certificate using openssl. Clients can also use keytool or Microsoft CA to generate the client certificate

To generate the client certificate using openssl:

1. Generate the private key

```
openssl genrsa -aes256 -out renclient.key
```


2. Use the private key to generate a certificate request. When you create this request, ensure that the certificate is issued to your identity. This may mean using your email address as the common name when prompted.

   ```
   openssl req -new -key renclient.key -days 365 -out renclient.csr
   ```

3. Send the CSR file to CA to get the certificate.

   Download the certificate as a base64-encoded renclient.cer file. Note that base64-encoded cer files are PEM files.

   Also download the CA's certificate as RootCA.cer, which is also base64-encoded.

4. Generate a p12 archive containing the downloaded certificates and the private key.

   ```
   openssl pkcs12 -export -out renclient.pfx -inkey renclient.key -in RootCA.cer ⇒
   -in renclient.cer
   ```

   The renclient.pfx contains the private key.

   Note: The files created when you follow steps 1 through 4 are renclient.key, renclient.csr, renclient.cer, RootCA.cer, and renclient.pfx.

   ---

5. Next, install the RootCA certificate. On Windows-based machines, this is done with **certmgr**.

6. Install the renclient.pfx file.

   Note: Similar to step 5, on Windows-based machine, you can install renclient.pfx file with **certmgr** by remembering to use the appropriate certificate storage for personal certificates.

   ---

7. Go to the REN Server Buffer Test page, with client certificates enabled for the REN server. The browser will prompt you to choose a client certificate to use.

Installing PSMCAPI Certificates

To install PSMCAPI certificates:

1. Generate a private key in the keystore using the following command:

   ```
   keytool -genkeypair -alias <fully qualified machine name> -keystore <full path⇒
    to the psmcapi keystore>
   ```

   You will be prompted to enter various inputs. Ensure that the name field is fully qualified name of the machine where psmcapi is running. This command creates the keystore if it does not already exist.

   Example:

   ```
   >keytool -genkeypair -alias example.oracle.com -keystore psmcapi_keystore
   Enter keystore password:
   Re-enter new password:
   What is your first and last name?
     [Unknown]:  example.oracle.com
   What is the name of your organizational unit?
     [Unknown]:  PeopleTools
   What is the name of your organization?
     [Unknown]:  PeopleSoft
   What is the name of your City or Locality?
   ```



```
  [Unknown]:  Pleasanton
What is the name of your State or Province?
  [Unknown]:  CA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=example.oracle.com, OU=PeopleTools, O=PeopleSoft, L=Pleasanton, ST=CA, C⇒

=US correct?
  [no]:  yes

Enter key password for <example.oracle.com>
        (RETURN if same as keystore password):
```

2. Generate the certificate request using the following command:

```
keytool -certreq -alias <same alias as in step 1> -file <output filename> -key⇒

store <full path to the Java keystore used by the CTI adapter>
```

   You will be prompted for the keystore password.

   Example:

```
>keytool -certreq -alias example.oracle.com -file example.csr -keystore psmcap⇒

i_keystore
Enter keystore password:

>cat example.csr
-----BEGIN NEW CERTIFICATE REQUEST-----
MIICsDCCAm4CAQAwezELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAkNBMRMwEQYDVQQHEwpQbGVhc2Fu
dG9uMRMwEQYDVQQKEwpQZW9wbGVTb2Z0MRQwEgYDVQQLEwtQZW9wbGVUb29sczEfMB0GA1UEAxMW
...
yYcbMAsGByqGSM44BAMFAAMvADAsAhRAU7bb602ca7ifVVHk1hwAJkTSNgIUCJP2oI3Sf4eD/7Ri
gbHVPF4jnY4=
-----END NEW CERTIFICATE REQUEST-----
```

3. To obtain your certificate, submit the CSR to the CA that issued the selected root certificate.

4. If necessary, import the Root CA certificate into the keystore. It is necessary if this certificate is not already in the Java truststore.

```
keytool -import -alias <Root CA alias> -file <Root CA cert> -keystore <full pa⇒

th to the Java keystore used by the CTI adapter>
```

   You will be prompted for the keystore password. Respond with *yes* when asked if you wish to trust this certificate.

   Example:

```
>keytool -import -file RootCA.cer -alias RootCA -keystore psmcapi_keystore
Enter keystore password:
Owner: CN=example.oracle.com, OU=PeopleTools, O=PeopleSoft, L=Pleasanton, ST=C⇒

A, C=US
...
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

5. Import the signed certificates into the psmcapi keystore using the following command:

```
keytool -import -alias <same alias as in step 1> -file <certificate file> -key⇒
```



```
store <full path to the Java keystore used by the CTI adapter> -trustcacerts
```

You will be prompted for the keystore password.

Example:

```
>keytool -import -file abc.cer -alias example.oracle.com -keystore psmcapi_key⇒

store -trustcacerts
Enter keystore password:
Certificate reply was installed in keystore
```

These are the certificates needed in your psmcapi keystore. The client certificate (in this example, *example.oracle.com*) and the root CA certificate. The root CA certificate can also be stored in a separate truststore.

```
>keytool -list -keystore psmcapi_keystore
Enter keystore password:
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 2 entries

example.oracle.com, Oct 14, 2020, PrivateKeyEntry,
Certificate fingerprint (SHA-256): AC:24:48:06:64:C8:F4:45:B3:92:39:72:DB:64:70:C1:⇒

FE
rootca, Oct 23, 2020, trustedCertEntry,
Certificate fingerprint (SHA-256): AE:34:20:21:A2:3B:5A:79:77:AD:46:E8:E6:78:B0:3A:⇒

5E
```

See “Implementing Client Authentication” (Security Administration).

Configuring External Keystore in REN Server

Certificates that are used to secure a Web server can be reused to secure a REN server. The Java keystore that stores these certificates can be converted to a PKCS #12 keystore. Subsequently, you can configure the REN server to use the newly created external PKCS #12 keystore when you define a REN server the SSL and the UseExternalKeystore options are selected.

To convert a Java keystore to a PKCS #12 keystore, you can use the PSExportToPKCS12.bat file that is delivered by Oracle.

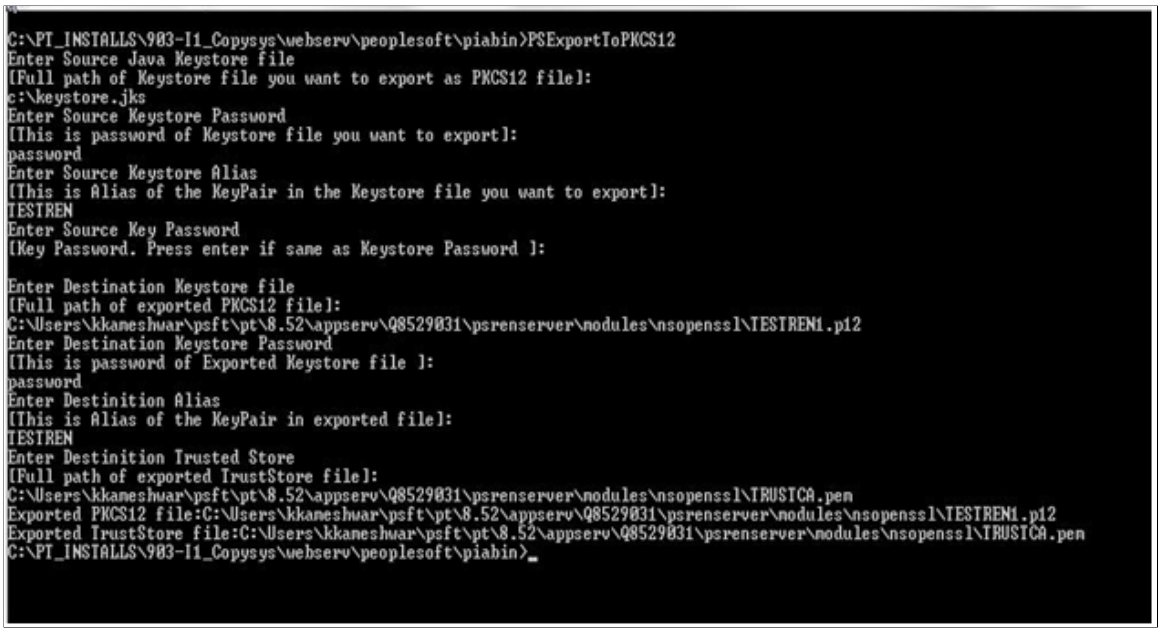
1. Access the piabin directory in your web server installation and locate the PSExportToPKCS12.bat file.

   Note: Ensure that the JAVA_HOME environment variable is set correctly.

2. Run the PSExportToPKCS12.bat in Windows or PSExportToPKCS12.sh in UNIX.

   Note: The PSExportToPKCS12.bat and PSExportToPKCS12.sh scripts must be run using a version of the Sun Java Runtime environment compatible with your PeopleTools installation.

This example illustrates the conversion of Java keystore to PKCS #12 keystore using the PSExportToPKCS12.bat file.



Note: The PSExportToPKCS12.bat file must be run prior to defining a REN server in the REN Server Configuration page. The file must not be run during the boot sequence of the application server.

3. Enter values for the following:

   - Source Java Keystore — the full path of Java Keystore.
   - Source Keystore Password — the password of the Java Keystore
   - Source Keystore Alias — alias of the KeyPair in the Java Keystore.
   - Source Key Password — if the password is the same as Source Keystore Password, you can leave it blank.
   - Destination Keystore — the full path of exported PKCS #12 keystore including the file extension (*.p12* is preferred).
   - Destination Keystore Password — the password of exported keystore.
   - Destination Alias — alias of the KeyPair in exported keystore.
   - Destination Trusted Store — the full path of the exported TrustStore file including the file extension (*.pem* is preferred).

   The Java Keystore is converted to PKCS #12 keystore.

Configuring UQSRV and MCFLOG logs

Server authentications involving SSL may encounter errors that are not logged in the generic error logs. Additional logging can be activated from the Tuxedo domain.

The new logging will be sent to a new UQ server and MCFLog server log, controlled by the JVM command line in the psappsrv.cfg file. Additionally, other REN Java clients, such as CTI adaptors can use the same logging structure.

There are two new sections in the psappsrv.cfg file, one for PSUQSRV and one for PSMCFLOG. Inside those sections are instructions for this new logging:

```
;To turn on SSL and other additional logging, add

; -DRENClient.log=true -DRENClient.log.level=All

;Available log levels are:

; None, Emergency, Alert, Critical, Error, Warning, Notice, Info, Debug,and All

;For example: JavaVM Options=-Xmx256m -Xms512m

JavaVM Options=-Dxdo.ConfigFile=%PS_HOME%/appserv/xdo.cfg -Xms256m -Xmx512m
```

Add -DRENClient.log=true to turn on logging. Then -DRENClient.log.level can be set to one of the listed debugging levels to capture more or less debugging information. For most debugging purposes, "All" is the appropriate choice.

Add -Dhttps.protocols=TLSv1.3 to require the UQ and MCF Log servers to communicate with the REN server in TLSv1.3, after updating the REN server configuration files.

For example: JavaVM Options=-Dxdo.ConfigFile=%PS_HOME%/appserv/xdo.cfg -Xms256m -Xmx512m -Dhttps.protocols=TLSv1.3.

See Configuring REN Servers.

Chapter 16

PSRENCONFIG Quick Reference

PSRENCONFIG Parameters

This section lists the REN server configuration parameters. Each parameter listed below is grouped by its section within the psrenconfig.txt file. Identically named parameters in different sections will affect different functions of the REN server, and are independent of each other.

The list of parameters described here is not complete. For a complete list of parameters, see the psrenconfig.txt file created with your appserver domain.

Note: Default values may differ between releases including patch releases. For the most up to date default value information, see the psrenconfig.txt file in your installation. These default values may not be the recommended settings for your specific type of installation.

Global Parameters

Parameter	*Default Value*	*Description*
listenbacklog	512	The maximum length of the pending connection queue on the listening sockets.

Server Parameters

Parameter	*Default Value*	*Description*
maxthreads	256	The maximum number of connection threads that will be created.
minthreads	5	The starting number of connection threads.

Event Routing

Parameter	*Default Value*	*Description*
reaper_interval	3600	How frequently, in seconds, the REN server will do internal cache cleanup.

Parameter	*Default Value*	*Description*
default_kn_expires	“+3600”	How quickly, in seconds, cached items will, by default, become ready for cleanup.