Oracle® Communications Contacts Server

System Administrator's Guide

Release 8.0

E56051-04

May 2021

ORACLE

Oracle Communications Contacts Server System Administrator's Guide, Release 8.0

E56051-04

Copyright © 2015, 2021, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

4 Managing Contacts Server

5 Monitoring Contacts Server

6 Improving Contacts Server Performance

7 Migrating Information to Contacts Server

8 Managing the Contacts Server Database

9 Backing Up and Restoring Files and Data

10 Troubleshooting Contacts Server

11 Using Contacts Server Notifications

A Contacts Server Command-Line Utilities

B Contacts Server Configuration Parameters

Preface

This guide explains how to administer Oracle Communications Contacts Server and its accompanying software components.

Audience

This document is intended for system administrators whose responsibility includes Contacts Server. This guide assumes you are familiar with the following topics:

- Oracle Communications Calendar Server
- Oracle Communications Messaging Server
- Oracle GlassFish Server or Oracle WebLogic Server
- Oracle Directory Server Enterprise Edition and LDAP
- System administration and networking
- General deployment architectures

Related Documents

For more information, see the following documents in the Contacts Server documentation set:

- *Contacts Server Installation and Configuration Guide*: Provides instructions for installing and configuring Contacts Server.
- *Contacts Server Release Notes*: Describes the new features, fixes, known issues, troubleshooting tips, and required third-party products and licensing.
- *Contacts Server Security Guide*: Provides guidelines and recommendations for setting up Contacts Server in a secure configuration.
- *Contacts Server RESTful Protocol Guide*: Describes the RESTful protocol that enables HTTP clients to fetch, add, and edit address book related data that is stored by Contacts Server.

Nomenclature

The following nomenclature is used throughout the document.

Convention	Meaning
Application Server	The term **Application Server** or **application server** is used in this document to refer to either GlassFish Server or WebLogic Server. **Supported Application Server**: Oracle Communications Contacts Server 8.0.0.4.0 and previous releases were deployed on GlassFish Server, which is no longer supported by Oracle. For that reason, Contacts Server 8.0.0.5.0 and beyond are only supported on Oracle WebLogic Server. Oracle strongly recommends that you upgrade your Contacts Server environments to release 8.0.0.5.0 or higher and migrate to WebLogic Server to receive full Oracle support.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.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 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

1

Contacts Server System Administration Overview

This chapter provides an overview of Oracle Communications Contacts Server, and describes the basic administration tasks and tools used to perform those tasks.

About Contacts Server

Contacts Server enables end users to store and retrieve contact information such as name, email address, photo, birthdays, and any other information that relates to the contact. Contacts Server supports all properties defined in the vCard specification, RFC 6350, available on the IETF website at:

http://tools.ietf.org/html/rfc6350

Contacts Server provides a Network Address Book that facilitates centralized storage and access of contacts for a large number of users. Being full-featured, it not only provides contact creation, management and searching capabilities along with multiple group and multiple address book support, but includes features that enterprises demand, such as Global Address List integration and address book sharing.

The following sections describe Contacts Server in more detail:

- Managing Address Books
- Managing Contacts
- Contacts Server Support for Corporate Directory
- Contacts Server Support for Industry Standards

Managing Address Books

Contacts Server enables end users to own multiple address books. Each address book requires a display name when created. You can list, modify, and delete address book entries and their properties. Contacts Server creates the user's default address book with a special contact called the personal contact card (PCC). The PCC is the single personal contact entry for the user. Each user's PCC is populated by Contacts Server with the user's first and last name. Only the user can view or modify the PCC.

Users can share their address books for other users to subscribe to. Contacts Server uses Access Control Lists (ACLs) to control subscription. ACLs consist of one or more Access Control Entries (ACEs), which are strings that grant a particular level of access to a particular entity. Access rights can be specified for an individual or an LDAP group. A share notification email is sent to the address book subscriber when access

rights are granted. For more information about address book subscription and access rights, see "About Controlling Access to Address Books".

Managing Contacts

Contacts Server enables users to manage their contacts according to their individual needs. Contacts Server users can create contact groups, classify contacts, and perform actions on those groups. Groups consist of contacts from that address book, another local address book, a shared address book, or a corporate address book. Groups can also consist of external members. Contacts Server represents contacts by using URIs. Contacts Server uses a UID for contacts defined on the same server, and email addresses for external members, including those in a corporate directory.

Note: Contacts Server allows only one photo per contact, as opposed to multiple photos per contact.

In addition, Contacts Server provides the capability to import and export contact information.

Contacts Server supports the following formats for importing contacts:

- Outlook CSV
- Thunderbird CSV
- Thunderbird LDIF
- vCard 3.0

Contacts Server supports the vCard 3.0 and CSV formats for exporting contacts.

Note: The recommended export format is vCard 3.0. Only use CSV if vCard 3.0 is unavailable.

For more information on importing and exporting contacts, see "Importing Contact Groups" and "Exporting Contact Groups".

Contacts Server Support for Corporate Directory

Contacts Server supports read-only access to corporate directory listings stored in Oracle Directory Server Enterprise Edition (Directory Server). You can configure Contacts Server for a default corporate directory. In addition, you can define additional per-domain corporate directories. LDAP data is translated to vCard format for output. For more information on configuring the corporate directory, see "Managing the Corporate Directory".

Contacts Server Support for Industry Standards

Contacts Server is based on standards. Table 1–1 lists the standards that Contacts Server uses.

These are internet standards, published in RFCs approved by the Internet Engineering Task Force (IETF).

Table 1–1 Contacts Server Supported Standards

Standard Name	Standard Details
vCard	■ vCard (RFC6350) ■ vCard 3.0 (RFC2426)
CardDAV	■ Based on RFC6352 for access control ■ Supports draft about CardDAV directory ■ Supports RFC6764, Locating Services for CardDAV
Other	■ HTTP and HTTP Auth (RFC 2616 & RFC2617) ■ WebDAV (RFC4918, RFC5689 & RFC3744) ■ Collection Synchronization for WebDAV (RFC6578)

Overview of Contacts Server Administration Tasks

A Contacts Server administrator is responsible for the day-to-day tasks of maintaining and managing Contacts Server and its users. The tasks also include managing Contacts Server components, application server, and potentially other Unified Communications Suite components.

You perform the following tasks as a Contacts Server administrator:

- Stopping and starting Contacts Server
- Managing user accounts, address books, and contacts
- Monitoring Contacts Server
- Tuning Contacts Server performance
- Migrating data to Contacts Server
- Managing the Contacts Server back-end database
- Backing up and restoring files
- Troubleshooting Contacts Server

About Contacts Server Administration Tools

Contacts Server is deployed on an application server domain.

When GlassFish Server is used as the container, you can use the GlassFish Server Administration Console and **asadmin** command to manage the Contacts Server web container. See the GlassFish Server documentation for more information.

When WebLogic Server is used as a container, you can use WebLogic Server Administration Console to manage the Contacts Server web container. See the WebLogic Server documentation for more information.

Contacts Server provides a number of command-line utilities for administering the server. These utilities run under the parent command, **davadmin**. For more information, see "Contacts Server Command-Line Utilities".

Directory Placeholders Used in This Guide

Table 1–2 lists the placeholders that are used in this guide:

Table 1–2 Contacts Server Directory Placeholders

Placeholder	Directory
ContactsServer_home	Specifies the installation location for the Contacts Server software. The default is **/opt/sun/comms/nabserver**.
GlassFish_home	Specifies the installation location for the Oracle GlassFish Server software. The default is **/opt/glassfish3/glassfish**.
WebLogic_home	The base directory in which Oracle WebLogic Server software is installed.
GlassFish_Domain	Oracle GlassFish Server domain in which Contacts Server is deployed. For example, **GlassFish_home/domains/domain1**
WebLogic_Domain	Oracle WebLogic Server domain in which Contacts Server is deployed. For example, *WebLogic_home*/**user_projects/domains/base_domain**. **Note**: In case of WebLogic Server, it must have at least one Managed Server instance configured and the Managed Server instance must be hosting the Contacts Server.
AppServer_Domain	Domain of the application server in which Contacts Server will be deployed. Domain refers to either *Glassfish_Domain* or *Weblogic_Domain*.

2

Stopping and Starting Contacts Server

This chapter explains how to stop and start Oracle Communications Contacts Server.

Overview of Stopping and Starting Contacts Server

Stopping and starting Contacts Server involves stopping and starting processes and databases on the Contacts Server front-end and back-end hosts.

To stop and start the Contacts Server process on the front-end hosts, you must stop and start the application server domain in which Contacts Server is deployed.

To stop and start the Contacts Server database on the back-end hosts, you use the appropriate MySQL or Oracle Database command. See the following documentation for more information:

- "Starting and Stopping MySQL Automatically" in *MySQL 5.5 Reference Manual*
- "Stopping and Starting Oracle Software" in *Oracle Database Administrator's Reference 19c for Linux and UNIX-Based Operating Systems*

When you start Contacts Server, you must first start the Contacts Server back-end database hosts, as well as the remote document stores, before starting the Contacts Server front-end hosts.

Stopping and Starting Contacts Server

The following examples show how to stop and start Contacts Server deployed on GlassFish Server and WebLogic Server.

For GlassFish Server:

Example of a default GlassFish Server installation with Contacts Server deployed in domain1:

- To stop Contacts Server:

  ```
  GlassFish_home/bin/asadmin stop-domain domain1
  ```

- To start Contacts Server:

  ```
  GlassFish_home/bin/asadmin start-domain domain1
  ```

For WebLogic Server:

You can stop or start the domains in WebLogic Server Administration Console. You can also stop or start the domains using the scripts provided in the **bin** directory of the domain. You should restart the Administration Server and Managed Server on which Contacts Server is deployed. For more information, see the discussion about starting

and stopping servers in Administering Server Startup and Shutdown for Oracle WebLogic Server.

Stopping and Starting the Remote Document Store Server

The Contacts Server document store is used to store and retrieve *large data*, such as photos and logos.

To stop and start the Contacts Server remote document store server, use the **stop-as** and **start-as** commands.

- To stop the remote document store server:

  ContactsServer_home/sbin/stop-as

- To start the remote document store server:

  ContactsServer_home/sbin/start-as

3

Managing Users, Accounts, Address Books, and Contacts

This chapter describes how to set up and manage Oracle Communications Contacts Server users, accounts, address books, and contacts.

Provisioning Contacts Server Users

This section describes how to provision Contacts Server users and contains the following topics:

- Provisioning Contacts Server Overview
- Provisioning Contacts Server Users by Using Delegated Administrator

Provisioning Contacts Server Overview

Contacts Server uses Directory Server to store and retrieve user and resource information and to perform authentication. Contacts Server does not add or modify LDAP data. Contacts Server data (such as address book and contact information) is stored in an SQL database, which can be either MySQL Server or Oracle Database.

By default, Contacts Server automatically creates the necessary entries in the SQL database for users upon their initial Contacts Server login. However, you must also perform some basic LDAP user provisioning for users to be able to access Contacts Server services, and for Contacts Server automatic account creation to work. You can provision Contacts Server users in the Directory Server LDAP by using either Delegated Administrator or LDAP tools.

You must provision Contacts Server users so that Contacts Server can automatically create accounts and users can access Contacts Server services. You must provision users with the following attributes:

- An email attribute, such as **mail**.
- A unique ID attribute corresponding to the value of the server configuration parameter, **davcore.uriinfo.permanentuniqueid**. The default value is **davUniqueId**. Be sure to also index the attribute used for **davcore.uriinfo.permanentuniqueid**, as Contacts Server performs searches on it.

To define these attributes, the corresponding object classes must be present in the Directory Server LDAP. The Communications Suite **comm_dssetup** script adds the necessary object classes. The **davEntity** object class defines the **davUniqueId** attribute. If your deployment consists of multiple back-end databases, you must also define the store ID attribute. The **nabUser** object class defines the default store ID attribute. The default value of the store ID is **nabStore**.

For more information, see the topic on Contacts Server LDAP object classes and attributes in *Communications Suite Schema Reference*.

Denying Users Access to Services

By default, if you provision Contacts Server users for email and unique ID attributes (and the store ID attribute when multiple back-end databases are deployed), users have a status of **active**. The **active** status enables users to access Contacts Server services. To deny Contacts Server services to users, you specify a value of either **inactive** or **deleted** for the user's **nabStatus** attribute.

About Migrating Users

If you have a co-existent deployment of both Contacts Server and Convergence WABP, and are migrating users to Contacts Server, you must update the user's LDAP data once the user is marked for migration and taken offline for migration. You can only migrate the user at that point. Contacts Server uses an LDAP attribute to determine if a user has been migrated. By default, the **nabStore** attribute is used, but you can choose another attribute if desired. In a single back-end deployment, this attribute must be added with the value of **defaultbackend**. In a multiple back-end deployment, the value must be the logical back-end ID for the database where the user's data resides after migration. Again, the object class that defines the **nabStore** attribute is **nabUser**.

Provisioning Contacts Server Users by Using Delegated Administrator

Starting with version 7.0.0.10.0, Oracle Communications Delegated Administrator enables you to provision Contacts Server users. See *Delegated Administrator System Administrator's Guide* for more information.

About Controlling Access to Address Books

Contacts Server uses Access Control Lists (ACLs) that you define to control access to address books. (ACLs are also used to control access to accounts.) An ACL applies to a single address book (or account). An ACL consists of one or more Access Control Entries (ACEs), which are strings that grant a particular level of access such as read-only access or read and write access. ACEs collectively apply to the same address book. Multiple ACE strings can apply to a single address book.

You can also define ACEs for LDAP groups. Groups and users are each represented by a mail address. An access right granted to a group is effective for all members of the group.

Access to address books is denied unless explicitly granted. Some access rights are predefined and cannot be changed. For example, Contacts Server gives users full access to their own address books.

ACEs are specified in the following format:

ace_principal:*right*

where:

- *ace_principal* can be one of the following values:
  - @ grants access to all users.
  - @*domain* grants access to all users on a specific domain. Example:

    @example.com
  - *user*@*domain* grants access to a specific user. Example:

Bob@example.com

- *group@domain* grants access to a set of users who belong to a defined group. Example:

  MyGroup@example.com

- *rights* can be one of the following values:
  - **n** (none) denies access
  - **r** (read) grants read-only access
  - **w** (write) grants read and write access
  - **a** (all) grants all levels of access

You set Contacts Server access rights by using the **davadmin** command with the **acl** property on the command line, or buy using the Convergence client. The **acl** property is a semicolon-separated list of ACE strings.

ACEs function in the following way:

- More specific access rights override less specific access rights. For example, access rights granted to a particular user are more specific than rights granted to a user as member of a group. The user-specific access rights override the access rights granted through group membership.
- Access rights granted to all users (using the @ value) are considered least specific.
- When a user is a member of multiple groups, that user is given the highest level of access granted by any one of the groups.
- Contacts Server access control ignores nesting levels within each group.

When determining group membership for access rights, Contacts Server considers only the users' domain name (DN) defined in the LDAP directory. The DN value may be set in the **uniquemember** attribute, or set in the **memberurl** attribute as a URL that resolves to the DN of the group.

Managing Accounts

This section describes tasks related to managing Contacts Server accounts.

Managing accounts includes:

- Enabling and Disabling Automatic Account Creation
- Creating Accounts with Default Properties Automatically Upon Login
- Manually Creating Accounts
- Listing Accounts
- Managing Email Notifications
- Deleting Accounts
- Subscribing to and Unsubscribing from Address Books

You manage Contacts Server accounts by using the **davadmin** command. You authenticate the **davadmin** command with the application server administrative user name and password to allow to communicate with the server or database. You can use the **davadmin passfile** operation to store the necessary passwords in an encrypted *wallet* for use by subsequent **davadmin** commands. If you do not store passwords in the wallet, then you must enter them by using a no-echo prompt on the command line.

See the discussion about **passfile** operation in *Contacts Server System Administrator's Guide* for more information.

Enabling and Disabling Automatic Account Creation

You can enable or disable, on a system-wide basis, automatic account creation. When automatic account creation is enabled, users' accounts are automatically created for them when they first log in to Contacts Server.

To Enable Automatic Account Creation

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

```
davadmin config modify -o davcore.autocreate.enableautocreate -v true
```

To Disable Automatic Account Creation

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

```
davadmin config modify -o davcore.autocreate.enableautocreate -v false
```

Creating Accounts with Default Properties Automatically Upon Login

To create accounts with default contacts properties automatically when users log in:

1. Provision users in LDAP with the minimum functionality that Contacts Server requires.

   See "Provisioning Contacts Server Overview" for more information.

2. Use the **davadmin config** command to set any of the following autocreate parameters to customize your deployment:

   - **davcore.autocreate.displaynameattr**
   - **davcore.autocreate.emailnotificationaddressattr**
   - **davcore.autocreate.enableemailnotification**

   For more information about these parameters, see "Contacts Server Configuration Parameters".

3. Enable account autocreation.

   See "To Enable Automatic Account Creation."

4. Provide users instructions for logging in to Contacts Server.

Manually Creating Accounts

Use the **davadmin account create** command to create Contacts Server accounts. You can specify certain account properties on the command line. You can create accounts one at a time or in batch mode by using the **-f** *file* option. When you use **-f**, you specify a file of accounts that you create. The file format is *account*:*property_list*, where *property_list* is optional and contains a comma separated list of *property*=*value* fields.

Users must be already provisioned in the LDAP Directory Server before you can create the Contacts Server account. See "Provisioning Contacts Server Overview" for more information.

> **Tip:** You can customize accounts by configuring them with specific properties before users initially log in to access Contacts Server.

To manually create a single account:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin account create -a account
   ```

   Where:

   account is the user account.

   For example, to create the account **john.smith@example.com**:

   ```
   davadmin account create -a john.smith@example.com
   ```

See "davadmin account" for more information on creating an account with specific account properties.

Listing Accounts

Use the **davadmin account list** command to view all accounts or properties of a specific account.

To List All Accounts

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin account list
   ```

To List Properties of an Account

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin account list -a account
   ```

   Where:

   account is the user account.

For example, to list the properties of the account **johnsmith@example.com**:

```
davadmin account list -a johnsmith@example.com
```

Managing Email Notifications

Use the **davadmin account modify** command to manage account email notifications.

To Enable Email Notification

When email notification is enabled, Contacts Server sends and email to the account owner when changes to the account are made.

To enable email notification:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin account modify -a account -y notifemail=1
   ```

   Where:

   account is the user account.

For example, to enable email notifications for the account **john.smith@example.com**:

```
davadmin account modify -a john.smith@example.com -y notifemail=1
```

To Disable Email Notification

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin account modify -a account -y notifemail=0
   ```

   Where:

   account is the user account.

For example, to disable email notifications for the account **john.smith@example.com**:

```
davadmin account modify -a john.smith@example.com -y notifemail=0
```

To Add or Remove Email Notification Recipients

Recipients of email notifications are users who have Contacts Server accounts. You add users as recipients of email notification for an account by specifying the account (email addresses) of the users to add. You remove users as recipients of email notification by specifying the accounts of the users to keep; any existing recipients you do not specify are removed.

Important: When adding recipients, be sure to also specify all existing recipients you want to keep. If you do not, they will be removed.

To add or remove recipients for email notifications for an account:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command.

   ```
   davadmin account modify -a account -y notifrecipients="recipient1 recipient2
   ..."
   ```

   Where:

   account is the user account.

   recipient1 and *recipient2* are Contacts Server accounts to receive email notifications.

For example, to add **jane.jones** and **sam.taylor** so that they receive email notifications for the account **john.smith@example.com**:

```
davadmin account modify -a john.smith@example.com -y
notifrecipients="jane.jones@example.com sam.taylor@example.com"
```

Deleting Accounts

Use the **davadmin account delete** command to remove accounts.

> **Note:** The **davadmin account delete** command removes the account from the Contacts Server database but does not remove the user from the LDAP directory.

To delete an account:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin account delete -a account
   ```

   Where:

   account is the user account.

For example, to delete an account named **john.smith@example.com**:

```
davadmin account delete -a john.smith@example.com
```

Subscribing to and Unsubscribing from Address Books

For a user to subscribe to another account's address book, you first use the **davadmin addressbook modify** command to give the subscribing user's account access rights to the other account's address book. The access level must be at least read permission. You then use the **davadmin account subscribe** command to set up the subscription itself.

To Subscribe to an Address Book

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Give the subscribing account access rights to the address book of the other account. Set the access rights in the form of an ACL:

   ```
   davadmin addressbook modify -a subscribed_to_account -y acl=ace_principal:right
   ```

   Where:

   subscribed_to_account is the user account being subscribed to.

   ace_principal is the account that is subscribing to the address book.

   right is the ACL being granted to the subscribing account.

   See "About Controlling Access to Address Books" for more information on ACI's.

4. Subscribe to the account's address book:

   ```
   davadmin account subscribe -a subscribing_account -c URI_of_subscribed_to_
   ```

```
account
```

Where:

subscribing_account is the account making the subscription request.

URI_of_subscribed_to_account is the URI of the account being subscribed to.

For example, to give **cal196@example.com** read access to **cal200@example.com**, then to subscribe **cal196@example.com** to **cal200@example.com**:

```
davadmin addressbook modify -a cal200@example.com -y acl=cal196@example.com:r

davadmin account subscribe -a cal196@example.com -c
/home/caltest200@example.com/addressbook/
```

To Unsubscribe From an Address Book

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Remove the permission, in the form of an ACL, from the subscribing-account, by setting it to none (**n**). See "About Controlling Access to Address Books" for more information on ACLs.

   ```
   davadmin addressbook modify -a subscribed_to_account -y acl=subscribing_
   account:n
   ```

   Where:

   subscribed_to_account is the account being subscribed to.

   subscribing_account is the account that has subscribed.
4. Unsubscribe from the account:

   ```
   davadmin account unsubscribe -a subscribing_account -c URI_of_subscribed_to_
   account
   ```

For example:

```
davadmin account modify -a caltest200@example.com -y acl=caltest196@example.com:n

davadmin account unsubscribe -a cal196@example.com -c
/home/caltest200@example.com/addressbook/
```

Managing Address Books

Use the **davadmin addressbook** command to create, list, modify, and delete address books.

Managing address books includes:

- Creating Address Books
- Removing Address Books
- Modifying Address Books
- Listing Address Books

Creating Address Books

Contacts Server automatically creates a user's default address book upon login, when you have set the **davcore.autocreate.enableautocreate** configuration parameter to **true**. By default, Contacts Server adds a single person contact entry (PCC) for the user to the default address book. You can create additional address books for users. When creating an address book, you can specify a display name, description, and access control instructions. If you do not supply a display name, it defaults to the address book name supplied for the **-n** option.

To manually create an address book:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin addressbook create -a account -n name
   ```

   Where:

   account is the user account.

   name is the name of the address book.

For example, to create an address book named **socialab** for the account **john.smith@example.com**:

```
davadmin addressbook create -a john.smith@example.com -n socialab
```

Removing Address Books

Use the **davadmin addressbook delete** command to remove address books from accounts. To remove multiple address books, use the **-f** *filename* option. Create *filename* with the list of address books to be deleted. In this file, do not include any blank lines, otherwise the **delete** command will fail.

To remove an address book:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin addressbook delete -a account -n name
   ```

   Where:

   account is the user account.

   name is the name of the address book.

For example, to delete an address book named **socialab** from the account **john.smith@example.com**:

```
davadmin addressbook delete -a john.smith@example.com -n socialab
```

For example, to delete multiple address books specified in the file **addressbooktodelete.txt** from the account **john.smith@example.com**:

```
davadmin addressbook delete -f addressbooktodelete.txt -a john.smith@example.com
```

Modifying Address Books

Use the **davadmin addressbook modify** command to modify an address book's display name, description, and ACLs. In addition, you can set or remove one or more ACEs from the ACL.

To Modify an Address Book

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin addressbook modify -a account -y property=value
   ```

   Where:

   account is the user account.

   property is an address book property to set.

   value is the value of the property to set.

   You can specify multiple *property=value* pairs by separating them with a comma. See "davadmin addressbook" for information about the possible properties.

For example, to give **john.smith@example.com** read (**r**) access to the account **james.jones@example.com**:

```
davadmin addressbook modify -a james.jones@example.com -y
acl=john.smith@example.com:r
```

To Set an Address Book ACE

The following example shows how to set the ACE to read (**r**) access for **james.jones** on the address book **socialab** owned by **john.smith**:

```
davadmin addressbook modify -a john.smith@example.com -n socialab -y
set-ace=james.jones@example.com:r
```

To Remove an Address Book ACE

The following example shows how to remove the ACEs for **james.jones** and **sam.taylor** from the address book named **socialab** owned by **john.smith**.

```
davadmin addressbook mofify -a john.smith@example.com -n socialab -y
remove-ace=james.jones@example.com;sam.taylor@example.com
```

Listing Address Books

Use the **addressbook list** command to display the properties of an address book, including its ACLs, number of contacts, and number of contact groups.

To List an Account's Address Books

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/sbin directory.
3. Run the following **davadmin** command:

   ```
   davadmin addressbook list -a account
   ```

   Where:

account is the user account.

For example, to list the address books for the account **john.smith@example.com**:

```
davadmin addressbook list -a john.smith@example.com
```

To List an Address Book's Properties

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin addressbook list -a account -n name
   ```

   Where:

   account is the user account.

   name is the name of the address book.

For example, to list the properties of an address book named **socialab** for the account **john.smith@example.com**:

```
davadmin addressbook list -a john.smith@example.com -n socialab
```

Managing Contacts

Use the **davadmin contact** command to list contact properties and to delete contacts.

Managing contacts includes:

- Listing Contact Properties
- Deleting Contacts

Listing Contact Properties

Use the **davadmin contact list** command to list contact properties, including vCard information and the address book to which the contact belongs.

To list a contact's properties:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin contact list -a account
   ```

   Where:

   account is the user account.

For example, to list the properties of contacts belonging to the account **john.smith@example.com**:

```
davadmin contact list -a john.smith@example.com
```

Deleting Contacts

Use the **davadmin contact delete** command to delete contacts.

To delete a contact:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin contact delete -a account -c contact
   ```

   Where:

   account is the user account.

   contact is the name of the contact to delete.

For example, to delete the contact **1406701074936-0-.vcf** from the account **john.smith@example.com**:

```
davadmin contact delete -a john.smith@example.com -c 1406701074936-0-.vcf
```

Managing Contact Groups

Contact groups enable users to organize their contacts, making it easier to work with a specific set of people. For example, a user might want to organize contacts by family, work, and soccer team. Use the **davadmin ctgroup** command to manage contact groups.

Managing groups includes:

- Creating Contact Groups
- Listing Contact Groups
- Deleting Contact Groups
- Modifying Contact Groups
- Importing Contact Groups
- Exporting Contact Groups

Creating Contact Groups

You can create multiple contact groups per address book. When you create a contact group, you can also add members to it by using the **-M** option.

To create a contact group:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin ctgroup create -a account -n name -g groupname
   ```

   Where:

   account is the user account.

   name is the name of the address book.

   groupname is the name of the contact group.

For example, to create a contact group named **myctgroup** in the **socialab** address book for the account **john.smith@example.com**:

```
davadmin ctgroup create -a john.smith@example.com -n socialab -g myctgroup
```

Listing Contact Groups

Use the **davadmin ctgroup list** command to list an address book's contact groups.

To list contact groups:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin ctgroup list -a account -n addressbook
   ```

   Where:

   account is the user account.

   addressbook is the name of the address book.

For example, to list contact groups for the account **john.smith@example.com** in the address book **socialab**:

```
davadmin ctgroup list -a john.smith@example.com -n socialab
```

Deleting Contact Groups

Use the **davadmin ctgroup delete** command to delete a contact group.

To delete a contact group:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the **davadmin ctgroup** list command to list the contact groups, and note the URI of the contact to delete. See "Listing Contact Groups" for more information about listing groups.
4. Run the following **davadmin** command:

   ```
   davadmin ctgroup delete -a account -c contactgroup_uri
   ```

   Where:

   account is the user account.

   contactgroup_uri is the contact URI that you retrieved in the previous step.

For example, to delete the contact group **410448708259-29-.vcf** from the account **johnsmith@example.com**:

```
davadmin ctgroup delete -a john.smith@example.com -c 410448708259-29-.vcf
```

Modifying Contact Groups

Use the **davadmin ctgroup modify** command to modify the contact group name, members, and email addresses of members.

To modify a contact group:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the **davadmin ctgroup list** command to list the contact groups, and note the vCard file that identifies the contact group to modify. See "Listing Contact Groups"

for information about listing groups. The output will show the full WebDAV URL, for example:

/dav/home/john.smith@example.com/addressbook/1384904616388-1758.vcf

The part that you use in the **modify** command appears after the "addressbook/" string; in this example, **1384904616388-1758.vcf**.

4. Run the **davadmin** command to modify the contact group. For example, to add members to a contact group, run the following command:

```
davadmin ctgroup modify -a account -n name -c contactgroup -M members
```

Where:

account is the user account.

name is the name of the address book.

contactgroup is the vCard file that identifies the contact group to be modified, which you retrieved in the previous step.

members is a list of comma-separated list of members to add to the contact group.

For example, to add two members to the contact group **1384904616388-1758.vcf** in the address book **socialab** for the account **john.smith@example.com**:

```
davadmin ctgroup modify -a john.smith@example.com -n socialab -c
1384904616388-1758-GROUP.vcf, -M 1413320201700-4-.vcf,1413320035573-3-.vcf
```

Importing Contact Groups

Contacts Server enables you to import contact groups from CSV, LDIF, and vCard 3.0 formats. Use the **davadmin ctgroup import** command to import contact groups to an address book.

Note: The recommended format is vCard 3.0. Only use CSV if vCard 3.0 is unavailable.

To import a contact group:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

```
davadmin ctgroup import -a account -n name -m path
```

Where:

account is the user account.

name is the name of the address book.

path specifies the file and its path on the host that contains data to be imported.

For example, to import a contact group to the account **john.smith@example.com** and the address book **socialab** using the file **/temp/ctgroup/red_team_group.vcf**:

```
davadmin ctgroup import -a john.smith@example.com -n socialab -m
/temp/ctgroup/red_team_group.vcf
```

The sample data for the **red_team_group.vcf** file is:

```
BEGIN:VCARD
VERSION:3.0
FN:RedTeam
KIND:group
UID:urn:uuid:cd97370b-63a7-4fbf-9d82-fb2af1ad602c
MEMBER:1413320201700-4-.vcf
MEMBER:1413320035573-3-.vcf
END:VCARD
```

Exporting Contact Groups

Contacts Server enables you to export contact groups in vCard 3.0 format. Use the **davadmin ctgroup export** command to export contact groups. The **davadmin ctgroup export** command exports only the group vCard and not individual members.

To export a contact group:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the **davadmin ctgroup list** command to list the contact groups, and note the vCard file that identifies the contact group to export. See "Listing Contact Properties" for more information about listing contact information. The output will show the full WebDAV URL, for example:

   /dav/home/john.smith@example.com/addressbook/1384904616388-1758.vcf

   The part that you use in the **export** command appears after the "addressbook/" string, in this example, **1384904616388-1758.vcf**.
4. Run the following **davadmin** command:

   ```
   davadmin ctgroup export -a account -c contactgroup -x path
   ```

   Where:

   account is the user account.

   contactgroup is the vCard file that identifies the contact group to be exported, which you retrieved in the previous step.

   path is the file and path that the contact group would be exported to.

For example, to export the contact group**1384904616388-1758-GROUP.vcf** for the account **john.smith@example.com** to the file **/tmp/export.vcf**:

```
davadmin ctgroup export -a john.smith@example.com -c 1384904616388-1758-GROUP.vcf
-x /tmp/export.vcf
```

4

Managing Contacts Server

This chapter provides information and guidelines to help you manage the day-to-day operation of Oracle Communications Contacts Server.

Supported Application Server

Oracle Communications Contacts Server 8.0.0.4.0 and previous releases were deployed on GlassFish Server, which is no longer supported by Oracle. For that reason, Contacts Server 8.0.0.5.0 and beyond are only supported on Oracle WebLogic Server. Oracle strongly recommends that you upgrade your Contacts Server environments to release 8.0.0.5.0 or higher and migrate to WebLogic Server to receive full Oracle support.

Monitoring Contacts Server by Using Application Server

Contacts Server depends on Oracle GlassFish Server or Oracle WebLogic Server deployed as a web container. You monitor Contacts Server by using tools and commands of an application server.

For more information on administering GlassFish Server, see the Oracle GlassFish Server 3.0 documentation.

- Administering system security in *GlassFish Server Security Guide*.
- The **asadmin** utility subcommands in *GlassFish Server Reference Manual*.
- Overview of GlassFish Server administration in *GlassFish Server Administration Guide*.

For more information on administering Oracle WebLogic Server, see the Oracle WebLogic Server documentation.

- **Configuring Keystores** in *Administering Security for Oracle WebLogic Server 12.1.3*.
- **Configure keystores** in *Oracle Fusion Middleware Administration Console Online Help for Oracle WebLogic Server 12.2.1.3.0*.
- **Administration Console Online Help** in *Oracle Fusion Middleware Administration Console Online Help for Oracle WebLogic Server 12.2.1.3.0*.

Creating, exporting, and importing SSL Certificates in *Calendar Server Security Guide*. (The same concepts that apply to Calendar Server also apply to Contacts Server.)

Monitoring Contacts Server includes:

- Monitoring Application Server JDBC Connection Pools
- Checking Contacts Server Status

Monitoring Application Server JDBC Connection Pools

You can monitor GlassFish Server and WebLogic Server JDBC connection pools.

Monitoring GlassFish Server JDBC Connection Pools

If you use GlassFish Server JDBC connection pools, the following GlassFish Server statistics for JDBC Connection Pools are helpful in monitoring Contacts Server:

- **numConnFailedValidation** (count): Number of connections that failed validation.
- **numConnUsed** (range): Number of connections that have been used.
- **numConnFree** (count): Number of free connections in the pool.
- **numConnTimedOut** (bounded range): Number of connections in the pool that have timed out.

To get the statistics:

1. Check whether the JDBC connection pool service module is enabled:

```
asadmin get "server.monitoring-service.module-monitoring-levels.*"
...
server.monitoring-service.module-monitoring-levels.jdbc-connection-pool=OFF
```

2. If it is not enabled, start the JDBC connection pool service module, setting the monitoring level to **HIGH** to retrieve all statistics:

```
asadmin set
server.monitoring-service.module-monitoring-levels.jdbc-connection-pool=HIGH
server.monitoring-service.module-monitoring-levels.jdbc-connection-pool=HIGH
Command set executed successfully.
```

3. Get statistics for all connection pools:

```
asadmin get --monitor "server.resources.*"
```

Monitoring WebLogic Server JDBC Connection Pools

You can monitor a variety of statistics for each data source instance in your domain, such as the current number of database connections in the connection pool, the current number of connections in use, and the longest wait time for a database connection.

To view the current statistics for a JDBC data source:

1. In the **Domain Structure** tree, expand **Services**, then select **Data Sources**.
2. On the **Summary of JDBC Data Sources** page, click the data source name.
3. Select the **Monitoring** tab and then select the **Statistics** tab.

   Statistics are displayed for each deployed instance of the data source.

   For more information about these statistics, see Configuration Options.
4. (Optional) Click **Customize this table** to change the columns displayed in the statistics table. To make changes, you must select the **Lock & Edit** option. After the modifications, click **Activate Changes**.

   For more information on Monitoring WebLogic JDBC Resources, refer to Administering JDBC Data Sources for Oracle WebLogic Server.

Checking Contacts Server Status

You can use either the application server's Administration Console or the command-line utilities to check the Contacts Server status.

Checking Contacts Server Status with the Administration Console for GlassFish Server

1. Log in to the GlassFish Server host as **root**.
2. Start the GlassFish Server Administration console.
3. Navigate to **Web Applications** under the **Applications** tab.
4. Ensure that the process **nabserver** is deployed and enabled.

Checking Contacts Server Status with the asadmin Command for GlassFish Server

1. Log in to the GlassFish Server host as **root**.
2. Change to the *GlassFish_home*/**bin** directory.
3. Obtain the name of the **nabserver** component:

```
asadmin list-components -p admin-port
nabserver <ejb,web>
Command list-components executed successfully.
```

4. Show the status of the **nabserver** component:

```
asadmin show-component-status -p admin-port nabserver
Status of nabserver is enabled.
Command show-component-status executed successfully.
```

Checking Contacts Server Status with the Administration Console for WebLogic Server

To check Contacts Server status with the WebLogic Administration Console:

1. Log in to WebLogic Administration Console.
2. Navigate to **Deployments** under the domain.
3. Ensure that the **nabserver** process is deployed and enabled.
4. Under **Deployments**, ensure that **Health** of the nabserver deployment is **OK**.

Managing Logging

Managing logging includes:

- Logging Contacts Server Information to the Application Server Log File
- Configuring Logging
- Viewing Document Store Log Files

Logging Overview

Contacts Server maintains the following log files:

- **commands**: Stores information about requests that are sent to the server and information related to each operation performed that satisfies those requests. The **commands** log file contains servlet and core operation class entries that are designed to help you monitor requests to the server and help diagnose problems.

- **errors**: Stores error and debug-level information that is supplied by the server for use in diagnosing problems.
- **telemetry**: Stores entire Contacts Server servlet request and response transcripts.
- **scan**: Stores information on virus scanning actions.

Each log file has its own configuration parameters that controls the log file location, maximum size, log level, and number of files allowed.

Log files are created with a suffix of *.number*, for example, **commands.0**, **commands.1**, and so on. The log file numbered **.0** is the newest, the log file numbered **.1** is next newest, and so on. When a log file is filled to its maximum configured size, the logging system increments each of the existing log file suffixes to the next higher number, starting with the highest. If the number of log files reaches the configured maximum, the highest numbered log file is deleted and the next higher takes its place.

For example, Contacts Server is started for the first time and you have configured the maximum number of log files at 10. The logging system begins writing messages to the log file with the **.0** suffix. When the **.0** log file is filled to capacity, the logging system increments its suffix to the next higher number and the file becomes **.1**. The logging system then creates a new **.0** log file and begins writing messages to it. When the **.0** file become full, the logging system increments the **.1** file to **.2**, increments the **.0** file to **.1**, and creates a a new **.0** file. This process continues until the maximum number of configured log files is reached. When that happens, the logging system deletes the highest numbered (oldest) log file, **.9**, increments each of the lower numbered files' suffixes, and creates a new **.0** log file.

The Contacts Server log files are kept separate from the application server log files.

The GlassFish Server log files are stored in the *GlassFish_home***/domains/***domain_name***/logs** directory, for example, **/opt/glassfish3/glassfish//domains/domain1/logs**.

The WebLogic Server log files are stored in the *Weblogic_Domain*/**servers**/*managed_server_name*/**logs** directory.

Even though the container's log file is the **root** log file, by default, information that is stored in the Contacts Server's log files is not logged to the container's log file.

Logging Contacts Server Information to the Application Server Log File

By default, the Contacts Server **logToParent** flag is set to **false**. It prevents logging of information to the application server log file.

To log the Contacts Server information to the application server log file (**server.log** for GlassFish Server and *managed_server_name*.**log** for WebLogic Server) and the Contacts Server log file (**commands.0**), set the **log.dav.commands.logtoparent** parameter to **true**:

```
davadmin config -u admin -o log.dav.commands.logtoparent -v true
```

Configuring Logging

Use the **davadmin** command to configure Contacts Server logging parameters as shown in Table 4–1.

name can be **commands**, **errors**, **scheduling**, **telemetry**, or **scan**, depending on the type of logging you want to configure; use **error** to configure Contacts Server error logging. **SEVERE** and **WARNING** messages need immediate attention. **FINE**, **FINER**, and **FINEST** messages are usually informational only, but can provide more context for troubleshooting when accompanying **SEVERE** and **WARNING** messages.

Table 4–1 Contacts Server Log File Configuration Parameters

Parameter	Description
log.dav.*name*.**logdir**	Specifies the log file directory path
log.dav.*name*.**loglevel**	Specifies the log level: ■ **OFF**: No information is logged. ■ **SEVERE**: Logs catastrophic errors. ■ **WARNING**: Logs major errors or exceptions with the system. ■ **INFO**: Logs general informational messages. This is the default level. ■ **FINE**: Logs general debugging and tracing information to show the higher level flow through the code or more detailed information about a problem. ■ **FINER**: Logs more details than **FINE**. ■ **FINEST** or **ALL**: Logs the finest grain details about code flow or problem information. Enabling this level can result in massive amounts of data in the log file, making it hard to parse.
log.dav.*name***.logtoparent**	Enables or disables logging of the application server log file. When set to **true**, messages are stored in the application server log file and the Contacts Server log file. Set this parameter to **false** to disable logging to the application server log file.
log.dav.*name*.**maxlogfiles**	Specifies the maximum number of log files
log.dav.*name*.**maxlogfilesize**	Specifies the log file's maximum size

For more information about the logging configuration parameters and their default values, see "Contacts Server Configuration Parameters."

Viewing Document Store Log Files

The document store logs are named **astore.***number*, and are located in the *ContactsServer_home*/**logs** directory. Change to this directory to view the log files.

Modifying the Contacts Server Configuration

Modifying the Contacts Server configuration involves changing items such as LDAP operations, how Access Control List (ACL) entries are cached, automatic account creation, whether notifications are generated, and so on. You use the **davadmin config modify** command with configuration parameters to implement configuration changes. See "Contacts Server Configuration Parameters" for more information on individual parameters.

To modify the Contacts Server configuration:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin config modify -o configuration_parameter -v value
   ```

   Where:

   configuration_parameter is a specific Contacts Server configuration parameter.

value is an allowed value for the specific configuration parameter.

For example, to modify error logging to use the **FINE** level:

```
davadmin config modify -o log.dav.errors.loglevel -v FINE
```

Viewing the Contacts Server Configurations

Use the **davadmin config list** command to view Contacts Server configurations.

To view a configuration:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin config list -o configuration_parameter
   ```

   Where:

   configuration_parameter is a Contacts Server configuration parameter.

For example, to view the current setting for the **log.dav.errors.loglevel** parameter:

```
davadmin config list -o log.dav.errors.loglevel
```

Managing Contacts Server Back-End Databases

A standard Contacts Server installation consists of a single default back-end database that contains contact data. Perform the procedures in this section to add or modify back-end databases within your deployment.

Each back-end database must have its own document store. When you use multiple front-end hosts, all document stores must be available to all front-end hosts. You cannot make a document store local to a front-end host in a multiple front-end deployment. For more information, see the topic on configuring Contacts Server with multiple hosts in *Contacts Server Installation and Configuration Guide*.

Managing Contacts Server back-end databases includes:

- Adding an Additional Contacts Server Back-End Database
- Renaming the Default Contacts Server Back End Database
- Listing the Back-End Databases for a Contacts Server Deployment
- Purging a Contacts Server Back-End Database
- Clearing the Contacts Server Cache

Adding an Additional Contacts Server Back-End Database

A standard Contacts Server installation consists of a default back-end database that contains user data. Over time, you might want to add additional back-end user databases to your deployment.

In the case of multiple Contacts Server front ends, configure each to use the same initial default database back end. Then, you add additional back ends to each front end.

To add a new Contacts Server back-end database:

1. Install the database software on each back-end host.

See either the topic on installing a MySQL database or the topic on installing and creating an Oracle Database instance in *Contacts Server Installation and Configuration Guide*.

2. If you installed a MySQL database, do the following:
   a. If the Contacts Server software is not installed on the back-end host, copy the **config-mysql** and **Util.pm** scripts from an installed Contacts Server host and adjust the path to those scripts as shown in Steps b and c accordingly.
   b. Do one of the following:
      – If this is first database on the host, set up the database instance, and create the user and database by running the following command.

        ```
        ContactsServer_home/tools/unsupported/bin/config-mysql -s -u -c
        ```

      – If there is already a database on the host, just create the contact database by running the following command.

        ```
        ContactsServer_home/tools/unsupported/bin/config-mysql -c
        ```

3. If you installed Oracle Database 11g Release 2 or Oracle Database 12c not pluggable (non-CDB), do the following:
   a. If the Contacts Server software is not installed on the back-end host, copy the **config-oracle** and **Util.pm** scripts from an installed Contacts Server host and adjust the path to those scripts as shown in Steps b and c accordingly.
   b. To create the Oracle database user and schema, run the following command.

      ```
      ContactsServer_home/tools/unsupported/bin/config-oracle -c
      ```

4. If you installed Oracle Database 12c Container Database (that is, one that uses a pluggable database), see the topic on preparing Oracle Database 12c container database in the *Contacts Server Installation and Configuration Guide*. You cannot use the **config-oracle** script and must manually create the database user and schema.
5. Run the **config-backend** script on each front-end host.

   Note: If you use WebLogic Server, you cannot run the **config-backend** script. See the discussion about installing and configuring multiple Contacts Server back-end hosts for WebLogic Server manually in *Contacts Server Installation and Configuration Guide*.

   This script creates a JDBC connection pool and a JDBC resource on the GlassFish Server, and a **nabserver** attributed back-end configuration.

   ```
   ContactsServer_home/sbin/config-backend
   ```

   - If current deployment is using MySQL, you are prompted for the following information:

     ```
     Remote database server host name
     Remote database server port
     Contact db name on remote server
     Contact db user name
     Contact db user password
     Verifying the database input...
     Database input is verified
     Backend identifier for the remote db
     ```

```
Document store directory (leave blank if store is remote)
Document store host (leave blank if store is local)
Document store port (leave blank if store local)
Application Server admin user password
```

Make sure the value for "Contact db name on remote server" is the one that you used for the **config-mysql -c** command.

- If current deployment is using Oracle Database, you are prompted for the following information:

```
Remote database server host name
Remote database server port
Oracle database service name on remote server
Contact db user name
Contact db user password
Verifying the database input...
Database input is verified
Backend identifier for the remote db
Document store directory (leave blank if store is remote)
Document store host (leave blank if store is local)
Document store port (leave blank if store local)
Application Server admin user password
```

6. Enter **`Y`** when prompted to perform the tasks for creating the JDBC connection pool and resource, and **nabserver** back-end identifier.

   The system responds that the database back-end configuration is configured successfully.

7. Restart the application server.

Renaming the Default Contacts Server Back End Database

The **init-config** script creates the JDBC connection pool and resource, and adds the information to the **davserver.properties** file, for the one back-end host specified during the front-end host configuration. The JDBC resource for the back-end database is **defaultbackend**.

To rename this JDBC resource, to match other naming conventions, do the following on each front-end application server:

1. Create a JDBC resource associated with the **nabPool** connection Pool.

   For example, you might use **db1** as the resource name.

2. Save this change, and restart the application server.

3. Add the following two lines to each front-end host's *ContactServer_home*/**config/davserver.properties** file.

   ```
   store.dav.db1.backendid=JDBC_resource
   store.dav.db1.jndiname=jdbc/JDBC_resource
   ```

   For example, if your resource name is **db1**, then you would add:

   ```
   store.dav.db1.backendid=db1
   store.dav.db1.jndiname=jdbc/db1
   ```

The new resource name can be used in **nabStore** attribute values.

Note: Once your Contacts Server deployment is up and running, do not change the user back-end ID as defined by the **nabStore** attribute.

Listing the Back-End Databases for a Contacts Server Deployment

Use the **davadmin backend list** command to view the back ends configured for your Contacts Server deployment.

To list the back-end databases:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin backend list -u id
   ```

   Where:

   id is the application server administrator user name.

Purging a Contacts Server Back-End Database

The **davadmin backend purge** command immediately purges contact data marked for expiration from Contacts Server back-end database(s).

To purge a back-end database:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin backend purge -n name
   ```

   Where:

   name is the name of the back-end database.

For example, to purge a Contacts Server back end named **defaultbackend**:

```
davadmin backend purge -n defaultbackend
```

Clearing the Contacts Server Cache

Items that Contacts Server caches include ACLs, domain maps, LDAP authentication information, and URIs. Clear this cache if you made changes to your Directory Server, and you want Contacts Server to reflect those changes. Or, clear the cache so that any changes to ACLs immediately take effect.

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin cache clear
   ```

Managing Contacts Server LDAP Pools

You can configure a group of Directory Server hosts for use with Contacts Server. This group is referred to as an LDAP pool. Use the **davadmin ldappool** command to **create**, **modify**, **delete,** and **list** Contacts Server LDAP pools.

Managing LDAP pools includes:

- Creating an LDAP Pool
- Deleting an LDAP Pool
- Listing LDAP Pools
- Modifying an LDAP Pool

Creating an LDAP Pool

Use the **davadmin ldappool create** command to create an LDAP pool of Directory Servers.

To create an LDAP pool:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin ldappool create -n poolname -y property
   ```

   Where:

   poolname is the name of the LDAP pool.

   property is a comma-separated list of all *property=value* options for the specified LDAP pool.

For example, to create an LDAP pool named **myldap** and the *property=value* options **ldaphost=host1.example.com, ldapport=389, binddn='cn=Directory Manager' ,bindpassword=mypassword**:

```
davadmin ldappool create -n myldap -y
"ldaphost=host1.example.com,ldapport=389,binddn='cn=Directory
Manager',bindpassword=mypassword"
```

Deleting an LDAP Pool

Use the **davadmin ldappool delete** command to remove an LDAP pool.

To delete an LDAP pool:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin ldappool delete -n poolname
   ```

   Where:

   poolname is the name of the LDAP pool.

For example, to delete an LDAP pool named **myldap**:

```
davadmin ldappool delete -n myldap
```

Listing LDAP Pools

Use the **davadmin ldappool list** command to view LDAP pools.

To list existing LDAP pools:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin ldappool list
   ```

Modifying an LDAP Pool

You might need to modify properties of an existing LDAP pool, for example, if a host name or port changes. Use the **davadmin ldappool modify** command to change the properties of an existing LDAP pool.

To modify an LDAP pool:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

   ```
   davadmin ldappool modify -f file
   ```

   Where:

   file is the local input file containing the modifications. You use one line for each account, for batch operation. Lines are in the form *pool_name*:*property_list*. The properties are the same ones available for the **-y** option. For **delete** operations, only *pool_name* is used. For more information, see the description of the **ldappool -y** *property* option in *Calendar Server System Administrator's Guide*.

For example, to modify an LDAP pool by using the file **/tmp/update_pool.input**:

```
davadmin ldappool modify -f /tmp/update_pool.input
```

Managing the Contacts Server Document Store passfile

The remote document store mechanism provides storage of contact data on remote hosts. If you add additional document stores to your deployment, you must configure password authentication of the connection between the remote document store server (which runs on the remote host where the store is located), and the document store client (which runs on every Contacts Server front end). The password must be known by both the document store client and the remote document store server. The password is stored in a password file (called a *wallet*) on both the local and remote hosts.

Managing the document store passfile includes:

- Creating a passfile
- Listing a passfile
- Modifying a passfile

Creating a passfile

Use the **davadmin passfile create** command to create a passfile.

To create a passfile:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

```
davadmin passfile create
```

4. Respond to the password prompts.

For more information, see the topic on remote document store authentication in *Contacts Server Installation and Configuration Guide*.

Listing a passfile

Use the **davadmin passfile list** command to list the passwords stored in a passfile.

To list a passfile:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

```
davadmin passfile list
```

Modifying a passfile

Use the **davadmin passfile modify** command to modify the passwords in a passfile.

To modify a passfile:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** command:

```
davadmin passfile modify
```

4. Respond to the password prompts.

Managing Virus Scanning

To enhance security within your deployment, you can use Contacts Server virus scanning of files, such as photos, attached to a contact. Contacts Server virus scanning can examine files in real time to test and optionally reject incoming infected data. You can also choose to scan and optionally delete infected existing data on demand.

Virus scanning is not performed by Contacts Server itself. Instead, you configure an Oracle Communications Messaging Server's Message Transfer Agent (MTA) to filter the contact data. You can configure Contacts Server to share an existing MTA that has already been configured for Messaging Server virus scanning. Or, you can configure a standalone MTA that functions only for Contacts Server virus scanning.

Contacts Server reports all virus scanning activities and detected viruses in its log file for both real-time and on-demand scanning.

Managing virus scanning includes:

- Configuring Contacts Server for Virus Scanning

Configuring Contacts Server for Virus Scanning

The high-level steps to prepare your deployment to perform virus scanning for Contacts Server include:

1. Installing and Configuring the MTA
2. Configuring the MTA for Spam and Virus Filtering
3. Configuring Contacts Server Parameters for Virus Scanning

Installing and Configuring the MTA

It is possible that your deployment alread includes Messaging Server and an MTA for performing email virus scanning. If so, you can reuse the existing MTA to also scan contact attachments for viruses. If you do not have an existing MTA, you can install and configure a standalone MTA.

The general steps to install an MTA include:

1. Installing the Messaging Server software
2. Running the Messaging Server **configure** script
3. Disabling the Message Store and Webmail Server

For details, see the topic on installing a Messaging Transfer Agent in *Unified Communications Suite Installation and Configuration Guide*.

When configuring Messaging Server, the "configure" step requires a valid LDAP host that is used to include configuration data such as the default mail domain and messaging administrator account. The LDAP host that you specify must be available during virus scanning operations. However, due to MTA caching of LDAP data, this server is not heavily utilized.

Configuring the MTA for Spam and Virus Filtering

The MTA itself does not check for viruses. You configure the MTA to communicate with the desired virus scanning software, also referred to as the AVS. For instructions, refer to the vendor-specific sections in the topic on integrating spam and virus filtering programs in *Messaging Server System Administrator's Guide*.

The filter should use a Sieve rule to "refuse" the message from Contacts Server if a virus is found by the virus scanning software. The Sieve rule should return **FilterVerdictPositive**. Contacts Server checks SMTP return values for this exact string, which is defined in the Messaging Server **option.dat** file. For more information, see the topic on MTA configuration for virus scanning in *Calendar Server System Administrator's Guide*.

Note: You configure the MTA to perform a Sieve refuse action if there is a virus, which returns an SMTP code **5xy** and the MTA-configured target string **FilterVerdictPositive**. Contacts Server responds to the target string, where other errors are considered failures in service.

After you configure the MTA to communicate with the desired virus scanning software, you create a new incoming SMTP port in Messaging Server's **dispatcher.cnf** file, strictly for Contacts Server virus scanning use. In this way, Contacts Server traffic is tracked. In addition, a separate SMTP port makes it easier to destroy all data being scanned. You associate this incoming SMTP port with a new MTA channel in the **imta.cnf** file. Finally, you configure the receiving channel to use the **sourcespamfilter***n* that is configured with the desired virus scan software, so that incoming contact data is tested.

After creating the SMTP channel, you configure the MTA to detect the chosen email address. (The Contacts Server host sends the attachment data as an email with a user recipient email address.) The email address is set up to use the MTA's host name and domain, so that the MTA does not need to perform a lookup for the domain. The user email address itself is not significant because incoming data is not actually delivered.

For instructions on configuring the Messaging Server MTA, refer to *Messaging Server System Administrator's Guide*.

Configuring Contacts Server Parameters for Virus Scanning

Use the **davadmin config modify** command to configure Contacts Server parameters for virus scanning. Some parameters are required; others are optional.

To configure Contacts Server for virus scanning:

1. Use the **davadmin config modify** command to configure each of the following required parameters:
   - **davcore.virusscan.emailaddress**
   - **davcore.virusscan.host**
   - **davcore.virusscan.port**
   - **davcore.viruscan.onlineenable**
   - **davcore.virusscan.onlinevirusaction**

   For example:

   ```
   davadmin config modify -o "davcore.virusscan.emailaddress" -v
   "myvirususer@mymachine.example.com"
   ```

   The email address' domain must match the MTA's domain. The user name itself is not significant.
2. Configure optional **davcore.virusscan.*** parameters.

   See "Contacts Server Configuration Parameters" for more information on the optional **davcore.virusscan.*** parameters.

Virus Scanning Example Commands

To scan for viruses, use the **davadmin vscan scan** command. This command operates through the application server, and therefore, it can operate on any of the Contacts Server back-end hosts.

- To list the back ends:

  ```
  davadmin backend list
  ```

  Normally you would want to scan the **defaultbackend** because that is where Contacts Server user's attachments are stored.

- To scan the entire default back end:

  ```
  davadmin vscan scan -n defaultbackend
  ```

- To scan a single user's data given their Contacts Server registered email address:

  ```
  davadmin vscan scan -a user@domain
  ```

- To use an LDAP base (the distinguished name of the search base object) and filter to specify one or more users to scan:

  ```
  davadmin vscan scan -B "o=dav" -R "(|(uid=caluser222)(uid=caluser111))"
  Finished Virus Scan Set of 2 Users.
    User Login issues or data not found: 0
    Scanned = 0
    Virus hits = 0
    Scan Service Failures = 0
  See scan log for more information.
  ```

  Note: In this example, using only a uid filter might not be specific enough when there are multiple domains. You can use the **ldapsearch** command to test the specificity of the filter.

- To scan data at or after February 14th, 2015, 1 am Zulu:

  ```
  davadmin vscan scan -n defaultbackend -T 20150214T010000Z
  ```

  Specifying a **-T** scans data only at the specified time and later,. Additionally, **-T** saves time by ignoring older data already scanned. In the scan log, the time just before the scan began is printed after the run so it can be used with the **-T** option in the next scan if no new virus rules are relevant.

  Note: The **davadmin vscan** command uses the same virus scan configuration as when online virus scan is enabled (**davcore.virusscan.onlineenable** is set to **true**). However it does not use the **onlineenable** variable. Thus, you can run command-line scans without needing to affect incoming data.

About Logging for Virus Scanning

Virus scan activity for both online scanning and scanning from the command line is printed in the Contacts Server **scan** log. Found viruses are reported in the log. If actions against viruses are configured, those actions taken are reported in the log. Accounts that reference data that is found to be a virus are reported. The time just before a **davadmin vscan scan** command is started is printed after a scan. This can be used with the **-T** option in future scans.

Because the **davadmin vscan scan** command runs on the application server and not the **davadmin** client, most useful information is printed in the Contacts Server **scan** log, not always in the standard output of the **davadmin** command. The **scan** log also provides a central repository for all historical virus scan-related information and tracking.

Managing Logging for the MTA

For information about managing logging for the MTA, see *Messaging Server System Administrator's Guide*.

To view and test channel traffic, add the keyword **logging** to the **defaults** channel in the **imta.cnf** file. Add **LOG_CONNECTION=255** and **LOG_FILTER=1** to the Messaging Server **option.dat** file. Refer to the MTA documentation to interpret channel operations such as "E" enqueue and "D" dequeue, "O" open connection, "C" close connection. View messages coming in on the **tcp_vscan** channel, and dequeue onto the **bitbucket** channel.

About Proxy Authentication

Contacts Server uses a proxy user to bind to the Directory Server when making requests to search the directory. The LDAP entry for this user resembles the following:

uid=nab-admin-*hostfqdn***-***timestamp***,ou=People,***orgdn*

This user typically belongs to the **cn=Contacts End User Administrators Group** group. This special Contacts Server user makes Directory Server requests on behalf of the end user for whom the request is being carried out. The proxy process takes into account the Directory Server Access Control Instructions (ACIs) for that particular end user. The DN (Distinguished Name) of this newly created user is added to the server configuration as the **base.ldapinfo.ugldap.binddn**. For information on sample ACIs that show the attributes that Contacts Server needs for granting end users permission to search the LDAP directory, see *Contacts Server Security Guide*.

Managing the Corporate Directory

Contacts Server supports the use of a corporate directory, that is, a company-wide listing of user information made available to all Communications Suite users.

Managing the corporate directory involves:

- Configuring Contacts Server to Use the Corporate Directory
- Configuring a Domain-Specific Corporate Directory

Configuring Contacts Server to Use the Corporate Directory

To configure Contacts Server to use the corporate directory:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home***/sbin** directory.
3. Use the **davadmin config modify** command to set the following configuration parameters:

```
davadmin config modify -o davcore.ldapattr.corpdirectoryurl -v corpDirectoryUrl
davadmin config modify -o store.corpdir.useldapproxyauth -v true
davadmin config modify -o store.corpdir.enablecorpdir -v true
davadmin config modify -o store.corpdir.defaultcorpdirectoryurl -v
'ldap://ugldap/??sub?(objectclass=*)?vlv'
```

Configuring a Domain-Specific Corporate Directory

If you need access to multiple corporate directories, or need a corporate directory that is different from the server-wide default, use the multi-valued corpdirectoryurl LDAP attribute in the domain entry.

To configure Contacts Server to use a domain-specific corporate directory:

1. Log in to the Contacts Server host as **root**.

2. Change to the *ContactsServer_home*/**sbin** directory.
3. If you have not already done so, use the **davadmin ldappool create** command to create an LDAP pool, if the corporate directory resides on a different host than the one configured for the default user/group LDAP pool.

   See "Creating an LDAP Pool" for more information.
4. Add a **corpdirectoryurl** value to the LDAP attribute in the domain entry.

   For example, the following **ldapmodify** command adds the **corpdirectoryurl** value for an LDAP pool named **varriuspool**:

```
ldapmodify -h ugldap.example.net -D "cn=Directory Manager" -w ugadminpass
 dn: o=varrius.com,o=dav
 changetype: modify
 add: corpdirectoryurl
 corpdirectoryurl: ldap://varriuspool/??sub?(objectclass=*)?proxyauth=true
```

5. To have the change take effect immediately, clear the server domain cache:

```
cd ContactsServer_home/sbin
davadmin cache clear -t domainmap
```

Disabling the Corporate Directory for a Domain

To disable the corporate directory for a specific domain:

1. Add a **corpdirectoryurl** value to the LDAP attribute in the domain entry with the **enabled=false** URL extension.

   For example, the following **ldapmodify** command adds the **corpdirectoryurl** value for an LDAP pool named **varriuspool**:

```
ldapmodify -h ugldap.example.net -D "cn=Directory Manager" -w ugadminpass
 dn: o=varrius.com,o=dav
 changetype: modify
 add: corpdirectoryurl
 corpdirectoryurl: ldap://varriuspool/??sub?(objectclass=*)?enabled=false
```

   Note: If you have already defined the **corpdirectoryurl** attribute for a domain, then for each value that should be disabled, add the **enabled=false** to the LDAP URL extension.

2. To have the change take effect immediately, clear the server domain cache:

```
cd ContactsServer_home/sbin
davadmin cache clear -t domainmap
```

5

Monitoring Contacts Server

This chapter provides details on monitoring Oracle Communications Contacts Server.

About Monitoring Contacts Server

Contacts Server uses a managed bean (MBean) created in an application server to collect monitoring data. By using the application server's Java Management Extension (JMX) interface and a JMX-compliant client, you can access the monitoring data. The JMX client connects to the platform's MBeanServer by using a JMX Service URL. Once a client connects to the MBeanServer, it uses the Contacts Server monitoring MBean object name to access the MBean's attributes.

Contacts Server Monitoring Attributes

This section describes the attributes of the Contacts Server monitoring MBean object name, **com.sun.comms.davserver:type=monitor**.

General Monitoring Attributes

Table 5–1 describes the general monitoring attributes.

Table 5–1 General Monitoring Attributes

Name	Type	Description
ContactsCreated	Integer	The number of contacts and groups created since the server was started.
FailedLogins	Integer	The number of failed login attempts since the server was started.
BackendMonitorScheduleQData	CompositeData[]	The calendar schedule queue length per back-end database. For more information, see "Back-End Database Schedule Queue Attributes".
BackendMonitorARTData	CompositeData[]	The average response time per back-end database. For more information, see "Back-End Database Average Response Times Attributes".
BackendRTData	TabularType Map<*K*,*V*> TabularData (BackendRTData)	A dynamic collection of response time data of LDAP connections, provided in a Map interface, that is, Map<*String backendID*, *BackendRTData rtData*>. Both the UG lookup and LDAP authentication connections are monitored. For more information, see "LDAP Response Time Monitoring Attributes".

Back-End Database Schedule Queue Attributes

Table 5–2 describes the back-end database schedule queue monitoring attributes.

Table 5–2 Back-End Database Schedule Queue Monitoring Attributes

Name	Type	Description
backendID	String	Name ID of this back-end database as defined on this front-end host.
message	String	Optional exception or informational message from this back-end database.
activeCount	Long	The count of resources on the schedule queue that are scheduled for immediate processing. A value of **-1** means no data is available.
retryCount	Long	The count of resources on the schedule queue that initially failed and are waiting for a later retry. The default retry time period is 1 hour. The maximum retry default is 24.

Back-End Database Average Response Times Attributes

The average response time for a back-end database is passively calculated during normal work load and reported in milliseconds. The sample duration period is approximately 60 seconds. Numerical fields may have a value of **-1** if no data can be returned from that back-end database. However, if there is no activity, then the last good value is retained if possible. The data is measured by taking samples of real client requests. Thus, if no clients are active or are not making requests, there is no data to be measured.

Table 5–3 describes the back-end database average response time monitoring attributes.

Table 5–3 Back-End Database Average Response Times Monitoring Attributes

Name	Type	Description
backendID	String	Name ID of this back-end database as defined on this front-end host.
message	String	Optional exception or informational message from this back-end database.
ART	Long	The average response time for a random sampling of simple database requests in milliseconds over approximately a previous 60 seconds time frame. A value of **-1** means no data.
NSamples	Long	The number of samples taken in this average.
startTime	Long	The system time in milliseconds of the first sample.
endTime	Long	The system time in milliseconds of the last sample.
status	Long	The back-end database status as known by this front-end host. The possible values are: ■ **0** - Database is okay ■ **-1** - Database is down ■ **-2** - Database failed to start
statusTime	Long	The system time at which this JMX request was issued.

LDAP Response Time Monitoring Attributes

Table 5–4 describes the LDAP response time monitoring attributes.

Table 5–4 LDAP Response Times Monitoring Attributes

Name	Type	Description
backendID	String	Key. Name ID of this LDAP host, in format, *BackendType-HostName*, for example, **LDAPUg-01.example.com**.
RT	Long	The response time of simple back-end requests in milliseconds.
message	String	Optional information about the connection, for example, "Exception occurred during LDAP healthCheck()."
timestamp	Long	The system timestamp that was issued by this request.

Using a Java Management Extension Client to Access the Monitoring Data

Contacts Server itself does not provide a client to access the monitoring data. Instead, you can use any Java Management Extension (JMX) client.

To access the monitoring data, a JMX client needs the following information:

- Application server host name or IP address
- Application server port number (GlassFish Administration port or WebLogic Managed Server port)
- Application server administrative user name and password
- MBean ObjectName, which is **com.sun.comms.davserver:type=monitor**
- Attribute names

If you use GlassFish Server:

You connect a JMX client to the GlassFish Server's MBeanServer by using a JMX Service URL of the following form:

```
service:jmx:rmi:///jndi/rmi://host:port/jmxrmi
```

where:

- *host* is the name or IP address of GlassFish Server
- *port* is the GlassFish Server administration port number

If you use WebLogic Server:

Oracle recommends that you use the T3/T3S protocol support provided in the **wlthint3client.jar** library for a remote access.

See Accessing WebLogic Server MBeans with JMX section in Developing Custom Management Utilities Using JMX for Oracle WebLogic Server.

You can connect a JMX client to the WebLogic Server's MBeanServer by using a JMX Service URL of the following form:

```
service:jmx:t3s://host:port/jndi/weblogic.management.mbeanservers.runtime
```

where:

- *host* is the name or IP address of the WebLogic Server
- *port* is the WebLogic Server administration port number

Note: Ensure to provide SSL Port when using **t3s** protocol in the URL.

More information on JMX and JMX clients is available on the Java documentation web site at:

https://docs.oracle.com/javase/8/docs/technotes/guides/jmx/

Using the responsetime Script

In addition to using monitoring data gathered by the Contacts Server monitoring MBean, you can also check the health of your hosts by using the Contacts Server supplied **responsetime** script. This script sends a set of basic requests to Contacts Server and measures the amount of time needed to process those requests. When the **responsetime** script shows a spike or a large increase in response time, this indicates a potential issue with Contacts Server that needs to be addressed.

To run the **responsetime** script, you must provide the server type (**contacts**), the application server host name and port, and an LDAP user account to run the script. When the script finishes, it displays the number of milliseconds needed to run the series of requests to **stdout**. When the script encounters no problems, it returns an exit status of **0**. If the script encounters a problem, it returns an exit status of **1** to **stderr**. See "responsetime Script Error Codes" for a list of error codes and descriptions.

responsetime Script Syntax

Use the **responsetime** script to check the health of your Contacts Server hosts.

Location

ContactsServer_home/**sbin**

General Syntax

```
responsetime -t contacts -H host -p port [-s path_of_truststore]
             [-x context_root] [-L locale] [-h]
```

Table 5–5 describes the options.

Table 5–5 Options for responsetime Script

Option	Description
-t	Specifies to monitor Contacts Server (contacts).
-H	Specifies the application server host name.
-p	Specifies the application server administrative port.
-s	Specifies the path to the truststore file, if a secure connection is used.
-x	Specifies the context root for Contacts Server. The default is **/** (root).
-L	Specifies the language locale to use to display messages. The format is *LL_CC_VV*, where: ■ *LL* is the language code. ■ *CC* is the country code. ■ *VV* is the variant.

Table 5–5 (Cont.) Options for responsetime Script

Option	Description
-h	Displays usage help.

The **responsetime** script requires that you stream the following user name and password, each on a separate line, to the script by using **stdin**:

- **RT_USER=***user*
- **RT_PWD=***password*

For information on creating a dedicated user account for **RT_USER**, see "Creating a Dedicated User Account for the responsetime Script".

responsetime Script Error Codes

Table 5–6 describes the **responsetime** script error codes and descriptions.

Table 5–6 responsetime Script Error Codes

Error Code	String	Description
211	Invalid option:	An invalid option was entered on the command line.
212	The "{0}" option is required.	A required option was not entered on the command line. The "{0}" string is replaced in the message with the name of the missing option.
500	Ok	The request succeeded and the amount of time, in milliseconds, is displayed to **stdout**.
501	Application server is down.	The **responsetime** script cannot connect to the application server host.
502	Contacts Server is down or server path not found.	The **responsetime** program had trouble sending a request to the application server host.
504	Login failure.	A problem occurred when trying to log in to the application server host.
505	Invalid user name or password.	Either the user name or the password was invalid.
510	Unable to locate or open messages resource bundle.	A problem occurred when accessing the localization resource bundle.

responsetime Script Example

The following example shows how to invoke the **responsetime** script and run it by using **csrtuser** as **RT_USER**.

```
#!/bin/sh
#
echo "RT_USER=csrtuser\nRT_PWD=password" | sbin/responsetime -t contacts -H
sc11.example.com -p 8080 -x /nabserver

bash> example_csrt.sh
1374
bash>
```

Creating a Dedicated User Account for the responsetime Script

The **responsetime** script requires a user account in LDAP to be specified in the **RT_USER** variable. You should create a dedicated user account for the **responsetime** script to use. Create this user by using the Contacts Server **config-rtuser** script, which is located in the *ContactsServer_home*/**sbin** directory. The **config-rtuser** script both creates the user in LDAP and runs the **davadmin** command to create the user in the Contacts Server database.

To create a dedicated user for the **responsetime** script by using the **config-rtuser** script:

1. Log in to the Contacts Server host as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the **config-rtuser** script:

   ```
   config-rtuser
   ```

4. Respond to the prompts for user account and password, Directory Manager password, and application server administrative password.
5. When prompted to proceed, type **Y**.

The script runs the **ldapmodify** command to create the user account.

6

Improving Contacts Server Performance

This chapter describes how to tune your Oracle Communications Contacts Server deployment.

Tuning Contacts Server Logging

The Contacts Server logging function is I/O intensive. For optimal performance, decrease the log level to **WARNING**. Another option is to store the **log** directory on a fast storage device, such as a solid-state (SSD) system.

To change the log level on the **errors** and **commands** log files to **WARNING**:

1. Log in as **root**.
2. Change to the *ContactsServer_home*/**sbin** directory.
3. Run the following **davadmin** commands:

```
davadmin config modify -o log.dav.errors.loglevel -v WARNING
davadmin config modify -o log.dav.commands.loglevel -v WARNING
```

Tuning Oracle GlassFish Server

Because Contacts Server runs in an Oracle GlassFish Server container, it is important to correctly tune GlassFish Server. The GlassFish Server items that you tune include the Java Virtual Machine (JVM), JDBC pool, and HTTP settings.

This section provides GlassFish Server tuning recommendations for a medium-sized Contacts Server deployment. Use the following topics to adjust the values for the various configuration settings to suit your deployment:

- Tuning Java Virtual Machine Options
- Tuning JDBC Pool
- Tuning HTTP Service and Listener

Tuning Java Virtual Machine Options

The Java Virtual Machine (JVM) runs the byte codes in a compiled Java program. The JVM translates the Java byte codes into the native instructions of the host machine. GlassFish Server is Java process that requires a JVM to run and support the Java applications running on it. JVM settings are part of a GlassFish Server configuration.

You set the JVM options either by using the **asadmin** command or the GlassFish Server Administration Console.

Use the following JVM option values as a starting point for your Contacts Server deployment:

```
-XX:+UseParallelOldGC
-XX:ParallelGCThreads=6
-Xms3200m
-XX:MaxPermSize=192m
-server
-Dsun.rmi.dgc.server.gcInterval=1800000
-Dsun.rmi.dgc.client.gcInterval=1800000
-Xmx3200m
-XX:NewRatio=2
```

Tuning JDBC Pool

A JDBC connection pool is a group of reusable connections for a particular database. Because creating each new physical connection is time consuming, GlassFish Server maintains a pool of available connections.

A JDBC resource is created by specifying the connection pool with which the resource is associated. Multiple JDBC resources can specify a single connection pool. Some common connection pool properties are the database name (URL), the user name, and the password.

You administer the JDBC Pool settings by using the GlassFish Server **asadmin** command.

Use the following JDBC option values as a starting point for your Contacts Server deployment:

```
max-pool-size=200
cachePrepStmts=true
prepStmtCacheSize=512
```

Tuning HTTP Service and Listener

Tuning the monitoring settings for the HTTP server instances that handle client requests is important for ensuring peak GlassFish Server performance. You can enable or disable monitoring statistics collection for the HTTP service by using either the Administration Console or **asadmin** subcommands.

See the topic on administering the monitoring service in *Oracle GlassFish Server Administration Guide* for more information.

Table 6–1 shows the GlassFish Server HTTP service tuning settings to use as a starting point.

Table 6–1 HTTP Service Tuning

HTTP Setting	Attribute	Value
keep-alive	**max-connections**	250
NA	**timeout-in-seconds**	30
request-processing	**header-buffer-length-in-bytes**	16384
NA	**request-timeout-in-seconds**	20
NA	**thread-increment**	10
connection-pool	**max-pending-count**	4096
NA	**queue-size-in-bytes**	4096

Table 6–1 (Cont.) HTTP Service Tuning

HTTP Setting	Attribute	Value
NA	**receive-buffer-size-in-bytes**	4096
NA	**send-buffer-size-in-bytes**	8192

Table 6–2 shows the HTTP listener tuning settings to use as a starting point.

Table 6–2 HTTP Listener Tuning

HTTP Listener Setting	Value
acceptor-threads	1
accessLoggingEnabled	false
xpowered-by	false

Tuning Oracle WebLogic Server

The WebLogic Server configuration described in this section is for a medium-sized deployment. Adjust the values according to your deployment. You should perform modifications in the managed domain in which Contacts Server is deployed.

- Tuning JVM Options for WebLogic Server
- Tuning JDBC Pool for WebLogic Server
- Tuning HTTP Service and Listener for WebLogic Server

Tuning JVM Options for WebLogic Server

For details about setting the JVM options in Oracle WebLogic Server see the discussion about setting Java parameters for starting WebLogic Server and specifying Java options for a WebLogic Server instance in the following documents:

- Oracle Fusion Middleware Tuning Performance of Oracle WebLogic Server
- Oracle Fusion Middleware Administering Server Startup and Shutdown for Oracle WebLogic Server

JVM options:

```
-XX:+UseG1GC
-XX:ParallelGCThreads=6
-Xms3200m
-XX:MaxPermSize=192m
-server
-Dsun.rmi.dgc.server.gcInterval=1800000
-Dsun.rmi.dgc.client.gcInterval=1800000
-Xmx3200m
-XX:NewRatio=2
```

Tuning JDBC Pool for WebLogic Server

WebLogic Server instance uses a self-tuned thread-pool. The best way to determine the appropriate pool size is to monitor the current size of the pool, shrink counts, grow counts, and wait counts.

Configure the parameters related to JDBC Pool using WebLogic Administration Console:

1. Log in to WebLogic Server Administration Console.
2. Click **Lock & Edit**.
3. From the **Domain Structure** section, click the domain name. For example, domain1.
4. Navigate to **Services** and then **Data Sources**.

   JDBC Datasources - **defaultbackend** is displayed in the **Configuration** tab.
5. Select the JDBC Data Source name from the list, navigate to the **Connection Pool** tab, and then perform the following modifications:
   - Change the value of **Initial Capacity** to 200. The default value is 1.
   - Change the value of **Maximum Capacity** to 200. The default value is 15.
   - Change the value of **Statement Cache Size** to 512. The default value is 10.

   Note: Setting the size of the statement cache to 0 turns Off the statement caching. Therefore, setting this parameter to a non-zero value is equivalent to setting **cachePrepStmts=true** in GlassFish Server.
6. Click **Save**.
7. Click **Activate Changes**.
8. Restart WebLogic Server Administration Server and Managed server.

   Note: For more information, see the discussions about self-tuning thread pool, tune the number of database connections, tune pool sizes, and tuning data sources in the Oracle WebLogic Server documentation.

Tuning HTTP Service and Listener for WebLogic Server

WebLogic Server is enabled with self-tuning for most of the HTTP parameters. Ensure that the following parameters are set by default. If the parameters are not set, you can set them using the WebLogic Server Administration Console.

1. Log in to WebLogic Server Administration Console.
2. From the **Domain Structure** section, click the domain name.
3. Click **Environment**, **Servers**, *Managed Server Name*, and **Tuning** tab.

   Note: The **Enable Native IO** option is selected by default.

   You should set the **Accept Backlog** value to 300.
4. Select **Environment**, **Servers**, *Managed Server Name*, **Tuning Tab**, and **Advanced** section.
5. Set the Self-Tuning Thread Minimum Pool Size value to 1 and Self-Tuning Thread Maximum Pool Size value to 400.
6. Select **Environment**, **Servers**, **Protocols** tab, and then **HTTP** tab.

Note: The **Keep-Alive** option is enabled by default.

7. Select **Services**, **Messaging**, and **JMS Servers**.
8. Click JMS Server that Contacts Server has created. For example, JMSServer-DAV.
9. Navigate to the **Configuration** tab, **General** tab, **Advanced** section, and verify the following:
   - **Message Buffer Size**: -1, which indicates that the server automatically determines a size based on the maximum heap size of JVM. This default value is set to either one-third of the maximum heap size or 512 megabytes, whichever is smaller.

   For more information, refer to Fusion Middleware Tuning Performance of Oracle WebLogic Server.

Tuning MySQL Server

For MySQL Server, configure the parameters that affect cache size and maximum connection size. For example, use the following values as a starting point:

```
back_log = 50
max_connections = 200
binlog_cache_size = 1M
max_heap_table_size = 64M
sort_buffer_size = 8M
join_buffer_size = 8M
thread_cache_size = 8
thread_concurrency = 8
query_cache_size = 64M
query_cache_limit = 2M
ft_min_word_len = 4
memlock
thread_stack = 192K
transaction_isolation = REPEATABLE-READ
tmp_table_size = 64M
log-bin=mysql-bin
expire_logs_days=1
binlog_format=mixed
slow-query-log = 1
long_query_time = 2
log_long_format
tmpdir = /tmp
innodb_additional_mem_pool_size = 16M
innodb_buffer_pool_size = 2G
innodb_data_file_path = ibdata1:10M:autoextend
innodb_file_io_threads = 4
innodb_thread_concurrency = 16
innodb_flush_log_at_trx_commit = 1
innodb_log_buffer_size = 8M
innodb_log_file_size = 256M
innodb_log_files_in_group = 3
innodb_max_dirty_pages_pct = 90
innodb_lock_wait_timeout = 120
innodb_flush_method=O_DIRECT #UFS only
```

See *MySQL Server Administration* for more information on how to change these settings.

Caution: You can view contents of the back-end store by using standard MySQL tools. Do not use MySQL tools to modify your data.

Tuning Oracle Solaris CMT Server

This section provides tuning recommendations for Chip Multi-threading (CMT) architectures such as Sun servers with CoolThreads technology.

Set the following parameters and values in the **/etc/system** file.

```
set rlim_fd_max=260000
set hires_tick=1
set sq_max_size=0
set ip:ip_squeue_bind=0
set ip:ip_squeue_fanout=1
set ip:ip_soft_rings_cnt=16
```

Set the following parameters for TCP tuning:

```
ndd -set /dev/tcp tcp_time_wait_interval 60000
ndd -set /dev/tcp tcp_conn_req_max_q 3000
ndd -set /dev/tcp tcp_conn_req_max_q0 3000
ndd -set /dev/tcp tcp_max_buf 4194304
ndd -set /dev/tcp tcp_cwnd_max 2097152
ndd -set /dev/tcp tcp_xmit_hiwat 400000
ndd -set /dev/tcp tcp_recv_hiwat 400000
```

For Sun Fire T1000 and T2000 systems with 1.0GHz CPU, interrupt fencing by setting the following parameter:

```
psradm -i 1-3 5-7 9-11 13-15 17-19 21-23
```

For ZFS, set the **recordsize** variable to 16 K (same as innoDB block size) by running the following commands:

```
zfs create rpool/data
zfs set recordsize=16K rpool/data
```

Tuning Reference

Use the following documentation to find more information on tuning GlassFish Server, MySQL Server, and network performance for Contacts Server:

- Optimizing MySQL Server:

  For more information, refer to the *Optimizing the MySQL Server* section in MySQL documentation.

- MySQL benchmarks:

  http://www.mysql.com/why-mysql/benchmarks/

- Scaling MySQL, T5440, ZFS:

  https://downloads.mysql.com/presentations/MySQL_Perfornance__Tuning_Overview_jp.pdf

  https://learn.oracle.com/ols/course/mysql-57-performance-tuning/51871/61185

- Tuning network performance:

Refer to the *TCP/IP Tunable Parameters* section in *Solaris Tunable Parameters Reference Manual* for more information.

- Tuning GlassFish Server:

  http://docs.oracle.com/cd/E18930_01/html/821-2431/

- WebLogic Server:

  https://docs.oracle.com/middleware/12213/wls/PERFM/toc.htm

7

Migrating Information to Contacts Server

This chapter describes how to migrate information from Oracle Communications Convergence Personal Address Book (PAB) to Oracle Communications Contacts Server.

Introduction to Migrating to Contacts Server

Contacts Server stores users' contact information in either a MySQL Server or Oracle database. Convergence stores Personal Address Book (PAB) data in LDAP in Directory Server. To use PAB contact data in Contacts Server, you must migrate it by using the **davadmin migration** command.

The **davadmin migration** command migrates Convergence PAB contacts and contact groups to the Contacts Server back-end database in vCard format. The migration transfers whatever information is found in the LDAP directory to the Contacts Server back-end database. Additionally, if you migrate the same PAB address book multiple times, the user does not end up with duplicate data in the migrated address book.

> **Note:** Original PAB LDAP entries remain the same after migration. Keep these entries in the short term while diagnosing any migration issues.

The data migration process assumes the following conditions:

- You can take a reasonable amount of downtime to complete the migration.
- You can perform a trial run to check results before doing the actual migration.
- You can examine and fix address books that failed to migrate, by examining the **master_log** and **user_log** files for errors, then rerunning the migration on those address books that failed.
- If your deployment consists of multiple domains, you have configured ACIs to allow the Contacts Server administrator ID search and read access to non-default domains.

> **Note:** You can also have a coexistent deployment of Contacts Server and PAB hosts. For more information, see the topic on address book co-existence in *Contacts Server Installation and Configuration Guide*.

About the Personal Address Book

The Convergence PAB stores users' contact information in the Directory Server LDAP directory under the distinguished name **o=PiServerDb**. The following example shows the directory structure in which the address book entries for user **jsmith** are located:

```
o=PiServerDb
   o=example.com
     o=piPStoreOwner=jsmith
```

The following sample LDIF file shows the LDAP entries that store address book data for the user **jsmith** under the **o=piPStoreOwner** entry. The example includes entries for the user's personal address book, corporate directory, and personal store:

```
dn: piPStoreOwner=jsmith,o=example.com,o=PiServerDb
piDefaultAB: e10976f864e00
lastPurgeDate: 20060217T074523Z
piPStoreOwner: jsmith
objectClass: piPStoreRoot
objectClass: top

dn: piEntryID=e10976f864e00,piPStoreOwner=jsmith,o=example.com, o=PiServerDb
displayName: Personal Address Book
objectClass: PITYPEBOOK
objectClass: piLocalBook
objectClass: top
piEntryID: e10976f864e00
multiLineDescription: This is your Business Address Book
piBookType: abook

dn: piEntryID=e10976f865771,piPStoreOwner=jsmith,o=example.com, o=PiServerDb
displayName: Corporate Directory
objectClass: PITYPEBOOK
objectClass: piRemoteBook
objectClass: top
piEntryID: e10976f865771
multiLineDescription: This is your Corporate Directory
piRemotePiURL: ldap://corpdirectory
piBookType: abook

dn: piEntryID=e10976f8659f2,piPStoreOwner=jsmith,o=example.com, o=PiServerDb
displayName: Applications
objectClass: PITYPEBOOK
objectClass: top
piEntryID: e10976f8659f2
piBookType: pbook

dn: piEntryID=e10976f865bd3,piPStoreOwner=jsmith,o=example.com, o=PiServerDb
displayName: Personal Store
objectClass: PITYPEPROFILE
objectClass: piEntry
objectClass: top
piEntryID: e10976f865bd3
memberOfPIBook: e10976f8659f2

dn: piEntryID=e10976f8665f4,piPStoreOwner=jsmith,o=example.com, o=PiServerDb
displayName: Applications
objectClass: PITYPEPROFILE
objectClass: piEntry
objectClass: top
piEntryID: e10976f8665f4
```

```
memberOfPIBook: e10976f8659f2
```

Each PAB address book becomes a corresponding address book in the Contacts Server back-end database for the user, except for the Corporate Directory and Certificate Book. The migration does not migrate those entities. The migration uses the PAB address book's **piEntryID** as the URI of the address book, except for the default address book. Because a Contacts Server account is created with a default URI of **addressbook**, the migrated default PAB address book uses its **piEntryID** as the URI.

About the Migration Process

The migration process involves the following high-level steps:

1. The **davadmin migration** command locates the user's PAB address book and retrieves its LDAP entry.
2. To determine the search base for the user and address books under the **o=PiServerDb** tree, the migration process uses either the **psRoot** attribute, or, if it is empty, constructs a lookup of the user's uid and email domain.
3. The migration process performs an LDAP search on the base **piPStoreOwner=***uid*, **o=***domain*, **o=PiServerDb** using the filter **(&(objectclass=PITYPEBOOK)(piBookType=abook))** to return the list of address books for the user.
4. The migration process makes each address book into a corresponding collection in the Contacts Server back-end database for the user, except for the Corporate Directory and Certificate Book, which are not migrated.
5. To get the list of contacts for each address book, the migration process performs an LDAP search on the base **piPStoreOwner=***uid*, **o=***domain*, **o=PiServerDb** using the filter **(&(objectclass=PITYPEPERSON)(objectclass=piEntry)(memberOfPIBook="***book entryid***"))** where *bookentryid* is the **piEntryID** of the address book.
6. The migration process transforms the LDAP entry of the contact into vCard format for storing in the Contacts Server back-end database. This transformation uses an LDIF-to-vCard import mechanism translation file. This translation file defines the mapping of various PAB LDAP attributes to their vCard counterparts.
7. Each transformed contact becomes a node under the address book collection using the URI formed by the **piEntryID**, which is appended with a **.vcf** file extension.

davadmin migration Command

To migrate PAB information to Contacts Server, run the **davadmin migration migrate** command. For more information, see "Contacts Server Command-Line Utilities".

> **Note:** You must use the **davadmin migration** command to migrate contacts from the Convergence PAB to Contacts Server. Do not export and import contacts as CSV files as a way of migrating the contacts.

For example, to migrate the address books and contacts for a single user **john.smith@example.com**:

```
davadmin migration migrate -a john.smith@example.com -X "cn=Directory Manager" -L
pab-ds.example.com:636 -l /tmp -S
Enter Admin password:
```

```
Enter Migration Admin password:
log tag = /tmp/nabserver_migration/2012-11-05_113142/master_log
```

Migration Logging and Status

Each migration creates a migration log file (**master_log**), which logs the list of users migrated, along with a success or failure status. This log file name is returned as the **log tag** value when the **davadmin migration** command completes. You can use this log tag to display the migration status by running the **davadmin migration status** command.

Each migration also creates a separate migration log per user (**user_log**) located in a subdirectory named with the user's email address. This log shows each address book processed and the contact entries of the migrated address book.

For example, to check on the status of the migration:

```
davadmin migration status -G /tmp/nabserver_migration/2012-11-05_113142/master_log
Enter Admin password:
[john.smith@example.com] Migration begin
[john.smith@example.com] Migration completed
```

Troubleshooting the Migration

Troubleshooting the migration involves looking at the following errors:

- Back-End Database Error
- LDAP Error
- Read Timed Out Error

Back-End Database Error

If your migration produces errors similar to the following:

```
[user@host.example.com] Validation exception: backend throws an error
(com.sun.comms.nabserver.backends.BackendException: SQL
error: Error in allocating a connection. Cause: In-use
connections equal max-pool-size and expired max-wait-time.
Cannot allocate more connections.(INTERNAL_STORE_ERROR)) while doing nodeExists on
dav/home/user@host.example.com/addressbook/dropbox/00000000000
0000000000000000000405ce44e000063da0000003100004aa0/ while storing:
0000000000000000000000000000000405ce44e000063da0000003100004aa0
```

then you might need to do the following:

1. Reduce the value of **davcore.serverlimits.maxmigrationthreads**.

   The default value is 2. Try changing the value to 2*(number of CPUs).

2. Change the MySQL connection pool size.

   The default is 32. Change this value to 4*(number of threads). To change this value, edit the **/etc/my.cnf** file and increase the MySQL **max_connection** setting, for example, **max_connections = 200**.

LDAP Error

If your migration using an LDAP filter produces an error like:

```
LDAP search failure: error result
```

then your filter might be too broad and you might be running into an LDAP search limit exceeded issue. Try narrowing down your filter. For example, if your filter was:

```
davadmin migration -X calmaster -L cs6.example.com:8080 -B "o=dirbase" -R
"objectclass=icscalendaruser"
```

change it so that it resembles the following:

```
davadmin migration -X calmaster -L cs6.example.com:8080 -B "o=dirbase" -R
"(&(uid="a*")(objectclass=icscalendaruser))"
```

If you use this approach, you need to run multiple migration commands to complete the migration for the entire directory.

Read Timed Out Error

If your migration produces errors similar to the following:

```
2010-04-19_151418/master_log:[user@host.example.com] Exception on creation of
http://host-cs.example.com:80: Read timed out
```

then you might need to increase the timeout period for HTTP connections. To do so, change the **davcore.serverlimits.httpconnecttimeout** and **davcore.serverlimits.httpsockettimeout** parameters by using the **davadmin config** command.

8

Managing the Contacts Server Database

This chapter provides information about managing your Oracle Communications Contacts Server database.

Administering the MySQL Server Database

The following documentation provides information about administering MySQL.

- Starting and Stopping MySQL Automatically in *MySQL 5.5 Reference Manual*
- MySQL Server and Server-Startup Programs in *MySQL 5.5 Reference Manual*
- MySQL Server SQL Administrative and Utility Programs in *MySQL 5.5 Reference Manual*
- Backing Up and Restoring Files and Data

Caution: You can view contents of the back-end database by using standard MySQL tools. Do not use MySQL tools to modify your data.

Administering the Oracle Database

The following documentation provides information about administering Oracle Database.

- Stopping and Starting Oracle Software in *Oracle Database Administrator's Reference for Linux and UNIX-Based Operating Systems*
- Administering Oracle Database in *Oracle Database Administrator's Reference for Linux and UNIX-Based Operating Systems*
- Backing Up and Restoring Files and Data

Caution: You can view contents of the back-end database by using standard Oracle Database tools. Do not use Oracle Database tools to modify your data.

9

Backing Up and Restoring Files and Data

This chapter describes the backing up and restoring files and data in Oracle Communications Contacts Server.

About Contacts Server Backup

Contacts store backup and restore is one of the most important administrative tasks for your Contacts Server deployment. You must implement a backup and restore policy for your contacts store to ensure that data is not lost if problems such as system crashes, hardware failures, or accidental deletion of information occur.

This chapter describes the options for backing up and restoring the Contacts Server database (either MySQL database or Oracle Database), and the document store. You need to understand the pros and cons of these options to make the proper choice for your deployment.

This information also assumes that you are backing up your LDAP Directory Server. Contacts Server uses Directory Server to authenticate users, groups, and resources. Contacts Server uses the **davUniqueId** LDAP attribute to map each contacts entry (in LDAP) to a unique account in the contacts store. The unique identifier links various entries from different database tables for a user, group, and resource. You must use a unique identifier, and one that does not change, for user, group, and resource entries stored in LDAP.

Contacts Server Backup and Restore Techniques

The section describes the ways to back up the Contacts Server data store.

Note: You cannot back up Contacts Server by backing up the active contacts database and the Contacts Server **data** directory while Contacts Server is running. If you do so, bad data results. Thus, you must use one of the methods described in this section.

Using the davadmin db Commands

Contacts Server provides the **davadmin db backup** and **davadmin db restore** commands to back up and restore the Contacts Server data.

Pros:

- Supports partial backup and restore.
- You can also use **backup** and **restore** to migrate data from one Contacts Server host to another.

Cons:

- The **davadmin db backup** command is relatively slow.
- The **davadmin db restore** command might take longer than the **backup** command, as it needs to rebuild the database and indexes.

Using ZFS Snapshots

You can use Oracle Solaris ZFS snapshots to produce an atomic snapshot of the file system containing the MySQL database or Oracle Database and the document store. Then use **zfs send** or third-party file system backup software to back up the snapshot. See *ZFS Administration Guide* for more information.

Pros:

- Performance is better than **davadmin db backup**.

Cons:

- This method does not support partial backup and restore.

MySQL Server Backup and Restore Techniques

The following methods back up the MySQL Server database only. For general information about MySQL Server backup and restore, see the MySQL Server database documentation at:

http://dev.mysql.com/doc/refman/5.5/en/backup-and-recovery.html

- MySQL Async Replication

  Use MySQL replication to replicate the databases. See the MySQL Async Replication documentation at:

  http://dev.mysql.com/doc/refman/5.5/en/replication.html

- MySQL database dump

  Use **mysqldump** to dump the databases for backup or transfer to another SQL server. See documentation about **mysqldump** at:

  http://dev.mysql.com/doc/refman/5.5/en/mysqldump.html

- Point-in-time backup and recovery using the binary log.

  The binary log files provide you with the information you need to replicate changes to the database. See the documentation about point-in-time backup and recovery documentation using the binary log at:

  http://dev.mysql.com/doc/refman/5.5/en/point-in-time-recovery.html

Oracle Database Backup and Restore Techniques

For general information about Oracle Database backup and restore, see the Oracle Database documentation at:

http://docs.oracle.com/cd/E11882_01/nav/portal_14.htm#backup_and_recovery

10

Troubleshooting Contacts Server

This chapter describes troubleshooting strategies for Oracle Communications Contacts Server.

Troubleshooting Contacts Server Initial Configuration

If you experience trouble configuring Contacts Server while running the **init-config** initial configurator script and you receive an error from the application server, ensure that you are running the recommended Java version based on the JDK support available for the container and that your environment is configured appropriately.

> **Note:** If you use GlassFish Server 3.x, use the JDK version 1.7 and if you use WebLogic Server 12.x, use the JDK version 1.8. For more information, see the installation guide of the corresponding application server.

Troubleshooting Application Server and Java

If you upgrade your Java SE to Java SE Development Kit 7, Update 7 (JDK 7u7) or later, you must also upgrade GlassFish Server to the recommended patch level. If you use WebLogic Server, upgrade Java to the recommended JDK8 update version as suggested by the WebLogic Server version. Otherwise, you may encounter problems running the **davadmin** command.

Troubleshooting Tips

Begin troubleshooting by ensuring that the application server web container is running and that Contacts Server is deployed. You can use either the application server's Administration Console or the command-line utilities.

Topics in this section:

- GlassFish Server
  - Using the asadmin Command to Specify GlassFish Server Port
  - Using GlassFish Server to Check Contacts Server Status
- WebLogic Server:
  - Using the WebLogic Server Administration Console to Check Contacts Server Status
- Generic:

- Troubleshooting Contacts Server nabserver Process
- Troubleshooting a Failing davadmin Command
- Troubleshooting Back-end Database Errors
- Refreshing Domain Information
- Tuning Directory Server

Using the asadmin Command to Specify GlassFish Server Port

If you have more than one GlassFish Server instance installed, use the **asadmin -p** command to specify the instance's administrative port number.

Using GlassFish Server to Check Contacts Server Status

You can use either the GlassFish Server Administration Console or the **asadmin** command to check Contacts Server status.

To use the Administration Console to check status:

1. Start the GlassFish Server Administration Console.
2. Navigate to Web Applications under the Applications tab.
3. Ensure that the **nabserver** process is deployed and enabled.

To use the **asadmin** command to check status:

1. Log in to the GlassFish Server host as **root**.
2. Change to the *GlassFish_home*/**bin** directory.
3. Run the following commands:

```
asadmin list-components -p admin-port
nabserver <ejb,web>
Command list-components executed successfully.
asadmin show-component-status -p admin-port nabserver
Status of nabserver is enabled.
Command show-component-status executed successfully.
```

If the **nabserver** is not enabled, check the log files specified in "Troubleshooting Contacts Server nabserver Process".

Using the WebLogic Server Administration Console to Check Contacts Server Status

To check the Contacts Server status by using the WebLogic Server Administration console:

1. Start the WebLogic Server Administration Console.
2. In the **Domain Structure** section, click the domain name. For example, domain1.
3. Navigate to **Environment, Servers**, and then to the **Configuration** tab.

> **Note:** Ensure that the Administration Server and Managed Server in which Contacts Server is deployed are up and running.

4. Navigate to **Deployments**.
5. Ensure that **nabserver** is deployed under the **Configuration** tab.

Troubleshooting Contacts Server nabserver Process

To troubleshoot when Contacts Server is not starting or not allowing clients to connect:

1. If the **nabserver** process is not enabled, check the application server log in which Contacts Server is deployed.
   - On GlassFish Server:

     Check server.log in the *GlassFish_home*/**domains/domain1/logs** directory.
   - On WebLogic Server:

     Check *managed_server_name*.log in *Weblogic_Domain*/**servers**/*managed_server_name*/**logs** directory.
2. If the **nabserver** process is deployed and enabled but Contacts Server clients have trouble connecting, check the Contacts Server log, **error.***, in the **/var/opt/sun/comms/nabserver/logs** directory. To increase the Contacts Server log level to collect more information to help to troubleshoot the problem, run the following command:

   ```
   davadmin config -o log.dav.errors.loglevel -v FINEST
   ```

Troubleshooting a Failing davadmin Command

For GlassFish Server:

If a **davadmin** command fails to run, use the **-e** option to get more details about the failure. For example:

```
davadmin version
Enter Admin password:*********
DAV server connection failed. Is the server running?

davadmin version -e
Enter Admin password:*********
JMXconnection exception for url
service:jmx:rmi:///jndi/rmi://commsuite.example.com:46633/jmxrmi
- Exception creating connection to: 1.1.1.1; nested exception is:
java.net.SocketException: java.security.NoSuchAlgorithmException: Error
constructing implementation (algorithm: Default, provider: SunJSSE, class:
com.sun.net.ssl.internal.ssl.DefaultSSLContextImpl)
```

This example shows SSL errors. In this case, ensure that the truststore file is valid. The default truststore file, **.asadmintruststore**, is located in the Contacts Server **config** directory.

To verify that the truststore file is valid:

1. Log in to the GlassFish Server host as **root** where Contacts Server is deployed.
2. Run an **asadmin** command.

   GlassFish Server creates a new **.asadmintrustore** file located under the root (**/**) directory.
3. Ensure that this file is the same as the one in the Contacts Server **config** directory.

See also "Troubleshooting Application Server and Java".

For WebLogic Server:

If you use WebLogic Server and when **davadmin** command is successful, the following output is displayed:

```
/opt/sun/comms/nabserver/sbin/davadmin version
Enter Admin password: *********
Handshake succeeded: TLSv1.2
Oracle Communications Contacts Server version: 8.0.0.5.0 (built yyyy-mm-dd-Time)
```

The following example shows the output when the **davdmin** command fails:

```
/opt/sun/comms/nabserver/sbin/davadmin version
Enter Admin password: *********
Handshake failed: TLSv1.2, error = sun.security.validator.ValidatorException: PKIX
path building failed: sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target
Handshake failed: TLSv1.1, error = sun.security.validator.ValidatorException: PKIX
path building failed: sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target
Handshake failed: TLSv1, error = Received fatal alert: handshake_failure
Handshake failed: TLSv1.2, error = sun.security.validator.ValidatorException: PKIX
path building failed: sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target
Handshake failed: TLSv1.1, error = sun.security.validator.ValidatorException: PKIX
path building failed: sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target
Handshake failed: TLSv1, error = Received fatal alert: handshake_failure
Server unavailable at url:
service:jmx:t3s://commsuite.example.com:46633/jndi/weblogic.management.mbeanserver
s.runtime
```

In the example, 46633 is the secure port of the managed server in which Contacts Server is deployed.

This example shows SSL errors. In this case, check the following to troubleshoot the issue:

- WebLogic Server is configured in Secure mode using the supported keystores
- WebLogic Administration Console is accessible as https://hostname:secure_port/console
- The **extractSSLArgs.sh** script runs successfully in a secure mode before doing initial configuration.

  sh ./extractSSLArgs.sh -u *weblogic_admin_user* **-p** *weblogic_admin_user_password* -l t3s://*weblogic_server_host*:*SSL_port*

- If there is a problem in running the above script successfully, try to use WLST command to connect to the server.

  wls:/offline> connect(**weblogic_admin_user**,**weblogic_admin_user_password**,**t3s://***weblogic_server_host*:*SSL_port*");

- *WebLogic_Domain*/**config** contains a valid **.wls_sslargs** file and the contents correspond to the same keystore options that is configured at the WebLogic Server Side Secure Configuration.
- davadmin.properties file under *ContactsServer_home*/**config** folder contains proper details.

  For example:

  port=*managed_server_port*

  secure=*location of truststore used in configuring WebLogic Server in secure mode*

For more information, see the discussion about running extractSSLArgs.sh to validate and store WebLogic Server SSL details in the *Contacts Server Installation and Configuration Guide*.

Troubleshooting Back-end Database Errors

If you find a back-end error, do one of the following to ensure that the database is running by pinging the JDBC **connectionpool**.

If you use GlassFish Server:

1. Start the GlassFish Server Administration Console.
2. Select **JDBC Resources** from Resources, then select **Connection Pools**.
3. Choose the **nabPool** and perform a ping.

   Note: If the ping fails, check the Pool properties to ensure they are correct.

4. You can also perform a command-line ping as follows:

```
asadmin list-jdbc-connection-pools -p admin-port
__CallFlowPool
__TimerPool
DerbyPool
nabPool
Command list-jdbc-connection-pools executed successfully.

asadmin ping-connection-pool -p admin-port nabPool
Command ping-connection-pool executed successfully.
```

5. Even if you ping the pool, sometimes Contacts Server is not able to load the back end. In this case, you see errors similar to the following:

```
SEVERE  [2009-09-03T22:00:53.310-0700] <...JdbcBackend.getDataSource> Cannot
lookup DataSource: javax.naming.NameNotFoundException: defaultbackend1 not
found

SEVERE  [2009-09-03T22:00:53.313-0700] <...nabserver.loadBackend> failed to
instantiate or create backend
com.sun.comms.nabserver.backends.BackendException: Cannot get DataSource:
javax.naming.NameNotFoundException: defaultbackend1 not found(OPERATION_NOT_
SUPPORTED)
```

6. To see the pool and resource data clearly, view the GlassFish Server configuration file:

   GlassFish_home/domains/domain1/config/domain.xml

7. If the cause of the error is not clear, delete and recreate the connection pool and JDBC resource by using the **asadmin** command. If you recreate the JBDC resource, be sure to use the same user name and password that you initially used to create the resource.

   - MySQL Server:

```
asadmin delete-jdbc-connection-pool -p admin-port nabPool
asadmin create-jdbc-connection-pool -p admin-port --user admin
--datasourceclassname com.mysql.jdbc.jdbc2.optional.MysqlDataSource
--restype javax.sql.DataSource --property
```

```
"DatabaseName=nab:serverName=mysqlhost:user=nab:password=mysqlpass:portNumb
er=3306:networkProtocol=jdbc" nabPool
asadmin create-jdbc-resource -p admin-port --user admin --connectionpoolid
nabPool jdbc/defaultbackend
```

- Oracle Database:

```
asadmin delete-jdbc-connection-pool -p admin-port nabPool
asadmin create-jdbc-connection-pool --user admin --port admin-port
--restype javax.sql.DataSource --datasourceclassname
oracle.jdbc.pool.OracleDataSource --isconnectvalidatereq=true
--validationmethod table --validationtable DUAL --property
\"url=jdbc\\\:oracle\\\\:thin\\\:@//${dbhost}\\\:${dbport}/${sid}:user=${na
buser}:password=${nabuserpw}\" nabPool
asadmin create-jdbc-resource -p admin-port --user admin --connectionpoolid
nabPool jdbc/defaultbackend
```

8. Restart GlassFish Server after recreating the **connectionpool** and resource.

If you use WebLogic Server:

1. Start the WebLogic Server Administration Console.
2. In the left pane of the Console, under **Domain Structure**, select the domain name.
3. Click **Services** and **Data Sources**.

   JDBC DataSources - **defaultbackend** is displayed in the **Configuration** tab.
4. Select the **defaultbackend** JDBC Data Source name from the list.
5. Select **Configuration** and **General** tab.

   The settings for **defaultbackend** are displayed.
6. Navigate to the **Connection Pool** tab and ensure that the properties are correct.
7. Navigate to **Monitoring,** and then click the **Testing** tab.
8. Select the listed managed server name and click the **Test Data Source** button.

   Success or Error message displays in the Administration Console.

   Note: If the connection fails, verify the Pool properties from the **Connection Pool** tab to ensure all properties are correct.

   Occasionally, Contacts Server may not load the backend even though the connection to pool succeeds. To see the pool and resource data clearly, view the WebLogic Server configuration file. For example, *Weblogic_Domain*/**config/config.xml**
9. Delete and recreate the Connection Pool and JDBC resource from WebLogic Server Administration Console if the cause of the error is unclear.
   a. Click **Lock & Edit** before making changes to the configuration.
   b. Click **Activate Changes** after making the changes and saving the configuration.

Note: If you recreate the JDBC resource, ensure to use the same user name and password that you initially used to create the resource.

If you use WebLogic Server as a container, for creating the JDBC resource, see the discussion about installing and configuring multiple Contacts Server back-end hosts for WebLogic Server manually in the *Contacts Server Installation and Configuration Guide*.

c. Restart WebLogic Server after recreating the connection pool and resource.

10. Verify the MySQL logs for any errors.

Refreshing Domain Information

Contacts Server fetches and caches some domain information that is stored in the LDAP directory, such as domain status. The system does not periodically refresh domain information, unlike user and group information.

If you need to refresh domain information, use one of the following methods:

- Restart the application server.
- Using the **davadmin** command, make a change to any of the LDAP-related configuration options (**base.ldapinfo.***), which causes the server to refresh all cached LDAP data.
- Use the **davadmin cache clear** command to clear the acl, domainmap, auth, and uri caches.

Tuning Directory Server

When your Oracle Directory Server Enterprise Edition contains many tens of thousands of entries, it might become necessary to tune how the directory performs searches. For example, you might experience search time outs or failures from Contacts Server. For more information, see the topic on configuring search limit in *Oracle Fusion Middleware Administrator's Guide for Oracle Directory Server Enterprise Edition*.

Enabling Telemetry Logging

To troubleshoot issues with a particular user or client, it is useful to log all protocol interactions. You can force all telemetry logs by setting the **service.dav.telemetry.forcetelemetry** parameter to **true**. Do not use this setting unless required as it generates lots of data.

To set the **service.dav.telemetry.forcetelemetry** parameter to **true**:

```
davadmin config modify -o service.dav.telemetry.forcetelemetry -v true
```

To enable telemetry logging at a reduced level, set the **service.dav.telemetry.filter** parameter. This parameter takes a space-separated list of request URI prefixes that should be logged. For example:

- **/rest/** logs all RESTful API access.
- **/dav/principals/nabuser1/ /dav/home/nabuser1/** logs all Contacts Server access to **nabuser1**'s account (both principals and home collections, and all the resources underneath).

To set the **service.dav.telemetry.filter** parameter:

```
davadmin config modify -o service.dav.telemetry.filter -v URIs
```

Using the Browser Servlet in GlassFish Server Deployments

You can use a browser servlet to view address books and contacts information from a browser. You might find this helpful when troubleshooting Contacts Server problems.

To access this browser servlet, take any valid **nab** URI and replace the **dav** prefix following **nabserver** with **browse**. For example, in a browser, change the following:

```
http://example.com:3080/nabserver/dav/principals/smithj/addressbook/
```

to:

```
http://example.com:3080/nabserver/browse/home/smithj/addressbook/
```

The servlet returns a a view of the address book's properties. You can navigate among properties and delete them as well. The servlet also has some import function if you want to use a server-side import instead of a client-side import.

The delete and file import features are enabled only when the logging level is set at **FINE** or lower. To specify the logging level, use the **log.dav.errors.loglevel** configuration parameter.

> **Tip:** You can log in with Contacts Server administrator credentials (the default is **nabmaster**) to view multiple accounts with one login. Also, when viewing multiple accounts, clear your browser cache before viewing the next account.

11

Using Contacts Server Notifications

This chapter describes the Oracle Communications Contacts Server notification architecture, how to enable notifications, the different types of notifications, and how to customize notifications.

Overview of Notification Architecture

Contacts Server is capable of generating notifications for changes to the address book data in the database, or for some preset trigger. Notifications are published as Java Message Service (JMS) messages. Contacts Server also includes a JMS consumer program that consumes the JMS notifications and sends email messages to end users. A notification is sent by the server when a user, different than the one being notified, makes a change to the contacts database, for example, by granting an address book permission.

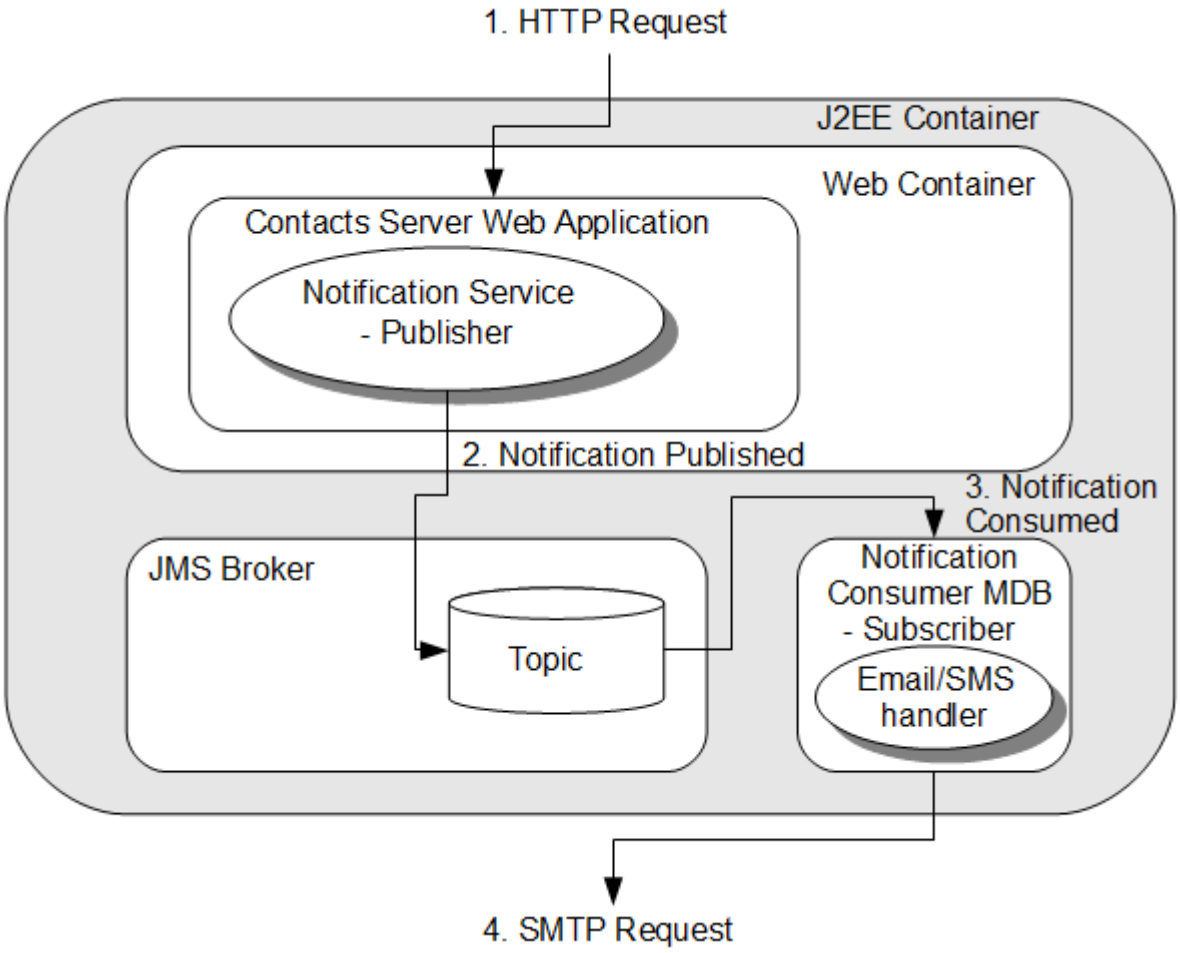
Contacts Server notification services use a publish/subscribe paradigm. Contacts Server publishes messages, in this case, notifications. Receiving clients (the subscribers) receive only those messages that they are interested in.

Contacts Server utilizes the built-in Java Messaging Service (JMS) in the application server to communicate contact data changes. Contacts Server bundles a consumer program that "consumes" this information and sends email for certain subset of the notifications as detailed in "Notification Types." For more information, see the JMS website at:

`http://java.sun.com/products/jms/`

Figure 11–1 shows that the Contacts Server notification service consists of two major components, the *Notification Service* and *Notification Consumer*. The Notification Service component is part of Contacts Server itself, and is the publisher that posts messages of a pre-configured JMS topic managed by the JMS provider. The Notification Consumer component is the subscriber or the message consumer of that JMS topic.

Figure 11–1 Contacts Server Notifications Services Architecture



The Notification Service component provides interfaces for Contacts Server to publish JMS messages to a specific JMS topic (**DavNotificationTopic)** of the JMS broker. The Notification Service component is part of the main Contacts Server servlet that is deployed in the application server web container. The Notification Consumer component listens on the JMS bus for the specific topic (**DavNotificationTopic)** notification messages, consumes the messages, and sends notification email to recipients, if applicable. The consumer checks the notification type and other instructions provided in the JMS message to determine what action is to be taken. The Notification Consumer component message-driven bean (MDB) runs in the application server J2EE container. The consumer MDB is deployed in **EMBEDDED** mode, and thus is running in the same JVM of the J2EE container.

You can choose to write your own customized Notification Consumer programs. See "Writing a Java Messaging Service Consumer".

About Server Email Notifications

Server notifications are used to notify users mostly about changes to their address books due to actions by other users, such as granting permission to an address book. To enable email notifications at a server level, both the **notification.dav.enablejmsnotif** and **notification.dav.enableemailnotif** configuration parameters must be set to **true**.

Enabling Contacts Server Notifications

Table 11–1 describes the Contacts Server notifications that are controlled by the configuration parameters.

Table 11–1 Notification Configuration Parameters

Parameter	Description
notification.dav.enableemailnotif	Controls server-wide email notification. When this parameter is set to **true**, Contacts Server sends email notifications for address book collection creation and sharing (access change). End users can choose to receive notifications either by enabling their own account through Convergence or by requesting that an administrator do so by using the **davadmin** command. These notifications are text emails sent to users. If set to **false**, server-wide email notification is disabled.
notification.dav.enablejmsnotif	Controls server-wide JMS notification. When set to **true**, Contacts Server publishes notifications to the JMS bus. This parameter must be set to **true** for any notification to work.

You can enable or disable these parameters by using Jconsole or the **davadmin** utility. You do not need to restart the application server for a change to these parameters to take effect.

The settings are not cumulative. That is, to receive email notification, not only should **notification.dav.enableemailnotif** be set to **true**, so should **notification.dav.enablejmsnotif**.

Other **notification.dav.*** configuration parameters control items such as the SMTP server to use and its settings, maximum notification payload, location of notification templates, and so on. The **davcore.autocreate.enableemailnotification** parameter determines if notification is enabled by default on a newly created account and the **davcore.autocreate.emailnotificationaddressattr** parameter specifies which LDAP attribute to set as the default notification address when autocreating an account. (The default value is **mail**.) For more details, see "Contacts Server Configuration Parameters".

Enabling Notifications on an Account

To enable notifications for all accounts:

1. Use the **davadmin** command to set the **davcore.autocreate.enableemailnotification** to **true**.

   ```
   davadmin config modify -o davcore.autocreate.enableemailnotification -v true
   ```

2. If necessary, change the value of the LDAP attribute corresponding to **davcore.autocreate.emailnotificationaddressattr**, which is used to set the email notification address during account autocreation. The default value is **mail**.

Modifying Notifications on an Account

Contacts Server stores the values for the **davcore.autocreate.enableemailnotification** and **davcore.autocreate.emailnotificationaddressattr** parameters in the database as properties for each account. You modify these parameters by running the **davadmin account** command.

For more information on the **davadmin account** command see "davadmin account".

Managing Notification Templates

This section describes the Contacts Server notification service in more detail and how to customize notification templates for your deployment.

Topics in this section:

- Notification Types
- Templates, Resource Bundle, and Other Configuration Files
- Customizing Templates
- Preserving Customized Template Files During Upgrade

Notification Types

Table 11–2 describes the notification types. It also lists the payload data, which is the resource content (for example, contact data) in byte array format. Attachments are not included.

Table 11–2 Notification Types

Notification Type	Description	Payload	Contacts Server Consumer Action
AUTOCREATECARD	Initial creation of a user's home collection (and its default sub-collections)	None	Email sent if creation happened. Creation due to user login or explicit account creation by using the **davadmin** command does not trigger an email.
CREATE_COLLECTION	Creation of a address book collection	None	None.
CREATE_CARD_RESOURCE	Creation of an entry in a address book collection	Contact data	None.
DELETE_COLLECTION	Deletion of a address book collection	None	None.
DELETE_CARD_RESOURCE	Deletion of an entry in an address book collection	Contact data	None.
MODIFY_CARD_ RESOURCE	Modification of an entry in an address book collection	Contact data	None.
MOVE_COLLECTION	An address book collection was moved	None	None.
MOVE_RESOURCE	An entry in an address book collection was moved	None	None.
SHARE_ACCOUNT	An account was shared	None	An email is sent if additional permission was granted.
SHARE_CARD_ COLLECTION	An address book collection was shared	None	An email is sent if additional permission was granted.
NONE	Undefined type	Contact data	Not applicable.

The notification message contains a type field that indicates what action triggered the notification and thus helps the consumer decide how to process it.

Templates, Resource Bundle, and Other Configuration Files

This section contains the following topics:

Notification Configuration

You enable or disable notifications and set the values of the SMTP server used by the notification consumer by using the **davadmin** command or Jconsole. See "Contacts Server Configuration Parameters" for details on each of the configuration properties that you can set for notifications.

Resource Bundles

The value of the user's locale/preferred language attribute (defined by the **davcore.ldapattr.preferredlang** configuration parameter) in the user's directory entry is used to localize notification email. The attribute is retrieved from LDAP every time a notification is triggered and is then passed along as part of the notification object being published. If the user does not have any preferred locale/language, it defaults to the consumer module's system's default.

Template Files

Notification templates are files that contain pre-formatted notification messages.

Table 11–3 describes the available notification email templates. In a deployed production environment, by default the templates should be located in the **/config/templates** sub-directory, for example, **/opt/sun/comms/nabserver/config/templates/**. The location of the templates directory is defined by the **notification.dav.configdir** configuration parameter.

Table 11–3 Scenarios That Trigger Notifications and Templates Files Used

Message Type	Notification Type	Template Files	From	To	Description
Auto creation	**AUTOCREATECARD**	**autocreatecard.fmt**	User's address	User's address	Notifies of auto creation of user's home collection due to login.
Address book creation	**CREATE_COLLECTION**	**createaddressbook.fmt**	User's address	User's address	Notifies of an address book being created.
Contact creation	**CREATE_CARD_RESOURCE**	**createcontact.fmt**	User's address	User's address	Notifies of a contact being created.
Address book deletion	**DELETE_CARD_COLLECTION**	**deleteaddressbook.fmt**	User's address	User's address	Notifies of an address book being deleted.

Table 11–3 (Cont.) Scenarios That Trigger Notifications and Templates Files Used

Message Type	Notification Type	Template Files	From	To	Description
Contact delection	**DELETE_CARD_RESOURCE**	**deletecontact.fmt**	User's address	User's address	Notifies of a contact being deleted.
Share contact	**SHARE_ACCOUNT**	**share_account.fmt**	Sharer's email address	Sharee's email address	Notifies of an address book account being shared.
Share address book collection	**SHARE_CARD_COLLECTION**	**shareaddressbook.fmt**	Sharer's email address	Sharer's email address	Notifies of an address book collection being shared.

A recipient list stored in the property, **SUN_NOTIFRECIPIENT**. By default, it's the scheduling address of the LDAP user on behalf of whom the operation is processed. It can be modified through interfaces provided by WCAP or by using the **davadmin** command.

Customizing Templates

Because JavaMail has interfaces to parse an entire string into a MIME message, a notification template file is designed to be a well-formatted email MIME message that contains character sequences denoted by a starting "%{", and an ending "}".

A template contains the following trinket types:

- Resource bundle key: A place holder for locale-specific resource, in the format of ${key};

  For example, trinket ${summary} contains a key "summary" that uniquely identifies a locale-specific object in the resource bundle.

- Value trinket: A place holder for notification field value, in the format of %{trinket};

For a complete list of keys, refer to the **email.properties** file.

For more information on values and trinkets, and template examples, see the topic on customizing templates in *Calendar Server System Administrator's Guide*. Though that guide is written for Calendar Server, the topic also applies to Contacts Server.

Preserving Customized Template Files During Upgrade

Customized notification template files are preserved during a Contacts Server upgrade. Normally, there should be no problem merging customized notification template files during the upgrade. If the upgrade encounters a problem with merging these files, the following message is displayed:

```
log_msg "There are conflicts in merging $file customization"
log_msg "Please finish the merge by manually resolving the conflicts in
$cfgFileNew"
```

The **$file** and **$cfgFileNew** are substituted with actual file names.

Writing a Java Messaging Service Consumer

Contacts Server Notification Services use a publish/subscribe paradigm. All Contacts Server notification messages are posted to a pre-defined JMQ Topic called **DavNotificationTopic**. Each message consists of the associated contact data as the message body and some additional information passed in as properties.

For more information, see the topic on notification message format and sample code in *Calendar Server System Administrator's Guide*. Though that guide is written for Calendar Server, the topic also applies to Contacts Server.

Managing Contacts Server Java Messaging Server Destinations

You can manage Java Messaging Server (JMS) destinations in Contacts Server by using the **imqcmd** command. For a complete list of **imqcmd** options, see the "Command Utility" chapter in *Sun Java System Message Queue 4.1 Administration Guide*.

For more information, see the topic on managing JMS destinations in *Calendar Server System Administrator's Guide*. Though that guide is written for Calendar Server, the topic also applies to Contacts Server.

A

Contacts Server Command-Line Utilities

This appendix provides information about the Oracle Communications Contacts Server command-line utilities.

Overview of the Command-Line Utilities

You use the **davadmin** command to administer Contacts Server. The **davadmin** command is installed in the *ContactsServer_home*/**sbin** directory with user or group **bin/bin** permissions.

The Contacts Server **davadmin** command is identical to the Calendar Server **davadmin** command, with a few changes and enhancements described in this appendix. The Contacts Server **davadmin** command does not include the **calendar** and **calcomponent** commands, nor the **delcomponent** and **upgrade** actions. It also does not include the **-d** option for the **account** command. Finally, the Contacts Server **migration** command is slightly different, as it does not include the **-T** or **-c** options. For complete information about the **davadmin** command, see *Calendar Server System Administrator's Guide*.

Note: The **davadmin** command-line utilities administer aspects of the server and do not affect any LDAP entries.

davadmin Security

The **davadmin** command requires you to authenticate with a user name and password to be able to communicate with the server or database. You can use the **davadmin passfile** operation to store the necessary passwords in an encrypted *wallet* for use by subsequent **davadmin** commands. If you do not store passwords in the wallet, then you must enter them by using a no-echo prompt on the command line. See the topic on the **passfile** operation in *Calendar Server System Administrator's Guide* for more information.

Environment Variables

Table A–1 describes the environment variables that you can use with the various **davadmin** commands.

Table A–1 davadmin Environment Variables

Environment Variable	Description
DAVADMIN_CLIFILE	Specifies the path to the bootstrap file. Can be used instead of the **-F** option.

Table A–1 (Cont.) davadmin Environment Variables

Environment Variable	Description
DAVADMIN_ACCOUNT	Specifies the account. Can be used instead of the **-a** option.

davadmin account

Use this command to administer Contacts Server user accounts in the database.

Location

ContactsServer_home/**sbin**

Syntax

```
davadmin account [ create | delete | list | modify | repair | subscribe |
                   unsubscribe ]
                 [-u id] [-W] [-F clifile] [-H hostname]
                 [-p port] [-s path] [-a account] [-g uniqueid (delete only)]
                 [-y property=value[,property=value...]] [-f file]
                 [-B ldapbaseuri] [-R ldapfilter]
                 [-c collection_path | -C collections_file_path]
                 [-m] [-o] [-v (list only)] [-e] [-r] [-q] [-h]
```

account Operation

Table A–2 describes the actions for the **account** operation.

Table A–2 Actions for account Operation

Action	Description
create	Creates an account for user who has been provisioned in the LDAP Directory Server. The user must have an email address.
delete	Deletes an account.
list	Lists properties of an account.
modify	Modifies an account.
repair	Repairs the user's email address in the database entries after an LDAP email change occurs. When used with the **-o** option, **repair** updates the owner lists of all accounts.
subscribe	Subscribes to an address book belonging to another user. That other user must grant the requesting user access before this can be done.
unsubscribe	Removes an address book from a user's subscription list.

Options for account Operation

Table A–3 describes the options for the **account** operation.

Table A–3 Options for account Operation

Short Option	Long Option	Description
-u	**--userid**	Specifies the application server administrator user name.
-W	**--usepasswordfile**	Specifies the **password** file, if available.
-F	**--clifile**	Specifies the file containing bootstrap information.
-H	**--hostname**	Specifies the server's host name. The default is **localhost**.
-p	**--port**	Specifies the server's administrative port number.
-s	**--secure**	Specifies the path and name of the **trustStore** file for a secure connection (HTTPS).
-a	**--account**	Specifies the account.
-g	**--uniqueid**	Specifies the account described by **uniqueid**. Used only for **delete**, if the **-a** option fails.
-y	**--property**	Specifies a comma-separated list of *property=value* fields. Possible address book properties are: ■ **notifemail** - Email notification enable flag. 0 = disabled, 1 = enabled ■ **notifrecipients** - Recipients of email notifications. Multiple values are separated by a space.
-c	**--collectionuri**	Specifies a collection path in which to subscribe or unsubscribe.
-C	**--collectionuris**	Specifies a local input file with collection paths, one per line, in which to subscribe or unsubscribe.
-f	**--file**	Specifies a local input file with one line for each account, for batch operation. Format is *account*:*property_list*,where *property_list* is optional and contains a comma separated list of *property=value* fields.
-B	**--ldapbaseuri**	Specifies a base URI in LDAP.
-R	**--ldapbaseuri**	Specifies a user search filter in LDAP. Default is **(objectClass=nabuser)**.
-m	**--email**	Used only for the **repair** action. Updates the email addresses due to an email change for users specified with **-a** or **-f** options.
-o	**--ownerlists**	Used only for the **repair** action. Updates the owner lists for all accounts.
-v	**--verbose**	Used for **account list** command. If true, outputs details of each account found. This option is implied if the **-a** option is used.
-e	**--detail**	Prints detailed output.

Table A–3 (Cont.) Options for account Operation

Short Option	Long Option	Description
-r	**--force**	Forces the operation. Does not prompt for confirmation.
-q	**--quiet**	Quiet mode.
-h	**--help**	Displays help.

davadmin addressbook

Use this command to create, modify, list, and delete user address books.

Location

ContactsServer_home/**sbin**

Syntax

```
davadmin addressbook [ create | delete | list | modify ]
                     [-u id] [-W] [-F clifile] [-H hostname]
                     [-p port] [-s path] [-a account] [-n name]
                     [-y property=value[,property=value...]] [-f file]
                     [-e] [-r] [-q] [-h]
```

addressbook Operation

Table A–4 describes the actions for the **addressbook** operation.

Table A–4 Actions for addressbook Operation

Action	Description
create	Creates an address book for user who has been provisioned in the LDAP Directory Server. The user must have an email address.
delete	Deletes an address book.
list	Lists properties of an address book.
modify	Modifies an address book.

Options for addressbook Operation

Table A–5 describes the options for the **addressbook** operation.

Table A–5 Options for addressbook Operation

Short Option	Long Option	Description
-u	**--userid**	Specifies the application server administrator user name.
-W	**--usepasswordfile**	Specifies the **password** file, if available.
-F	**--clifile**	Specifies the file containing bootstrap information.
-H	**--hostname**	Specifies the server's host name. The default is **localhost**.

Table A–5 (Cont.) Options for addressbook Operation

Short Option	Long Option	Description
-p	**--port**	Specifies the server's administrative port number.
-s	**--secure**	Specifies the path and name of the **trustStore** file for a secure connection (HTTPS).
-a	**--account**	Specifies the account.
-n	**--name**	Specifies the name of the address book collection.
-y	**--property**	Specifies a comma-separated list of *property=value* fields. Possible address book properties are: ■ **displayname** - The name of the address book. Defaults to name given with the `-n` option. ■ **description** - Description string. No default. ■ **acl** - The access control string set on the address book. ■ **set-ace** - Set one or more individual ACEs in the ACL. A semi-colon separated list of ACEs. ■ **remove-ace** - Remove one or more individual ACEs from the ACL. A semi-colon separated list of ACE principals. For example: **@**, **@***domain*, *group***@***domain*, or *user***@***domain*.
-f	**--file**	Specifies a local input file with *account*:*addressbook_name*:*property_list*, for batch operation, where *property_list* is optional and contains a comma separated list of *property=value* fields.
-e	**--detail**	Prints detailed output.
-r	**--force**	Forces the operation. Does not prompt for confirmation.
-q	**--quiet**	Quiet mode.
-h	**--help**	Displays help.

davadmin contact

Use this command to list, import, and export address book contacts.

Location

ContactsServer_home/**sbin**

Syntax

```
davadmin contact [ delete | list | import | export ]
                 [-u id] [-W] [-F clifile] [-H hostname]
                 [-p port] [-s path] [-a account] [-n name]
                 [-c contact] [-f format] [-L lang] [-m path]
                 [-x path] [-l logpath] [-r] [-e] [-q] [-h]
```

contact Operation

Table A–6 describes the actions for the **contact** operation.

Table A–6 Actions for contact Operation

Action	Description
delete	Deletes a contact.
list	Lists properties of a contact.
import	Imports a contact.
export	Exports a contact.

Options for contact Operation

Table A–7 describes the options for the **contact** operation.

Table A–7 Options for contact Operation

Short Option	Long Option	Description
-u	**--userid**	Specifies the application server administrator user name.
-W	**--usepasswordfile**	Specifies the **password** file, if available.
-F	**--clifile**	Specifies the file containing bootstrap information.
-H	**--hostname**	Specifies the server's host name. The default is **localhost**.
-p	**--port**	Specifies the server's administrative port number.
-s	**--secure**	Specifies the path and name of the **trustStore** file for a secure connection (HTTPS).
-a	**--account**	Specifies the account that owns the address book.
-n	**--name**	Specifies the name of the address book collection.
-c	**--contact**	Specifies the contact URI for which to request content.
-f	**--format**	Specifies the export format (**vcard3** or **csv**). The default is **vcard3**.
-L	**--language**	Specifies the export language.
-m	**--import-path**	Specifies the path to the file on the host that contains data to be imported.
-x	**--export-path**	Specifies the path to the file on the host to receive the exported data.
-l	**--logpath**	Specifies the path to the location of the log directory.
-r	**--force**	Forces the operation. Does not prompt for confirmation.
-e	**--detail**	Prints detailed output.
-q	**--quiet**	Quiet mode.
-h	**--help**	Displays help.

davadmin ctgroup

Use this command to create, modify, list, import, export, and delete contact groups from a user's address book.

Location

ContactsServer_home/**sbin**

Syntax

```
davadmin ctgroup [ create | delete | list | modify | import | export ]
                 [-u id] [-W] [-F clifile] [-H hostname]
                 [-p port] [-s path] [-a account] [-n name]
                 [-c contactgroup] [-g groupname] [-M members]
                 [-E email_addresses] [-f format] [-L lang]
                 [-m path] [-x path] [-l logpath] [-r] [-h]
```

ctgroup Operation

Table A–8 describes the actions for the **ctgroup** operation.

Table A–8 Actions for ctgroup Operation

Action	Description
create	Creates an address book contact group.
delete	Deletes an address book contact group.
list	Lists properties of an address book contact group.
modify	Modifies the properties of an address book contact group.
import	Imports an address book contact group.
export	Exports an address book contact group.

Options for ctgroup Operation

Table A–9 describes the options for the **ctgroup** operation.

Table A–9 Options for ctgroup Operation

Short Option	Long Option	Description
-u	**--userid**	Specifies the application server administrator user name.
-W	**--usepasswordfile**	Specifies the **password** file, if available.
-F	**--clifile**	Specifies the file containing bootstrap information.
-H	**--hostname**	Specifies the server's host name. The default is **localhost**.
-p	**--port**	Specifies the server's administrative port number.
-s	**--secure**	Specifies the path and name of the **trustStore** file for a secure connection (HTTPS).
-a	**--account**	Specifies the account that owns the address book.

Table A–9 (Cont.) Options for ctgroup Operation

Short Option	Long Option	Description
-n	**--name**	Specifies the name of the address book collection.
-c	**--contactgroup**	Specifies the contact URI for which to request content.
-g	**--groupname**	Specifies the contact group name.
-M	**--members**	Specifies a comma-separated list of contact group members.
-E	**--email_addresses**	Specifies a comma-separated list of contact group members' email addresses.
-f	**--format**	Specifies the export format (**vcard3**).
-L	**--language**	Specifies the export language.
-m	**--import_path**	Specifies the path to the file on the host that contains data to be imported.
-x	**--export_patch**	Specifies the path to the file on the host to receive the exported data.
-l	**--logpath**	Specifies the path to the location of the log directory.
-r	**--force**	Forces a delete operation.
-h	**--help**	Displays help.

ctgroup Examples

The following examples show how to use the **davadmin ctgroup** command.

To create a contact group in the personal address book with the name **group1**:

```
davadmin ctgroup create -a john.smith@example.com -g group1 -n personal
```

To create a contact group in the personal address book with the name **group1** plus an email address and two members:

```
davadmin ctgroup create -a john.smith@example.com -g group1 -n personal -E
my.group@example.com -M "urn:uuid:aaaaaaaaa,urn:uuid:bbbbbbbbbb"
```

To list all the contact groups in the user's default address book:

```
davadmin ctgroup list -a john.smith@example.com
```

To list the details of a contact group:

```
davadmin ctgroup list -a john.smith@example.com -c 1384904616388-1758-GROUP.vcf
```

To delete a contact group:

```
davadmin ctgroup delete -a john.smith@example.com -c 1384904616388-1758-GROUP.vcf
```

To remove all the current members of a group, replace them with one member, and change the group name to **newgroup**:

```
davadmin ctgroup modify -a john.smith@example.com -c 1384904616388-1758-GROUP.vcf
-M "urn:uuid:ccccccccc" -g newgroup
```

davadmin db

Use this command to perform database related operations, such as backing up and restoring the database, and upgrading the database schema.

Unlike other **davadmin** commands that communicate with the application server, the **davadmin db** commands communicate directly with the back-end database, and thus require that you specify the database host name, port, and password.

Although the **davadmin db** commands are not related to the application server like the other **davadmin** commands, **davadmin db** commands do still use parameter values in the **davadmin.properties** file if applicable.

Because each database back end is associated with a database host name, port, document store, and so on, in a multiple back-end deployment, use a unique **clifile** (specified with the **-F** option) for each back end in the deployment.

In a non-default deployment or multiple back-end deployment, properly define options such as (**-d** *database*) and (**-u** *dbuser*), which might need to use specific and not default values.

Syntax

```
davadmin db [ backup | init | list | restore | schema_version |
              schema_fullupgrade | schema_preupgrade ]
             [-h] [-e] [-W] [-t dbtype] [-H dbhost] [-p dbport] [-F clifile]
             [-u dbuserid]  [-d database] [-s truststore] [-b blockfactor]
             [-D domain] [-a account_mail] [-T token] [-O] [-i path]
             [-c] [-A docstore] [-z preupgradefunction] [-k backup_file]
```

db Operation

Table A–10 describes the actions for the **davadmin db** operation.

Table A–10 Actions for db Operation

Command	Description
backup	Backs up a database.
init	Completely initializes the database. **Caution**: All data will be lost.
list	List contents of a backup file. This is the default action if not included on the command line.
restore	Restores the contents of a database.
schema_version	Displays version information for the database, connector, and product schema number.
schema_ fullupgrade	Provides an optional way to perform a full upgrade of the database schema. For more information about upgrading database schema and upgrading Contacts Server, see "Upgrading Contacts Server" in *Contacts Server Installation and Configuration Guide*.
schema_ preupgrade	Provides an optional way to perform a pre-upgrade on the database schema. For more information about upgrading database schema and upgrading Contacts Server, see "Upgrading Contacts Server" in *Contacts Server Installation and Configuration Guide*.

Caution: Do not run either the **schema_fullupgrade** or **schema_preupgrade** without fully understanding the impact on your Contacts Server deployment.

The **davadmin db backup**, **list**, and **restore** commands require that you specify the associated document store by using the **-A** option, or the **docstore** option in the **clifile**.

Note: If you are using a remote document store, you must set the document store password on the Contacts Server host by using the **davadmin passfile** command and that password must match the one set for the remote document store. This password is used whenever the backup or restore commands access the remote document store.

Options for db Operation

Table A–11 describes the options for the **db** operation (in addition to the common options).

Table A–11 Options for db Operation

Short Option	Long Option	Description	Available for Following Actions
-d	**--database**	Specifies the name of the Contacts Server store to be saved or updated. The default is **nab**. For MySQL Server, this is the database name. For Oracle Database, this is the network service name (not SID nor pdb name).	**backup**, **restore**, **list**
-H	**--dbhost**	Specifies the database host. The default is **localhost**.	All
-p	**--dbport**	Specifies the database port. The default is 3306.	All
-u	**--dbuser**id	Specifies the database user. For MySQL Server, this is the connecting user name. For Oracle DB, this is the user/schema name.	All
-k	**--bkfile**	Specifies the path of the file where the database information is to be saved. Required.	**backup**, **restore**, **list**
-b	**--bkfactor**	Specifies blocking factor used during backup. The default is 20.	**backup**, **restore**, **list**
-T	**--token**	Specifies the incremental backup token or start time in milliseconds.	**backup**
-D	**--domain**	Domain name for per domain backup.	**backup**
-a	**--account**	User account email value for per user backup.	**backup**
-i	**--ipath**	Specifies the internal path for partial list or restore.	**restore**, **list**
-c	**--contents**	Lists the resources and header.	**list**

Table A–11 (Cont.) Options for db Operation

Short Option	Long Option	Description	Available for Following Actions
-A	**--docstore**	Specifies the document store (remote store specified as *host*:*port* or local store by fully qualified path to root of document store).	**backup**, **restore**, **list**
-t	**--dbtype**	Specifies the type of database, either **mysql** or **oracle**. The default is **mysql.**	All
-O	**--overwrite**	Overwrites existing data.	**backup**, **restore**
-s	**--dbsecure**	Supplies the path to the **trustStore** file that contains the SSL certificate for secure communications with the remote document store.	**backup**, **restore**
-z	**--dbupgradefunction**	Specifies to run the pre-upgrade function(s) on the database. Caution: Do not run **schema_preupgrade** without fully understanding the impact on your Contacts Server deployment. For more information, see "Upgrading Contacts Server" in *Contacts Server Installation and Configuration Guide*. The pre-upgrade functions are: ■ **services-up** - Processes all pre-upgrade functions that can be run with old services up. (Online DDL) Otherwise and most commonly, pre-upgrade functions must be run with services shut down. ■ **services-down** - Processes all pre-upgrade functions that cannot be run with services up. ■ **all** - Processes all available pre-upgrade functions. Services should be shut down. For a list of available functions by release, see "Preupgrade Functions" in *Contacts Server Installation and Configuration Guide*. Unless otherwise specified, never run pre-upgrade functions with services up. In addition, always back up your database before upgrading. Preupgrade functions are listed for each release. Some function names process multiple preupgrade functions.	**schema_preupgrade**

davadmin db Examples

- To perform a full database backup:

```
davadmin db backup -k backup_file
```

- To perform a full backup for a particular user:

```
davadmin db backup -k backup_file -a john.smith@example.com
```

- To perform an incremental backup:

```
davadmin db backup -k backup_file -T token obtained from last full backup
```

- To perform a full backup for a particular domain:

```
davadmin db backup -k backup_file -D sesta.com
```

- To list the contents of the backup file:

```
davadmin db list -c backup_file
```

When the **davadmin db list -c** command retrieves backup file content, it goes through the checksums and is thus a way to verify the structure of the backup file itself.

- To perform a restore from a backup file:

```
davadmin db restore -k backup_file
```

- To restore an entire user from a backup file:

```
davadmin db restore -O -e -W -k /export-filepath -i
"hosted.domain/mail:given.surname@hosted.domain/" -H mysqlcalhost -A  matching_
document_store_host:8007 > /log_output_file
```

- To restore only the default 'addressbook:'

```
davadmin db restore -O -e -W -k /export-filepath -i
"hosted.domain/mail:given.surname@hosted.domain/addressbook/" -H mysqlcalhost
-A  matching_document_store_host:8007 > /log_output_file
```

- To restore only an addressbook named **Soccer**:

```
davadmin db restore -O -e -W -k /export-filepath -i
"hosted.domain/mail:given.surname@hosted.domain/Soccer/" -H mysqlcalhost -A
matching_document_store_host:8007 > /log_output_file
```

- To back up using SSL and the trustStore file:

```
davadmin db backup -k /tmp/backup_file -O -A docstore_host.example.com:8008 -s
/my_home/my_truststore -u mysql
```

- To process a database schema preupgrade:

```
davadmin db schema_preupgrade -z preupgrade_function
```

This command processes one preupgrade function. A preupgrade function is an upgrade change to the database, which can be run before the formal upgrade. This command does not change the database schema version.

- To process all available preupgrade functions:

```
davadmin db schema_preupgrade -z all
```

Prior to running this command, ensure that all services are shut down.

- To process all preupgrade functions that cannot be run with services down:

```
davadmin db schema_preupgrade -z services-down
```

davadmin migration

Use this command to migrate a user's Personal Address Book (PAB) to Contacts Server.

Location

ContactsServer_home/**sbin**

Syntax

```
davadmin migration [ migrate | status ]
                   [-u id] [-W] [-F clifile] [-H hostname]
                   [-p port] [-s path] [-a account] [-X migrationadminuser]
                   [-f file] [-L migrationserver] [-S]
                   [-B ldapbaseuri] [-R ldapfilter] [-l logpath] [-G] [-h]
```

migration Operation

Table A–12 describes the actions for the **migration** operation.

Table A–12 Actions for migration Operation

Action	Description
migrate	Migrates the address book from the Personal Address Book host to the Contacts Server host.
status	Provides a status on the migration.

Options for migrate Operation

Table A–13 describes the options for the **migration** operation.

Table A–13 Options for migration Operation

Short Option	Long Option	Description
-u	**--userid**	Specifies the application server administrator user name.
-W	**--usepasswordfile**	Specifies the **password** file, if available.
-F	**--clifile**	Specifies the file containing bootstrap information.
-H	**--hostname**	Specifies the server's host name. The default is **localhost**.
-p	**--port**	Specifies the server's administrative port number.
-s	**--secure**	Specifies the path and name of the **trustStore** file for a secure connection (HTTPS).
-a	**--account**	Specifies the account.
-X	**--migrationadminuser**	Specifies the LDAP auth user for the Personal Address Book Directory Server host.
-f	**--file**	Specifies a local input file with one line for each user to be migrated, for batch operation.
-L	**--migrationserver**	Specifies the Personal Address Book Directory server host's name and port number as *server*:*port*.
-S	**--clientssl**	Specifies to use SSL when making client connections.
-B	**--ldapbaseuri**	Specifies a base URI in LDAP.

Table A–13 (Cont.) Options for migration Operation

Short Option	Long Option	Description
-R	**--ldapbaseuri**	Specifies a user search filter in LDAP. Default is **(objectClass=nabuser)**.
-l	**--logpath**	Specifies the path to use for the migration log file.
-G	**--tag**	Specifies the tag to use to access the migration status.
-h	**--help**	Displays help.

B

Contacts Server Configuration Parameters

This appendix provides information about the Oracle Communications Contacts Server configuration parameters.

davserver.properties File

The *ContactsServer_home*/**config/davserver.properties** file contains the main configuration settings. It consists of configuration parameters and their current values.

> **Caution:** Do not edit this file by hand. Always use the **davadmin** command to set configuration parameters.

The format of the **davserver.properties** file is:

```
parameter=value
parameter=value
:
:
```

Document Store Configuration Parameters

The *ContactsServer_home*/**config/ashttpd.properties** file contains the document store configuration parameters. Table B–1 describes the parameters in the **ashttpd.properties** file.

Table B–1 ashttpd.properties File Parameters

Parameter	Description	Default Value
service.host	Specifies the server host.	*
service.port	Specifies the server port number.	**8008**
store.datadir	Specifies the data directory.	**/var/opt/sun/comms/nabserver**
store.lockdir	Specifies the lock directory.	**/var/opt/sun/comms/nabserver/lock**
store.loglevel	Log level	**INFO**
store.sslkeystorepath	Keystore for the server private key	**/var/opt/sun/comms/nabserver/config/dskeystore.jks**
store.sslprotocols	SSL protocols	**TLSv1 TLSv1.1 TLSv1.2**
store.usessl	Use SSL to communicate with document store client.	**false**

The format of the **ashttpd.properties** file is the following:

```
key=value
key=value
:
:
```

Each line in the **ashttpd.properties** file stores a single property. Do not include a space before and after *key* and *value*. When the host uses multiple network interfaces, and the host should bind to only one, specify that interface with the **service.host** configuration parameter.

davadmin.properties File

You can provide options to the **davadmin** command by including them in the *ContactsServer_home*/**config/davadmin.properties** file.

Table B–2 describes the parameters in the **davadmin.properties** file.

Table B–2 davadmin.properties File Parameters

Parameter	Description
userid	Specifies the application server Administrator user ID.
hostname	Specifies the application server host name.
port	Specifies the application server administration port (JMX connector port).
secure	Specifies the path to the truststore file used for a secure connection (HTTPS) to the application server.
dbtype	Specifies the type of database, either **mysql** or **oracle**.
dbhost	Specifies the database host.
dbport	Specifies the database port.
dbuserid	Specifies the MySQL Server or Oracle Database user ID for database commands.
sslprotocols	Specifies the supported SSL protocols (**TLSv1**, **TLSv1.1**, and **TLSv1.2**) for the JMX proxy to communicate with the management beans in the server.

The format of the **davadmin.properties** file is:

```
parameter=value
parameter=value
:
:
```

corpdirnames-*lang*.properties File

You can customize the corporate directory **displayname** extension by using the *ContactsServer_home*/**config/corpdir/corpdirnames-***lang***.properties** file. The **displayname** extension enables you to provide a localized customized name for the corporate directory.

To change the default value, edit the **idtag** in the **corpdirnames-***lang***.properties** file. An **idtag** lookup is performed to a map of corporate directory names in various languages.

The following languages are supported:

- English
- French
- Spanish
- German
- Japanese
- Korean
- Simplified Chinese
- Traditional Chinese

For example, in the following URL, the directory name corresponding to the **idtag** "id2" in the **corpdirnames-***lang***.properties** file is returned when querying a list of public directories for the deployment in the given language:

```
ldap://virtuallistpool/o=HQ,o=isp??sub?(objectclass=*)?displayname="id2"
```

If you do not set the **displayname** in the URL, the URL uses the default **idtag** of "id1," which has the value "Corporate Directory," when performing the lookup in the English language.

Contacts Server Configuration Parameters

Table B–3 lists the configuration parameters and descriptions for Contacts Server.

Table B–3 Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
base.ldapinfo.cachesize	integer	Size of the LDAP authentication cache.	**1000**	**1**	**1000000**	8.0
base.ldapinfo.cachettl	integer (seconds)	Time to live (in seconds) of cached LDAP authentication info.	**60**	**1**	**Maximum int value**	8.0
base.ldapinfo.dcroot	string	Root of DC tree (Schema 1) or of the domain and users tree (Schema 2) in Directory Server.	**o=isp**	N/A	N/A	8.0
base.ldapinfo.defaultdomain	string	Default domain.	**demo.example.com**	N/A	N/A	8.0
base.ldapinfo.domainattrs	string	Space separated list of LDAP attributes to use when retrieving domain information.	**nabStatus nabDomainNames nabDomainAcl externalAuthPreUrlTemplate externalAuthPostUrlTemplate corpDirectoryUrl**	N/A	N/A	8.0
base.ldapinfo.loginseparator	string	Characters to be used as login separator (between user ID and domain).	**@**	N/A	N/A	8.0
base.ldapinfo.schemalevel	integer	Schema level.	**2**	**1**	**2**	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
base.ldapinfo.searchfilter	string	Search filter to look up users during authentication when none is specified in the **inetDomainSearchFilter** for the domain. The syntax is the same as **inetDomainSearchFilter** (see *Communications Suite Schema Reference*).	**(uid=%U)**	N/A	N/A	8.0
base.ldapinfo.serviceadmindn	string	DN of single admin in LDAP in absence of admin group.	N/A	N/A	N/A	8.0
base.ldapinfo.serviceadminsgroupdn	string	DN of service admins group in LDAP.	N/A	N/A	N/A	8.0
base.ldapinfo.userattrs	string	Space separated list of LDAP attributes to retrieve from user entries during the authentication phase.	**mail ismemberof**	N/A	N/A	8.0
base.ldapinfo.authldap.binddn	string	DN to use when authenticating.	N/A	N/A	N/A	8.0
base.ldapinfo.authldap.bindpassword	password	Password to use when authenticating.	N/A	N/A	N/A	8.0
base.ldapinfo.authldap.ldaphost	string	Space-delimited list of host names. Each host name can include a trailing colon and port number.	**localhost:389**	N/A	N/A	8.0
base.ldapinfo.authldap.ldappoolrefreshinterval	integer (minutes)	Length of elapsed time until the failover Directory Server reverts back to the primary Directory Server. If set to **-1**, do not refresh.	**1**	**1**	**60**	8.0
base.ldapinfo.authldap.ldappoolsize	integer	Maximum number of connections for this pool.	**10**	**1**	**100**	8.0
base.ldapinfo.authldap.ldapport	integer	Port number to which to connect. Ignored for any host name that includes a colon and port number.	**389**	**0**	**65535**	8.0
base.ldapinfo.authldap.ldaptimeout	integer (seconds)	Timeout for all LDAP operations.	**60**	**1**	**3600**	8.0
base.ldapinfo.authldap.ldapusessl	boolean	Use SSL to connect to the LDAP host.	**false**	N/A	N/A	8.0
base.ldapinfo.authldap.sslprotocols	string	Specifies a space-delimited list of the supported SSL protocols to communicate with the back-end LDAP service.	**TLSv1 TLSv1.1 TLSv1.2**	N/A	N/A	8.0.0.1
base.ldapinfo.ugldap.binddn	string	DN to use when authenticating.	N/A	N/A	N/A	8.0
base.ldapinfo.ugldap.bindpassword	password	Password to use when authenticating.	N/A	N/A	N/A	8.0
base.ldapinfo.ugldap.ldaphost	string	Space-delimited list of host names. Each host name may include a trailing colon and port number.	**localhost:389**	N/A	N/A	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
base.ldapinfo.ugldap.ldappoolrefreshinterval	integer (minutes)	Length of elapsed time until the failover Directory Server reverts back to the primary Directory Server. If set to **-1**, do not refresh.	**1**	**1**	**60**	8.0
base.ldapinfo.ugldap.ldappoolsize	integer	Maximum number of connections for this pool.	**10**	**1**	**100**	8.0
base.ldapinfo.ugldap.ldapport	integer	Port number to which to connect. Ignored for any host name that includes a colon and port number.	**389**	**0**	**65535**	8.0
base.ldapinfo.ugldap.ldaptimeout	integer (seconds)	Timeout for all LDAP operations.	**60**	**1**	**3600**	8.0
base.ldapinfo.ugldap.ldapusessl	boolean	Use SSL to connect to the LDAP host.	**false**	N/A	N/A	8.0
base.ldapinfo.ugldap.sslprotocols	string	Specifies a space-delimited list of the supported SSL protocols to communicate with the back-end LDAP service.	**TLSv1 TLSv1.1 TLSv1.2**	N/A	N/A	8.0.0.1
base.ldappool.*.binddn	string	DN to use when authenticating.	N/A	N/A	N/A	8.0
base.ldappool.*.bindpassword	password	Password to use when authenticating.	N/A	N/A	N/A	8.0
base.ldappool.*.ldaphost	string	Space-delimited list of host names. Each host name can include a trailing colon and port number.	**localhost:389**	N/A	N/A	8.0
base.ldappool.*.ldappoolrefreshinterval	integer (minutes)	Length of elapsed time until the failover Directory Server reverts back to the primary Directory Server. If set to **-1**, do not refresh.	**1**	**1**	**60**	8.0
base.ldappool.*.ldappoolsize	integer	Maximum number of connections for this pool.	**10**	**1**	**100**	8.0
base.ldappool.*.ldapport	integer	Port number to which to connect. Ignored for any host name that includes a colon and port number.	**389**	**0**	**65535**	8.0
base.ldappool.*.ldaptimeout	integer (seconds)	Timeout for all LDAP operations.	**60**	**1**	**3600**	8.0
base.ldappool.*.ldapusessl	boolean	Use SSL to connect to the LDAP host.	**false**	N/A	N/A	8.0
base.ldappool.*.sslprotocols	string	Specifies a space-delimited list of the supported SSL protocols for the LDAP pool to communicate with the back-end LDAP service.	**TLSv1 TLSv1.1 TLSv1.2**	N/A	N/A	8.0.0.1
davcore.acl.aclcachesize	integer	Maximum number of ACL entries kept in cache. Entries are removed from the cache only when this maximum is reached or when ACL configuration parameters are changed. If set to **0**, indicates no cache.	**1000**	**0**	**Maximum int value**	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
davcore.acl.aclcachettl	integer (seconds)	Maximum amount of time (in seconds) that an ACL entry can be kept in cache.	**60**	**1**	**Maximum int value**	8.0
davcore.attachment.enable	boolean	Enable or disable attachments.	**true**	N/A	N/A	8.0
davcore.auth.cert.enable	boolean	Enable certificate-based client authentication.	**false**	N/A	N/A	8.0
davcore.auth.cert.fallback	boolean	Fallback to user name and password authentication	**true**	N/A	N/A	8.0
davcore.autocreate.displaynameattr	string	LDAP attribute, whose value is used to set display name, during autocreation. Default setting used on autocreation.	**cn**	N/A	N/A	8.0
davcore.autocreate.emailnotificationaddressattr	string	LDAP attribute, whose value is used to set email notification address, during autocreation. Default setting used on autocreation.	**mail**	N/A	N/A	8.0
davcore.autocreate.enableautocreate	boolean	Enable autocreate operation. Default setting used on autocreation.	**true**	N/A	N/A	8.0
davcore.autocreate.enableemailnotification	boolean	Enable email notification. Default setting used on autocreation.	**true**	N/A	N/A	8.0
davcore.homeuri.*.backendid	string	When it is determined that a URI matches the pattern, this backendid template is used to identify the back end hosting this resource. The template can reference the variables **$1**, **$2**, and so on, saved during the pattern matching. The template can also reference LDAP attributes of the subject matching the **subjectfilter** attribute using the `${attrname}` syntax or the `${attrname,defaultvalue}` syntax. If this parameter is not set, the **uriinfo.backendidtemplate** parameter is used.	N/A	N/A	N/A	8.0
davcore.homeuri.*.rank	integer	When multiple URI patterns are configured, this value determines the order in which to evaluate those URI patterns. A lower number indicates that this pattern should be evaluated first.	**1**	**0**	**Maximum int value**	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
davcore.homeuri.*.subjectdomain	string	When it is determined that a URI matches the pattern, this domain template is used to identify the subject owning this resource. The template can reference the variables **$1**, **$2**, and so on, saved during the pattern matching. For example, if **subjectdomain** is set to **$2**, and using the URI in the uripattern example, the domain of the subject is **example.com**. If empty, indicates the default domain.	**$2**	N/A	N/A	8.0
davcore.homeuri.*.subjectfilter	string	When it is determined that a URI matches the pattern, this LDAP filter template is used to identify the subject owning this resource. The template can reference the variables **$1**, **$2**, and so on, saved during the pattern matching. For example, if **subjectfilter** is set to **(mail=$1@$2)**, and using the URI in the uripattern example, the LDAP filter becomes **(mail=john@example.com)**. If empty, indicates that this namespace is not associated with a particular subject.	**(mail=$1@$2)**	N/A	N/A	8.0
davcore.homeuri.*.uripattern	string	Regex pattern to be matched by the URI. This pattern can contain regex groups (identified by () parenthesis) that are saved into **$1**, **$2**, and so on. The last regex group identifies the local path if there is any. For example, if the pattern is **\^/home/([^/]+)@([^/]+)(/\z|/.*)**, the URI **/home/john@example.com/addressbook/** matches that pattern. **$1** is set to the value **john**, **$2** is set to the value **example.com**, and the local path is **/addressbook**.	**^/home/([^/]+)@([^/]+)(/\z|/.*)**	N/A	N/A	8.0
davcore.ldapattr.commonname	string	Common name attribute.	**cn**	N/A	N/A	8.0
davcore.ldapattr.corpdirectoryurl	string	LDAP attribute to locate a custom external corporate directory for this domain.	**corpDirectoryUrl**	N/A	N/A	8.0
davcore.ldapattr.davstore	string	Logical back-end id attribute.	**nabStore**	N/A	N/A	8.0
davcore.ldapattr.dngroupmember	string	Attributes for members in an LDAP group.	**uniquemember**	N/A	N/A	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
davcore.ldapattr.externalauthposturltemplate	string	LDAP attribute that determines whether external authentication should do a post auth lookup against this domain.	**externalAuthPostUrlTemplate**	N/A	N/A	8.0
davcore.ldapattr.externalauthpreurltemplate	string	LDAP attribute that determines whether external authentication should be used against this domain.	**externalAuthPreUrlTemplate**	N/A	N/A	8.0
davcore.ldapattr.groupobject	string	Space separated list of object class values indicating an LDAP group.	**groupofuniquenames groupofurls inetmailgroup**	N/A	N/A	8.0
davcore.ldapattr.inetresourcestatus	string	LDAP attribute for global status of resources.	**inetresourcestatus**	N/A	N/A	8.0
davcore.ldapattr.inetuserstatus	string	LDAP attribute for status of user's account with regards to global service access.	**inetuserstatus**	N/A	N/A	8.0
davcore.ldapattr.mail	string	Mail attribute.	**mail**	N/A	N/A	8.0
davcore.ldapattr.mailalternateaddress	string	Space separated list of alternate mail attributes.	**mailAlternateAddress**	N/A	N/A	8.0
davcore.ldapattr.mailgroupmember	string	Attributes for members in an LDAP group.	**mgrprfc822mailmember**	N/A	N/A	8.0
davcore.ldapattr.memberattr	string	LDAP attribute listing the groups of which the entry is a member.	**ismemberof**	N/A	N/A	8.0
davcore.ldapattr.nabstatus	string	Contacts Server status attribute.	**nabstatus**	N/A	N/A	8.0
davcore.ldapattr.preferredlang	string	Language attribute.	**preferredLanguage**	N/A	N/A	8.0
davcore.ldapattr.resourcetype	string	LDAP attribute to use to determine the CUTYPE (ROOM versus RESOURCE) of a resource. The CUTYPE of users and groups is not based on this attribute. The attribute value can take the following values: * location and room are mapped to a CUTYPE of ROOM. * thing and resource are mapped to a CUTYPE of RESOURCE. * other values are mapped to RESOURCE.	**kind**	N/A	N/A	8.0
davcore.ldapattr.uid	string	User ID attribute.	**uid**	N/A	N/A	8.0
davcore.ldapattr.urlgroupmember	string	Attributes for members in an LDAP group.	**memberurl**	N/A	N/A	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
davcore.otheruri.*.backendid	string	When it is determined that a URI matches the pattern, this backendid template is used to identify the back end hosting this resource. The template can reference the variables **$1**, **$2**, and so on, saved during the pattern matching. The template also references LDAP attributes of the subject matching the **subjectfilter**, using the `${attrname}` syntax or the `${attrname,defaultvalue}` syntax. If this parameter is not set, the **uriinfo.backendidtemplate** parameter is used.	N/A	N/A	N/A	8.0
davcore.otheruri.*.rank	integer	When multiple URI patterns are configured, this value determines the evaluation order. A lower number indicates that this pattern should be evaluated first.	**1**	**0**	**Maximum int value**	8.0
davcore.otheruri.*.subjectdomain	string	When it is determined that a URI matches the pattern, this domain template is used to identify the subject owning this resource. The template can reference the variables **$1**, **$2**, and so on, saved during the pattern matching. For example, if the **subjectdomain** is set to **$2**, and using the URI in the uripattern example, the domain of the subject is **example.com**. If empty, indicates the default domain.	**$2**	N/A	N/A	8.0
davcore.otheruri.*.subjectfilter	string	When it is determined that a URI matches the pattern, this LDAP filter template is used to identify the subject owning with this resource. The template can reference the variables **$1**, **$2**, and so on, saved during the pattern matching. For example, if the **subjectfilter** is set to **(mail=$1@$2)**, and using the URI in the uripattern example, the LDAP filter becomes **(mail=john@example.com)**. Can be empty, indicating that this namespace is not associated with a particular subject.	**(mail=$1@$2)**	N/A	N/A	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
davcore.otheruri.*.uripattern	string	Regex pattern to be matched by the uri. This pattern can contain regex groups (identified by () parenthesis) which is saved into **$1**, **$2**, and so on. The last regex group identifies the local path if there is any. For example, if the pattern is **^/home/([^/]+)@([^/]+)(/\z|/.*)**, the URI **/home/john@example.com/addressbook/** matchws that pattern. **$1** is set to the value **john**, **$2** is set to the value **example.com**, and the local path is **/addressbook**.	**^/home/([^/]+)@([^/]+)(/\z|/.*)**	N/A	N/A	8.0
davcore.principalsuri.*.backendid	string	When it is determined that a URI matches the pattern, this backendid template is used to identify the back end hosting this resource. The template can reference the variables **$1**, **$2**, and so on, saved during the pattern matching. The template can also reference LDAP attributes of the subject matching the **subjectfilter**, using the **${attrname}** syntax or the **${attrname,defaultvalue}** syntax. If this parameter is not set, the **uriinfo.backendidtemplate** parameter is used.	N/A	N/A	N/A	8.0
davcore.principalsuri.*.rank	integer	When multiple URI patterns are configured, this value determines the evaluation order. A lower number indicates that this pattern should be evaluated first.	**1**	**0**	**Maximum int value**	8.0
davcore.principalsuri.*.subjectdomain	string	When it is determined that a URI matches the pattern, this domain template is used to identify the subject owning with this resource. The template can reference the variables **$1**, **$2**, and so on, saved during the pattern matching. For example, if the **subjectdomain** is set to **$2**, and using the URI in the uripattern example, the domain of the subject is **example.com**. Can be empty to indicate the default domain.	**$2**	N/A	N/A	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
davcore.principalsuri.*.subjectfilter	string	When it is determined that a URI matches the pattern, this LDAP filter template is used to identify the subject owning with this resource. The template can reference the variables **$1**, **$2**, and so on, saved during the pattern matching. For example, if the **subjectfilter** is set to **(mail=$1@$2)**, and using the URI in the uripattern example, the LDAP filter becomes **(mail=john@example.com)**. Can be empty, to indicate that this namespace is not associated with a particular subject.	**(mail=$1@$2)**	N/A	N/A	8.0
davcore.principalsuri.*.uripattern	string	Regex pattern to be matched by the URI. This pattern can contain regex groups (identified by () parenthesis) which are saved into **$1**, **$2**, and so on. The last regex group identifies the local path if there is any. For example, if the pattern is **^/home/([^/]+)@([^/]+)(/\z|/.*)**, the URI **/home/john@example.com/addressbook/** matches that pattern. **$1** is set to the value **john**, **$2** is set to the value **example.com**, and the local path is **/addressbook**.	**^/home/([^/]+)@([^/]+)(/\z|/.*)**	N/A	N/A	8.0
davcore.reverseuri.*.backendid	string	Back-end id on which to apply this reverse mapping. There should be only one mapping per back end.	N/A	N/A	N/A	8.0
davcore.reverseuri.*.uritemplate	string	Canonical form of the URI prefix for this back end. This template should have a corresponding uripattern. It should not end with a slash. The template can reference LDAP attributes of the subject, using the **${attrname}** syntax or the **${attrname,defaultvalue}** syntax. The **${domain}** syntax can be used to reference the domain of the subject. If no template is defined for a given back end, the **uriinfo.defaulthomeuritemplate** parameter is used.	N/A	N/A	N/A	8.0
davcore.serverdefaults.exportconfigdir	filepath	Directory path for export XSL transformation files.	**config/export**	N/A	N/A	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
davcore.serverdefaults.importconfigdir	filepath	Directory path for import properties and translation files.	**config/import**	N/A	N/A	8.0
davcore.serverdefaults.jsonprefix	string	Default prefix to append to all JSON output.	**{}&&**	N/A	N/A	8.0
davcore.serverdefaults.sslprotocols	string	Specifies a space-delimited list of the supported SSL protocols as the default for the various back-end services' **sslprotocols** configuration. That is, if the specific **sslprotocols** parameter is not set, it is set to the value of **davcore.serverdefaults.sslprotocols**.	**TLSv1 TLSv1.1 TLSv1.2**	N/A	N/A	8.0.0.1
davcore.serverlimits.httpconnecttimeout	integer	HTTP connection timeout value (in milliseconds), when connecting to another server.	**5000**	**500**	**100000**	8.0
davcore.serverlimits.httpsockettimeout	integer	HTTP Socket timeout value (in milliseconds), when connecting to another server, and waiting for data.	**5000**	**500**	**100000**	8.0
davcore.serverlimits.maxaddressbookcontentlength	long (bytes)	Maximum size of a contacts resource.	**10000000**	**0**	**Maximum long value**	8.0
davcore.serverlimits.maxcontentlength	long (bytes)	Maximum size of a resource. Might be overwritten for certain types of content (for example text/vcard).	**10000000**	**0**	**Maximum long value**	8.0
davcore.serverlimits.maxgroupexpansion	integer	Maximum nested level of group expansion.	**3**	**0**	**-1**	8.0
davcore.serverlimits.maxhttpredirects	integer	Maximum number of HTTP redirects to follow, when connecting to another server.	**3**	**0**	**10**	8.0
davcore.serverlimits.maxmigrationthreads	integer	Maximum number of threads to create when running migration.	**2**	**1**	**20**	8.0
davcore.serverlimits.maxnumberofresourcesincollection	long (bytes)	Maximum number of resources allowed in a collection. A value of **-1** means no limit.	**10000**	**-1**	**Maximum long value**	8.0
davcore.serverlimits.maxresults	integer	Maximum number of resources returned by a single fetch operation. A value of **0** means no limit. Administrator users are not affected by this limit.	**10000**	**0**	**Maximum int value**	8.0
davcore.serverlimits.maxsearchtimerange	long (days)	Maximum bounded search range in days.	**3660**	**0**	**366000**	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
davcore.serverlimits.maxuploadcontentlength	long (bytes)	Maximum size when uploading data. This affects operations that let you create multiple resources in one request (for example import). It does not affect regular PUT.	**20000000**	**0**	**Maximum long value**	8.0
davcore.serverlimits.migrationtimeout	integer	Maximum number of hours to wait before terminating a migration.	**8**	**1**	**100**	8.0
davcore.serverlimits.minsearchcharacters	integer	Minimum number of characters allowed in a text filter search.	**3**	**0**	**256**	8.0
davcore.serverlimits.templockretry	integer	Maximum number of attempts to acquire a temporary lock when doing write operations.	**20**	**1**	**Maximum int value**	8.0
davcore.serverlimits.templocktimeout	integer (seconds)	Maximum amount of time to wait for a temporary lock when doing write operations.	**60**	**1**	**Maximum int value**	8.0
davcore.serverlimits.templockusebackend	boolean	If true, temporary locks are ensured at the back-end level instead of staying local to a server instance.	**false**	N/A	N/A	8.0
davcore.uriinfo.backendidtemplate	string	The backendid template is used to identify the back end hosting the home of a given subject. The template can reference LDAP attributes of the subject, using the **${attrname}** syntax or the **${attrname,defaultvalue}** syntax.	**${nabStore,defaultbackend}**	N/A	N/A	8.0
davcore.uriinfo.defaultdavuriprefix	string	Canonical form of DAV URI prefix for WebDAV based protocols. This prefix corresponds to one of the DavServlet specific path (for example **/dav**) as defined in the**web.xml** file. It should not end with a slash.	**/dav**	N/A	N/A	8.0
davcore.uriinfo.defaulthomeuritemplate	string	Canonical form of a subject home URI prefix. This template should have a corresponding uripattern. It should not end with a slash The template can reference LDAP attributes of the subject, using the **${attrname}** syntax or the **${attrname,defaultvalue}** syntax. The **${domain}** syntax can be used to reference the domain of the subject.	**/home/${mail}**	N/A	N/A	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
davcore.uriinfo.default principaluritemplate	string	Canonical form of a subject principal URI prefix. This template should have a corresponding uripattern. It should not end with a slash. The template can reference LDAP attributes of the subject, using the **${attrname}** syntax or the **${attrname,defaultvalue}** syntax. The **${domain}** syntax can be used to reference the domain of the subject.	**/principals/${mail}**	N/A	N/A	8.0
davcore.uriinfo.defaultr esturiprefix	string	Canonical form of REST URI prefix for WebDAV based protocols. This prefix corresponds to one of the RESTfulServlet specific path as defined in the **web.xml** file. It should not end with a slash.	**/rest**	N/A	N/A	8.0
davcore.uriinfo.director yrootcollection	string	Defines the root collection of all directory collections (without any prefix).	**/directory/**	N/A	N/A	8.0
davcore.uriinfo.emailse archfiltertemplate	string	LDAP Filter used when searching a subject by email address. The `%s` token is replaced by the email value to search.	**|(mail=%s)(mailalter nateaddress=%s)**	N/A	N/A	8.0
davcore.uriinfo.fulluriprefix	string	Full URL prefix to use wherever a full URL is required. It should not end with a slash. This prefix is used to construct attachment URLs embedded in resources. Modifying this parameter does not change full URLs in already existing resources. If SSL is used, the host name part of this prefix should match the host name associated with the certificate.	**http://localhost**	N/A	N/A	8.0
davcore.uriinfo.ldapcac hesize	integer	Maximum number of subjects (LDAP users, resources, and groups) kept in cache when mapping URIs and subjects. Entries are removed from the cache only when this maximum is reached or when any of the uriinfo configuration parameter is changed. Can be set to **0**, indicating no cache.	**1000**	**0**	**Maximum int value**	8.0
davcore.uriinfo.ldapcac hettl	integer (seconds)	Maximum time (in seconds) that subjects (LDAP users, resources, and groups) are kept in cache when mapping URIs and subjects.	**60**	**1**	**Maximum int value**	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
davcore.uriinfo.permanentuniqueid	string	Name of an LDAP attribute present in the LDAP entry of all subjects (users, groups, resources, and so on) and defining a permanent and unique identifier for each subject. The attribute value is used internally to do the mapping between the subject LDAP entry and its repository. As such, it should remain constant for the lifetime of the subject LDAP entry and it should be unique (at least within the subject domain). Changing this configuration parameter results in data loss when the user repositories have been created.	**davuniqueid**	N/A	N/A	8.0
davcore.uriinfo.principalsrootcollection	string	Defines the root collection of all principals in their canonical form. (without any prefix). This parameter is used to return the WebDAV DAV:principal-collection-set property.	**/principals/**	N/A	N/A	8.0
davcore.uriinfo.subjectattributes	string	Space separated list of LDAP attribute names to retrieve when doing a search for users, groups, or resources.	**cn nabstore nabstatus mail mailalternateaddress davuniqueid owner preferredlanguage uid objectclass ismemberof uniquemember memberurl mgrprfc822mailmember kind**	N/A	N/A	8.0
davcore.uriinfo.subjectsearchfilter	string	LDAP Filter used when a user is searching for other users. The %s token is replaced by the search string.	**(|(uid=%s*)(cn=*%s*)(mail=*%s*))**	N/A	N/A	8.0
davcore.uriinfo.subjectsearchfilterminimum	integer	The minimum number of characters allowed for the search string.	**3**	**-2147483648**	**Maximum int value**	8.0
davcore.uriinfo.uricachesize	integer	Maximum number of resolved URIs kept in cache. Entries are removed from the cache only when this maximum is reached or when any of the uriinfo configuration parameter is changed. Can be set to **0**, indicating no cache.	**10000**	**0**	**Maximum int value**	8.0
davcore.uriinfo.uricachettl	integer (seconds)	Maximum time (in seconds) that resolved URIs are kept in cache.	**60**	**1**	**Maximum int value**	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
davcore.uriinfo.useldapproxyauth	boolean	If **true**, use proxy authorization for any LDAP search on behalf of a user. If **false**, use administrator credentials for all LDAP searches.	**true**	N/A	N/A	8.0
davcore.virusscan.auth	boolean	Determines if the virus scan connection should use user and password authorization.	**false**	N/A	N/A	8.0
davcore.virusscan.clivirusaction	string	Action to be performed when a virus is detected during command-line operation. Value is empty or delete.	N/A	N/A	N/A	8.0
davcore.virusscan.debug	boolean	Determines if the virus scan SMTP connection should use debug.	**false**	N/A	N/A	8.0
davcore.virusscan.emailaddress	string	Sets the email recipient address that the MTA uses to trigger a custom virus scan. (Requires MTA configuration).	N/A	N/A	N/A	8.0
davcore.virusscan.host	string	Host of the MTA configured to accept virus scans.	N/A	N/A	N/A	8.0
davcore.virusscan.onlineenable	boolean	Enable and disable online virus scan.	**false**	N/A	N/A	8.0
davcore.virusscan.onlinefailureaction	string	Action to be performed when virus service fails during an online submission. Value is empty or reject.	N/A	N/A	N/A	8.0
davcore.virusscan.onlinevirusaction	string	Action to be performed when a virus is detected during an online submission. Value is empty or reject.	N/A	N/A	N/A	8.0
davcore.virusscan.pass	password	The SMTP authorization password for the SMTP virus scan connection.	N/A	N/A	N/A	8.0
davcore.virusscan.port	string	Port of the MTA host that is configured to accept virus scans.	**25**	N/A	N/A	8.0
davcore.virusscan.starttls	boolean	Determines if the virus scan connection should use starttls.	**false**	N/A	N/A	8.0
davcore.virusscan.timeout	string	Timeout value (in milliseconds) for the connection to the MTA host during a virus scan operation.	**10000**	N/A	N/A	8.0
davcore.virusscan.user	string	The SMTP authorization user for the SMTP virus scan connection.	N/A	N/A	N/A	8.0
davcore.virusscan.usessl	boolean	Determines if the virus scan connection should use SSL.	**false**	N/A	N/A	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
log.dav.commands.logdateformat	logdateformat	Specifies the date format pattern for the log.	**yyyy-MM-dd'T'HH:mm:ss.SSSZ**	N/A	N/A	8.0
log.dav.commands.logdir	filepath	Directory path for log files.	**logs**	N/A	N/A	8.0
log.dav.commands.loglevel	loglevel	Specifies the log level. Valid levels are OFF (no information is logged), SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, and ALL (all information is logged). The FINEST and ALL levels produce a large amount of data.	**INFO**	N/A	N/A	8.0
log.dav.commands.logtoparent	boolean	Flag to enable logging to the application server log file, in addition to the Contacts Server logs.	**false**	N/A	N/A	8.0
log.dav.commands.maxlogfiles	integer	Maximum number of log files.	**10**	**1**	**100**	8.0
log.dav.commands.maxlogfilesize	integer (bytes)	Maximum size of each log file.	**2097152**	**2097152**	**Maximum int value**	8.0
log.dav.errors.logdateformat	logdateformat	Specifies the date format pattern for the log.	**yyyy-MM-dd'T'HH:mm:ss.SSSZ**	N/A	N/A	8.0
log.dav.errors.logdir	filepath	Directory path for log files.	**logs**	N/A	N/A	8.0
log.dav.errors.loglevel	loglevel	Specify the log level. Valid levels are OFF (no information is logged), SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, and ALL (all information is logged). The FINEST and ALL levels produce a large amount of data.	**INFO**	N/A	N/A	8.0
log.dav.errors.logtoparent	boolean	Flag to enable logging to the application server log file, in addition to the Contacts Server logs.	**false**	N/A	N/A	8.0
log.dav.errors.maxlogfiles	integer	Maximum number of log files.	**10**	**1**	**100**	8.0
log.dav.errors.maxlogfilesize	integer (bytes)	Maximum size of each log file.	**2097152**	**2097152**	**Maximum int value**	8.0
log.dav.scan.logdateformat	logdateformat	Specifies the date format pattern for the log.	**yyyy-MM-dd'T'HH:mm:ss.SSSZ**	N/A	N/A	8.0
log.dav.scan.logdir	filepath	Directory path for log files.	**logs**	N/A	N/A	8.0
log.dav.scan.loglevel	loglevel	Specifies the log level. Valid levels are OFF (no information is logged), SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, and ALL (all information is logged). The FINEST and ALL levels produce a large amount of data.	**INFO**	N/A	N/A	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
log.dav.scan.logtoparent	boolean	Flag to enable logging to the application server log file, in addition to the Contacts Server logs.	**false**	N/A	N/A	8.0
log.dav.scan.maxlogfiles	integer	Maximum number of log files	**10**	**1**	**100**	8.0
log.dav.scan.maxlogfilesize	integer (bytes)	Maximum size of each log file	**2097152**	**2097152**	**Maximum int value**	8.0
log.dav.telemetry.logdateformat	logdateformat	Specifies the date format pattern for the log.	**yyyy-MM-dd'T'HH:mm:ss.SSSZ**	N/A	N/A	8.0
log.dav.telemetry.logdir	filepath	Directory path for log files.	**logs**	N/A	N/A	8.0
log.dav.telemetry.loglevel	loglevel	Specifies the log level. Valid levels are OFF (no information is logged), SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, and ALL (all information is logged). The FINEST and ALL levels produce a large amount of data.	**INFO**	N/A	N/A	8.0
log.dav.telemetry.logtoparent	boolean	Flag to enable logging to the application server log file, in addition to the Contacts Server logs.	**false**	N/A	N/A	8.0
log.dav.telemetry.maxlogfiles	integer	Maximum number of log files.	**10**	**1**	**100**	8.0
log.dav.telemetry.maxlogfilesize	integer (bytes)	Maximum size of each log file.	**2097152**	**2097152**	**Maximum int value**	8.0
notification.dav.configdir	filepath	Directory path for notification configuration files or format files	**config/templates**	N/A	N/A	8.0
notification.dav.dateformat	dateformat	Specifies the date format pattern for notification. For example, **EEE MMMMM dd, yyyy**.	**EEE MMMMM dd, yyyy**	N/A	N/A	8.0
notification.dav.enableemailnotif	boolean	Enables server-wide email notification	**true**	N/A	N/A	8.0
notification.dav.enablejmsnotif	boolean	Enables server-wide JMS notification	**true**	N/A	N/A	8.0
notification.dav.maxpayload	integer	Maximum payload size in bytes.	**10000000**	**-2147483648**	**Maximum int value**	8.0
notification.dav.smtpauth	string	SMTP-AUTH access control mechanism flag.	**false**	N/A	N/A	8.0
notification.dav.smtpdebug	string	Specifies SMTP debug flag.	**false**	N/A	N/A	8.0
notification.dav.smtphost	string	Specifies SMTP host.	N/A	N/A	N/A	8.0
notification.dav.smtppassword	password	Specifies SMTP password.	N/A	N/A	N/A	8.0
notification.dav.smtpport	string	Specifies SMTP port.	**25**	N/A	N/A	8.0
notification.dav.smtpstarttls	string	Use SMTP starttls flag.	**true**	N/A	N/A	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
notification.dav.smtpuser	string	Specifies SMTP user.	**user**	N/A	N/A	8.0
notification.dav.smtpussl	string	Specifies SMTP to use SSL flag.	**false**	N/A	N/A	8.0
notification.dav.timeformat	timeformat	Specifies the time format pattern for notification. Use 'a' for AM/PM marker. For example, **hh:mm:ss aaa**.	**hh:mm:ss aaa**	N/A	N/A	8.0
notification.dav.timezoneformat	timezoneformat	Specifies the time zone format pattern for notification. Use 'z' for general time zone, or 'Z' for RFC822 time zone.	**z**	N/A	N/A	8.0
service.dav.blacklist	string	List of clients to be denied of service, expressed as a space separated list of regular expressions. Any client whose User-Agent HTTP header contains any of the regex is denied access.	N/A	N/A	N/A	8.0
service.dav.propfinddavheadervalue	string	Value of the HTTP Dav header value to return in all PROPFIND responses.	**1, 3, access-control, addressbook**	N/A	N/A	8.0
service.dav.telemetry.filter	string	Space separated list of request URIs that a particular request should match (start with) to be logged by telemetry. For example: **/dav/home/jsmith/addressbook/ /dav/home/jdoe/addressbook/**	N/A	N/A	N/A	8.0
service.dav.telemetry.forcetelemetry	boolean	Force telemetry for all users. Use with caution, as it causes lots of data to be generated.	**false**	N/A	N/A	8.0
store.corpdir.defaultcorpdirectoryurl	string	Default corporate directory information to use when performing searches. Can be overwritten by domain specific information (**corpDirectoryUrl** LDAP attribute in the domain entry). If no baseDN is provided, the user's domain baseDN for users and group is used. The list of attributes to retrieve is ignored.	**ldap://ugldap/??sub?(objectclass=inetorgperson)**	N/A	N/A	8.0
store.corpdir.enablecorpdir	boolean	Enable or disable corporate directory lookups.	**true**	N/A	N/A	8.0

Table B–3 (Cont.) Contacts Server Configuration Parameters

Parameter	Type	Description	Default value	Minimum Value	Maximum Value	Version
store.corpdir.useldapproxyauth	boolean	If **true**, uses LDAP proxy authorization to issue LDAP searches on behalf of the logged-in user. If **false**, uses the LDAP Pool credentials for all LDAP searches. This parameter applies only to the default corporate directory configuration.	**true**	N/A	N/A	8.0
store.dav.*.attachstorehost	string	Specifies document store host.	N/A	N/A	N/A	8.0
store.dav.*.attachstoreport	integer	Specifies document store port.	**8008**	**-2147483648**	**Maximum int value**	8.0
store.dav.*.backendid	string	Specifies back-end identifier.	-	N/A	N/A	8.0
store.dav.*.dbdir	filepath	Specifies directory path for nabstore.	**data/db**	N/A	N/A	8.0
store.dav.*.jndiname	string	JNDI name pointing to this back end's JDBC DataSource, as defined in the J2EE container (for example, **jdbc/defaultbackend**).	N/A	N/A	N/A	8.0
store.dav.*.purgedelay	long (seconds)	Sets the delay between deletion of a resource and its actual removal (purge) from the back end. Setting this value too low might cause synchronization clients to perform a full resynchronization too often.	**2592000**	**0**	**Maximum long value**	8.0
store.document.password	password	Password to use when authenticating to a remote document store.	N/A	N/A	N/A	8.0
store.document.timeout	integer	The HTTP(S) connection and read timeout value.	**10000**	**-2147483648**	**Maximum int value**	8.0
store.document.usessl	boolean	Use SSL for communications with remote document store.	**false**	N/A	N/A	8.0