Vocera Administration Interface Guide

Version 5.2.2

Notice

Vocera Communications, Inc.
www.vocera.com
tel :: +1 408 882 5100
fax :: +1 408 882 5101

Last modified: 2018-12-10 14:38

Docs_VS_522Rel build 216

Contents

Overview

The Vocera Administration Interface (VAI) is a Java API that enables you to control and administer the Vocera system programmatically. Using VAI, you can perform almost all the Administration Console and User Console functions described in the *Vocera Administration Guide* and the *Vocera User Console Guide.* Familiarize yourself with the Administration Console and the User Console—structure, function, and features—before you start programming with VAI.

VAI Features

Applications built using VAI can run on any machine (including non-Windows platforms) that has network connectivity to the Vocera server. While VAI does not build in GUI capability, it supports efficient data retrieval in the service of GUI applications, and includes search methods and status callbacks that facilitate the development of such applications.

Here are some of the things you can do with VAI:

- Administer the Vocera system
  - Query and update system settings
  - Create, edit, delete, and query Vocera entities (users, groups, etc.)
  - Start and stop the Vocera Server
  - Backup, restore, and empty the Vocera database
- Integrate Vocera with enterprise applications. For example, you could update Vocera groups dynamically using data from a scheduling application.
- Integrate Vocera with backend databases. For example, you could populate the Vocera database from your HR database.
- Create a customized Administration Console. For example, you could enable access to selected features based on user roles.

VAI Limitations

The initial release of VAI has some limitations compared to the console applications. VAI cannot:

- Load data from a CSV file.
- Operate the Data Checker.

However, in both cases, you can obtain the same effect in your VAI application by writing your own code to perform that functionality.

About VAI Documentation

The *Vocera Administration Interface Guide* (this guide) explains how to develop applications using VAI. It describes key VAI features and explains how to perform common programming tasks.

Also, an HTML-based Javadoc reference to the VAI classes are in the `\VAI\docs\javadocs` directory of the **Vocera Developer Kit** CD.

VAI documentation uses a simplified version of Hungarian notation to indicate variable types in parameter declarations. For example, the prefix "i" indicates an integer (as in iResultCode), the prefix "s" indicates a String (as in sUserName), and the prefix "kps" indicates a KeyedPropertySet (as in kpsUser).

Also, when *myVai* appears in the text or a code example, it refers to an instance of the **`VAI`** class created to represent a connection to a Vocera server. See **VAI Example** for an example.

System Requirements

The following table lists the minimum hardware and software requirements for developing applications using VAI.

Table 1: VAI system requirements

Component	Requirement
Vocera libraries	The following libraries from the **`\vocera\server\lib`** directory on the Vocera Voice Server must be available and in the classpath of your development environment: • **`commons-configuration-1.10.jar`** • **`commons-io-2.4.jar`** • **`commons-lang-2.6.jar`** • **`crypto-1.1.2.jar`** • **`server.jar`** • **`slf4j-api-1.7.7.jar`** The classes that implement VAI are stored in **`server.jar`**. The other libraries support related functions, such as encryption and decryption tasks.
Java Compiler	JDK 8.0 (1.8)
Java Virtual Machine	JRE 8.0 (1.8)
(Optional) IDE	Any IDE or editor that can produce text files suitable for the Java compiler.
Hardware	RAM, CPU, and free disk space required by the Java compiler and IDE.
License	A license that supports VAI. You enter a license key when you install the Vocera server. If you are integrating a VAI application with an existing Vocera installation, make sure the Vocera Voice Server has an appropriate license. VAI requires a VMI-enabled license key, but it does not decrement your VMI license.

How to Upgrade an Existing VAI Application

To upgrade existing VAI applications:

1. Copy the required Vocera libraries files from the **`\vocera\server\lib`** directory of the Vocera Voice Server into your development directory.
   See **System Requirements** on page 6 for the complete list of libraries.
2. Optionally, rebuild your application.
   The Vocera 5.2 server.jar file is compiled using Java 8.0 (1.8). However, most VAI applications built with Java 5.0 compilers should run fine in JRE 8.0. Generally, you should not need to rebuild your application unless it uses methods that have changed in the latest version of server.jar or is in some way incompatible with JRE 8.0.
3. Test your application.
   A VAI-enabled license key must be installed on the Vocera server.
4. Deploy your application.
   If your application uses a certificate file for security, remember to deploy the certificate file with the application.

Getting Started With VAI

This section describes how you can get started developing applications with VAI.

VAI Class Hierarchy

In general, VAI class names correspond to data displayed in the Administration Console or the User Console. For example, the `Location` class encapsulates the data displayed in the Locations screen in the Administration Console. There are a few exceptions:

- The `Address` class and the `AddressSet` class correspond to Address Book page in the Administration Console.
- The `Contact` class and the `ContactSet` class correspond to Buddies (more specifically, to outside buddies) managed via the User Console.
- The `Site` class encapsulates data from the Telephony screen and the Sites screen of the Administration Console.

The following list shows the VAI class hierarchy.

- `class java.lang.Object`
  - `class vai.BadgeStatus`
  - `class vai.Entity`
    - `class vai.Address`
    - `class vai.Contact`
    - `class vai.Device`
    - `class vai.Group`
    - `class vai.Location`
    - `class vai.Site`
    - `class vai.User`
  - `class vai.EntitySet`
    - `class vai.AddressSet`
    - `class vai.ContactSet`
    - `class vai.DeviceSet`
    - `class vai.GroupSet`
    - `class vai.LocationSet`
    - `class vai.SiteSet`
    - `class vai.UserSet`
  - `class vai.LicenseInfo`
  - `class vai.PropertySet`
    - `class vai.IndexedPropertySet`
    - `class vai.KeyedPropertySet`
  - `class java.lang.Throwable` (implements `java.io.Serializable`)
    - `class java.lang.Exception`
      - `class vai.VAIException`
  - `class vai.VAI`

Developing VAI Applications

This section outlines the basic steps in developing an application using VAI.

How to Develop a VAI application:

To develop a VAI application:

1. Copy the required Vocera libraries files from the `\vocera\server\lib` directory of the Vocera Voice Server into your development directory.

See System Requirements on page 6 for the complete list of libraries.

2. Optionally copy the following files from the `\VAI\docs` directory of the **Vocera Developer Kit** CD into your development directory:
   - `VAIDevGuide.pdf`, an electronic version of the *Vocera Administration Interface Guide* (this document)
   - The `Javadocs` folder, an HTML-based reference to the VAI classes.
3. Write the code to implement your client application.

   You must have a Java compiler compatible with JDK 8.0 (1.8). You can write and edit code using any IDE or editor that can produce text files suitable for the Java compiler.

   At run time, your application's classpath must include the libraries listed in System Requirements on page 6.
4. Deploy your application.

   After you develop a VAI application, you'll need to package its files so that you or other users can install it. Many development environments include tools for packaging and deploying applications.

   If your application uses a certificate file for security, remember to deploy the certificate file with the application.

Using the Sample Applications

On the **Vocera Developer Kit** CD, Vocera provides sample VAI applications in the `\VAI\samples` directory. Each sample has its own `Readme.txt` file that describes how to build and run the application, as well as any other configuration information.

Before building and running a VAI sample application, make sure you copy the required Vocera libraries files from the `\vocera\server\lib` directory of the Vocera Voice Server into your development directory.

Note: VAI sample applications are sample software provided solely to illustrate the use of the API. Vocera provides the samples AS IS. You are solely responsible for verifying their suitability for any specific purpose or application.

Avoiding Version Mismatch Problems

When you deploy your VAI application, always make sure that the Vocera Server you are connecting to is at the same version or later of the `server.jar` file that you are using in your application. Otherwise, your application may encounter a server mismatch error and fail to connect to the server. If this happens, copy the `server.jar` from the `%vocera_drive%\vocera\server\lib` directory on the Vocera Server into your application's `\lib` folder. Generally, you should not need to revise or rebuild your application unless it uses methods that have changed since the version of `server.jar` on the Vocera Server.

VAI Example

Here's a simple code example that retrieves data from a Vocera server. A discussion of the code follows the listing.

VAI program example

```
import vai.*;
public class VAIDemo {
  public static VAI myVAI = new VAI();
  public static void main(String[] args) {

    try {
      myVAI.open("192.168.1.1", "Administrator", "admin",
                 null);
      KeyedPropertySet kps = myVAI.getSystemProperties();
      System.out.println(kps.toString());
      myVAI.close();
```

```
      } catch (VAIException ve) {
        System.out.println(ve.getMessage());
      }
    }
  }
```

The example instantiates a `VAI` object to represent a connection to a Vocera server. Throughout this documentation, when myVai appears in the text or a code example, it refers to an instance of the `VAI` class.

The `VAI.open()` method opens a connection to the Vocera Server. The method specifies the IP address(es) of the Vocera server computer(s), an administrator's user name, and the corresponding password. For simplicity, this example hard-codes the login credentials. When security is a concern, an application should prompt for login credentials at the beginning of each VAI session.

The example retrieves Vocera system properties into a `KeyedPropertySet` object, which manages property data as a list of key-value pairs, where each key is a string, and each value is either a string or another property set.

After printing the properties, the program closes the VAI connection to the Vocera Server and releases associated resources. As a best practice, call the `close()` method to close VAI connections explicitly.

The following listing shows a portion of the output generated by the sample program (actual values will vary depending on specific Vocera system settings). This listing is a string representation of a VAI `KeyedPropertySet` instance, which encapsulates a collection of key-value pairs. A key is either a string (such as Product Major Version) or another property set (indicated by the keyword Set and delimited by square brackets).

```
Set
[
   Product Major Version    = 4
   Product Minor Version    = 0
   Product Revision         = 0
   Time Last Update         = 1143050438804
   Self Register            = false
   ...
   Locale                   = US
   Mail Info                = Set
   [
      Server Type              = pop3
      Host                     =
      User Name                =
      Encrypted Password       =
      SMTP Host                =
      SMTP User Name           =
      Encrypted SMTP Password  =
      SMTP Authentication      = true
      Mail Check Interval      = 30000
      Default Recipient        =
      Domain Name              =
   ]
 ...
]
```

Figure 1: String representation of a KeyedPropertySet

Working With Entities

The section describes how to work with VAI entities. A VAI *entity* provides object-oriented access to data on the Vocera server. VAI implements classes you can use to work with entities (such as users and groups), one at a time or in sets. For example, in the following code fragment a `UserSet` object stores a list of all users on the Vocera system. A `User` object represents the first user in that list, and enables a call to the `getUserID()` method implemented by the `User` class.

UserSet and User example

```
UserSet uSet = User.getUsers(myVai);
if (uSet.size() > 0) {
  User u = uSet.elementAt(0);
  String sUserID = u.getUserID();
  System.out.println(sUserID);
}
```

`User.getUsers()` is a static method. It returns a set that lists the users in the Vocera database. The myVai parameter represents an instance of the `VAI` class initialized elsewhere. The example gets the set of Vocera users, retrieves the first user in the set, gets the user ID, and then prints the user ID.

Entity Operations

This section describes operations you can perform on VAI entities.

Creating Entities

Every class that extends `Entity` provides a `create()` method that has the following signature:

```
create(VAI vai, KeyedPropertySet ps)
```

The `create()` method for each class returns an instance of that class. The `vai` parameter represents a connection to the Vocera server (typically instantiated by your application class), and the `ps` parameter stores the key-value pairs that define properties for the class instance you are creating. Note that for each entity type there are some *required* properties. For example, the property set required to create a user is different from the property set required to create a location, as shown in the following code listing.

Property sets used to create users and locations

```
try {
  KeyedPropertySet kpsUser = new KeyedPropertySet(myVai);
  kpsUser.putString("User ID", "jruth");
  kpsUser.putString("First Name", "Jorge");
  kpsUser.putString("Last Name", "Ruth");
  kpsUser.putString("Password", "sultan");
  User u = User.create(myVai, kpsUser);

  KeyedPropertySet kpsLoc = new KeyedPropertySet(myVai);
  kpsLoc.putString("Name", "Cafeteria A");
```

```
  Location loc = Location.create(myVai, kpsLoc);
} catch (VAIException ve) {
  System.out.println(ve.getMessage());
}
```

This code assumes that `myVai` is a `VAI` object that has been instantiated and used to open a connection to a Vocera Server. Each entity class is defined by a specific set of required, optional, and default properties. The properties required to create a location are different from those required to create a user.

Querying Entities

Each entity class implements methods for querying property values specific to that class. Many entity classes also provide static methods that return data about the collection of those class instances as they exist in a Vocera database. For example, the following code gets information about the users in a group.

Querying an entity

```
public void printGroupInfo() {
  try {
    Group gGroup = Group.getGroupWithName(myVai,
                                          "Doctors",
                                          "Cupertino");
    EntitySet esMembers = gGroup.getMembers(false);
    Entity e = null;
    String sCurrName = "";
    for (int i = 0; i < esMembers.size(); i++) {
      e = esMembers.entityAt(i);
      switch (e.getType()) {
        case Entity.tyGroup:
          Group g = (Group) e;
          sCurrName = "[Group] " + g.getName();
          break;
        case Entity.tyUser:
          User u = (User) e;
          sCurrName = "[User] " + u.getFirstName() +
                      " " + u.getLastName();
          break;
        default:
          sCurrName = "No name";
          break;
      }
      System.out.println(sCurrName);
    }
  } catch (VAIException ve) {
    System.out.println(ve.getMessage());
  }
}
```

The `getGroupWithName()` method returns a group with a specified name and site. The `getMembers(false)` method call specifies that both direct members and indirect members (that is, users who belong to nested groups) are returned. The example prints the names of the members, whether they are groups or users.

Updating Entities

In contrast to APIs that provide accessor methods in pairs (such as `getName()` and `setName()`), in VAI you update entities by setting values in property sets. Classes that extend `Entity` inherit the `update()` method. The following code updates the description of a location with the string "NICU Nurse Station".

Updating the description of a location

```
try {
  KeyedPropertySet kps = new KeyedPropertySet(myVai);
  kps.putString("Description", "NICU Nurse Station");
  Location loc =
    Location.getLocationWithName(myVai,
                                 "NICU Nurses"
                                 "Cupertino");
  loc.update(myVai, kps);
} catch (VAIException ve) {
  System.out.println(ve.getMessage());
}
```

Use a `KeyedPropertySet` object to store the key-value pairs that you want to update. When you update an entity, the set should contain *only* the properties that you want to update. For example, when you need to change a user's last name, create and submit a property set that contains only one key-value pair: the key "Last Name" and the new value for the user's last name. If any property values contained in the `KeyedPropertySet` are invalid, the `update()` method fails and throws an exception.

Do not fetch a set of all the user's properties, enter a new last name, and then submit the entire set. You might overwrite changes made by someone else, either through a console or another VAI instance, that were made between your fetch and your post.

For more information about using property sets to update entities, see **Working With Properties**.

Deleting Entities

The `Entity` class provides a `delete()` method you can use to delete any entity from the Vocera database. For example, the following code deletes a user.

Deleting a user

```
try {
  User u = User.getUserWithUserID(myVai, "visitor04");
  u.delete();
} catch (VAIException ve) {
  System.out.println(ve.getMessage());
}
```

The `Entity.delete()` method deletes all data pertaining to an entity from the Vocera system. However, the deletion does not occur immediately. To ensure that no call activity is interrupted, the deletion takes effect when the system has no calls or Genie sessions in progress or after the server is restarted.

In contrast, the following code removes a user from a group, but that user's data remains in the Vocera database.

Removing members from a group

```
try {
  User u = User.getUserWithUserID(myVai,
                                  "visitor04");
  Group g = Group.getGroupWithName(myVai,
                                   "Visitors",
                                   "Headquarters");
  g.removeMember(u);
} catch (VAIException ve) {
  System.out.println(ve.getMessage());
}
```

Using Internal Names

Vocera entities do not necessarily have unique names. For example, there may exist several users or address book entries with the same first and last names, even within the same site. Moreover, such "external" names may change as a result of Administration Console edits or VAI calls. For this reason, each entity has a unique identifier, called an *internal name*, that is created automatically by the Vocera server when the entity is created. This internal name is invariant over the lifetime of the entity, and may therefore be used externally, such as in databases, to designate the entity.

Note: You should not expect users of your VAI client to know anything about Vocera internal names. Therefore, you should avoid using internal names in your client's UI.

The following code example shows how the Vocera system creates and uses internal names for users who have the same first and last names.

Getting internal names for entities

```
try {
  KeyedPropertySet kps1 = new KeyedPropertySet(myVai);
  kps1.putString("First Name", "Ted");
  kps1.putString("Last Name", "Doe");
  kps1.putString("User ID", "teddoe1");

  User u1 = User.create(myVai, kps1);
  System.out.println("Internal name for teddoe1: " +
                     u1.getInternalName());
  KeyedPropertySet kps2 = new KeyedPropertySet(myVai);
  kps2.putString("First Name", "Ted");
  kps2.putString("Last Name", "Doe");
  kps2.putString("User ID", "teddoe2");

  User u2 = User.create(myVai, kps2);
  System.out.println("Internal name for teddoe2: " +
                    u2.getInternalName());

  UserSet uSet = User.getUsers(myVai);
  int i = uSet.findFirstMatch("Doe,Ted");
  System.out.println("First match: " +
                      uSet.entityAt(i).getInternalName());

  i = uSet.findLastMatch("Doe,Ted");
  System.out.println("Last match: " +
                     uSet.entityAt(i).getInternalName());
  u1.delete();
  u2.delete();
} catch (VAIException ve) {
  System.out.println(ve.getMessage());
}
```

The `findFirstMatch()` and `findLastMatch()` methods used in the example are also useful for implementing a find-as-you-type feature in a user interface.

The example code prints the following output.

```
Internal name for teddoe1: u-tdoe
Internal name for teddoe2: u-tdoe0
First match: u-tdoe0
Last match: u-tdoe
```

Working with Addresses

Use the `Address` class to work with Address Book entries. The Vocera address book is a convenient way for badge users to contact places and people who are not badge users. For example, if people in your organization frequently need to contact local businesses, you can enter the business names and nicknames in the address book. Then, getting a price quotation from Northwestern Hardware can be as simple as using the badge to say "Call Northwestern."

Addresses are identified by names: one name for a place (for example, "Northwestern"), two names for a person (first and last, for example, "Jane Doe"). Therefore, in addition to the methods for standard entity operations, the `Address` class provides methods for determining the type (`isPlaceName()`), and for working with names (`getPlaceName()`, `getFirstName()`, `getLastName()`). Also, the methods `getAddressWithName()` and `getAddressesWithName()` are overloaded to return an `Address` or an `AddressSet` object, respectively, given various combinations of place names, first and last names, and site names.

The following code example returns a string describing the Address objects defined for a specified site (or for all sites, if a site is not specified).

Getting addresses for a site

```
public String getAddressNames(String sSite) {
  String sCurrSite = "";
  String sResult = "";
  String sCurrLastName  = "No last name";
  String sCurrFirstName  = "No first name";
  try {
    AddressSet asSet = Address.getAddresses(myVai, sSite);
    if (asSet.size() > 0) {
      for (int i = 0; i < asSet.size(); i++) {
        Address addr = asSet.elementAt(i);
        sCurrSite =
          (sSite.equals("") || sSite == null)
          ? addr.getSiteName() : sSite;
      if (addr.isPlaceName()) {
        sCurrLastName = addr.getPlaceName();
        sResult = sResult + "\n" +
                      "[Type] Place " + "\t" +
                      "[Name] " + sCurrLastName + "\t" +
                      "[Site] " + sCurrSite;
      } else {
        sCurrFirstName = addr.getFirstName();
        sCurrLastName = addr.getLastName();
        sResult = sResult + "\n" +
        "[Type] Person " +"\t" +
        "[Last Name] " + sCurrLastName +"\t" +
        "[First Name] " + sCurrFirstName +"\t" +
        "[Site] " + sCurrSite;
      }
      }
    } else {
      sResult = "There are no Addresses in the database.";
    }
  } catch (VAIException ve) {
    sResult = ve.getMessage();
  }
  return sResult;
}
```

The `isPlaceName()` method finds out whether the `Address` object represents a person or a place. Internally, place names are stored in the Last Name field, with First Name field empty. When the `Address` object represents a person, you can call `getLastName()` and `getFirstName()`. When the `Address` object represents a place, call `getPlaceName()` instead. You could also query the Address's property set for the values of the First Name and Last Name properties.

Working with Buddies and Contacts

A Vocera *buddy* is similar to an address book entry, in that it stores contact information. However, an address book entry represents a person or place outside of the Vocera system, while a buddy can be a badge user, a Vocera group, a Vocera address book entry, or a person or place outside the Vocera system. Also, address book entries are defined for entire sites (or the Global site), while buddies are defined for individual badge users. Buddies enable the use of nick names in prompts and voice commands (for example, "Call the Big Kahuna").

There are two types of buddies: inside buddies and outside buddies.

- An *inside* buddy represents another badge user, a group, or an address book entry. Badge users can contact inside buddies the same way they contact anyone with a badge. You can assign each buddy a special ring tone that plays when the badge user receives a call from that buddy. Also, inside buddies can be given VIP (very important person) status, enabling them to contact a badge user even when that badge user is blocking calls or is in Do Not Disturb mode.
- An *outside* buddy is someone who is not already represented in the Vocera database as a badge user, group, or address book entry. A badge user can contact an outside buddy by calling a telephone from a badge, or by sending an email message from a badge to an email account.

Vocera users can create and manage buddy lists as described in the *Vocera User Console Guide*. In VAI, you use the following classes to administer buddies programmatically:

- The `Contact` class represents an outside buddy as a VAI entity. Use the `Contact` class to create and delete outside buddies, specifying basic contact information (such as name, phone number) in a `KeyedPropertySet`.

  Unlike some other entity classes, the `Contact` class has some properties that VAI cannot update. To get a complete list of all `Contact` properties (read/write and read-only), call `getPropertyKeys()`. To get a list of read/write properties (for example, to display in a UI for editing), call `getPropertyKeysForUpdate()`.
- The `User` class provides a means for updating a user's buddy list. Each User object has a Buddies, an indexed set in which each element is itself a keyed set that contains a `Contact` object or a `User` object along with properties including nick name and VIP status.

The following code example creates an outside buddy (contact) and adds it to an existing badge user's buddy list.

Adding a contact

```
public static int addContact() {
    int iResultCode = -1;
    try {
     // Find the existing user.
      User u = User.getUserWithUserID(myVai, "rhall");

      // Properties for a new Contact.
      // A Contact represents an outside buddy.
      KeyedPropertySet kpsNewContact =
         new KeyedPropertySet(myVai);
      kpsNewContact.putString("Last Name", "Davis");
      kpsNewContact.putString("First Name", "Mills");
      kpsNewContact.putString("Desk Phone", "408-555-1234");
      kpsNewContact.putUser("Owner", u);

      Contact cNewContact =
        Contact.create(myVai, kpsNewContact);

      // Properties (including the new Contact) for a new
 Buddy.
      KeyedPropertySet kpsNewBuddy =
         new KeyedPropertySet(myVai);
```

```
      kpsNewBuddy.putContact("Name", cNewContact);
      kpsNewBuddy.putString("Nick Name", "New outside
 buddy");

      // Add the new Buddy to the user's existing set.
      KeyedPropertySet kpsUser = u.getProperties();
      IndexedPropertySet ipsBuddies =
         kpsUser.getIndexedSet("Buddies");
      ipsBuddies.add(kpsNewBuddy);

      // Update the user with new property set.
      KeyedPropertySet kpsUpdate =
         new KeyedPropertySet(myVai);
      kpsUpdate.putSet("Buddies", ipsBuddies);

      u.update(myVai, kpsUpdate);

      iResultCode = 0;
    } catch (VAIException ve) {
       System.out.println(ve.getMessage());
       iResultCode = ve.getResultCode();
    }
    return iResultCode;
  }
```

The following code example adds an existing badge user to another badge user's buddy list.

Adding an inside buddy

```
public static int addInsideBuddy() {
  int iResultCode = -1;
  try {
    // Search for inside buddy.
    User uBuddy = User.getUserWithUserID(myVai, "mdavis");

    // Properties for a new Inside Buddy.
    KeyedPropertySet kpsNewBuddy =
       new KeyedPropertySet(myVai);
    kpsNewBuddy.putUser("Name", uBuddy);
    kpsNewBuddy.putString("Nick Name", "New inside buddy");

    // Search for owner.
    User uOwner = User.getUserWithUserID(myVai, "rhall");

    // Add the new Buddy to the user's existing set.
    KeyedPropertySet kpsUser = uOwner.getProperties();
    IndexedPropertySet ipsBuddies =
       kpsUser.getIndexedSet("Buddies");
    ipsBuddies.add(kpsNewBuddy);

    // Update the user with new property set.
    KeyedPropertySet kpsUpdate =
       new KeyedPropertySet(myVai);
    kpsUpdate.putSet("Buddies", ipsBuddies);

    uOwner.update(myVai, kpsUpdate);

    iResultCode = 0;
  } catch (VAIException ve) {
     System.out.println(ve.getMessage());
     iResultCode = ve.getResultCode();
  }
  return iResultCode;
}
```

Working with Devices

Use the `Device` class to work with Vocera devices, such as badges. You can manage and track the devices that connect to the Vocera system.

Creating a Device

To create a device, use the `Devices.create()` method. The only required property for devices is the MAC Address property, a 12-character string.

Note: Vocera automatically adds new devices to the system when they connect to the server, so you rarely will need to create a device using VAI. Instead, use VAI to update device information.

Creating a device

```
public static void createNewDevice(String sMACAddr){
  try {
    KeyedPropertySet kpsDevice = new
 KeyedPropertySet(myVai);
    kpsDevice.putString("MAC Address", sMACAddr);
    Device d = Device.create(myVai, kpsDevice);
  } catch (Exception ex) {
    System.out.println(ex.getMessage());
  }
}
```

For more information about creating entities, see **Creating Entities**.

Updating a Device

As part of your device management practices, the System Device Manager should input information for each Vocera badge. This information will allow you to manage and track the badges that connect to the Vocera system.

Updating a device

```
//import java.text.*;

public static void updateDevice(Device d){
  try {
    DateFormat df = new SimpleDateFormat("MM/dd/yyyy");
    long lTracking = df.parse("11/30/2013").getTime();
    Site siteSC = Site.getSiteWithName(myVai, "Santa Cruz");
    Group gEDN = Group.getGroupWithName(myVai, "E D Nurse",
      "Global");

    KeyedPropertySet kpsDevice = new
 KeyedPropertySet(myVai);
    kpsDevice.putString("Serial No", "A2AM070505D4");
    kpsDevice.putString("Status", "Active");
    kpsDevice.putGroup("Owning Group",gEDN);
    kpsDevice.putLong("Tracking Time", lTracking);
    kpsDevice.putSite("Site", siteSC);
    kpsDevice.putBoolean("Shared", false);
    d.update(myVai, kpsDevice);
  } catch (Exception ex) {
    System.out.println(ex.getMessage());
  }
}
```

The serial number for a device must be consistent with the device's MAC address. Otherwise, the `update()` method will fail and throw an exception.

For more information about updating entities, see **Updating Entities**.

Getting Devices

The `Device` class provides several methods for getting devices. There are different methods for retrieving a set of devices and for retrieving a single device.

You can use the following methods to return a set of devices. Each of these methods returns a `DeviceSet` object, which provides methods to manipulate the set.

Table 2: Methods for retrieving a set of devices

Method	Description
`getDevices(VAI vai)`	Returns the set of all devices.
`getDevices(VAI vai, java.lang.String sSiteName)`	Returns the set of devices for a given site.
`getDevicesWithLabel(VAI vai, java.lang.String sLabel)`	Returns the set of devices with a specified label.
`getDevicesWithOwner(VAI vai, Group gOwner)`	Returns the set of devices with a specified owning group.
`getDevicesWithOwner(VAI vai, Group gOwner, java.lang.String sSiteName)`	Returns the set of devices with a specified owning group at a particular site.

Table 3: Methods for retrieving an individual device

Method	Description
`getDeviceWithInternalName(VAI vai, java.lang.String sInternalName)`	Returns the device with the specified internal name.
`getDeviceWithMACAddr(VAI vai, java.lang.String sMACAddr)`	Returns the device with the specified MAC address.
`getDeviceWithSerialNo(VAI vai, java.lang.String sSerialNo)`	Returns the device with the specified serial number.

Getting devices

```
public static void getSCDevices(){
  try {
    Device d;
    //Get the devices at the Santa Cruz site
    DeviceSet dsSC = Device.getDevices(myVai, "Santa Cruz");
    for (int i = 0; i < dsSC.size(); i++) {
      d = dsSC.elementAt(i);

      //Print the MAC address of each device at Santa Cruz
      System.out.println("MAC Address: " +
        d.getName());

      //Print the serial number of each device
      System.out.println("Serial No: " +
        d.getSerialNo());
    }
  }catch (Exception ex) {
    System.out.println(ex.getMessage());
  }
}
```

Getting the Color or Type of a Device

Vocera devices are either white or black. All B1000A badges are black. B2000 badges can be black or white. You can use the `Device.getColorFromSerialNo()` method to determine the color of a device based on its serial number.

A Vocera device can be one of three types: B2000, B1000A, or an unknown type. The `Device` class provides integer constants to represent these types. You can use these constants as the parameter value for `getDeviceType(int iType)` to return a localized String representation of the device type.

The following example shows how to get the color and type of a device.

Getting the color and type of a device

```
public static void getColorAndType(Device d){
  try {
    //Get the color of the device
    String sColor = d.getColorFromSerialNo(d.getSerialNo());
    //Print the color
    System.out.println(sColor);

    //Get the device type of the device
    String sType = Device.getDeviceType(d.getDeviceType());
    //Print the device type
    System.out.println(sType);
    }
  }catch (Exception ex) {
    System.out.println(ex.getMessage());
  }
}
```

Modifying Device Status Choices

Vocera provides a list of default status values for devices, such as "Unregistered," "Inventory," and "Active." However, you can define your own status choices based on the device management processes you have implemented.

The following example shows how to add, delete, and rename a device status.

Modifying device status choices

```
public static void modifyStatuses(){
  try {
    //Add a new status called "Assigned"
    Device.addStatusChoice(myVai, "Assigned",
    "Badge has been assigned to a group.");

    //Remove the status "Active" and replace it with
 "Assigned"
    Device.removeStatusChoice(myVai, "Active", "Assigned");

    //Change the name of status "Received for Repair"
    //to "In Repair"
    Device.renameStatusChoice(myVai, "Received for Repair",
  "In Repair", "System Device Manager has received
  the Vocera device for diagnosis and repair.");
  }catch (Exception ex) {
    System.out.println(ex.getMessage());
  }
}
```

Uploading Badge Logs

If you know the MAC address of a badge that is connected to the Vocera Server, you can use the `Device.uploadBadgeLogs()` method to upload the logs from the badge to the Vocera Server for troubleshooting purposes. To get the MAC address for the badge currently associated with a user, you can use the `User.getBadgeStatus()` method. If you already know the MAC address of a badge, you can get the associated user by using the `User.getUserWithMACAddr()` method.

Note: The `uploadBadgeLogs()` method is not supported on B1000A badges and Vocera smartphones.

The badge assembles logs files into a single `.tar.gz` file and uploads the file to the `\vocera \logs\BadgeLogCollector\uploads` directory on the Vocera Server. The format of the filename is *`DATETIME-USERNAME-BADGEMAC`*`-udd.tar.gz`.

The following example shows how to upload badge logs.

Uploading badge logs

```
public static void uploadBadgeLogs(VAI myVai, String userID)
{
  int iResultCode = -1;
  String macAddr = "";
  try
  {
    BadgeStatus[] bsa = User.getBadgeStatus(vai);
    BadgeStatus bsObj = null;
    for (int i = 0; i < bsa.length; i++) {
      bsObj = bsa[i];
      User uTemp = bsObj.u;
      if (uTemp.getUserID().equals(userID)) {
        macAddr = bsObj.sMACAddr;
        break;
      }
    }
    if (!macAddr.equals("")) {
      Device.uploadBadgeLogs(myVai, macAddr);
      if (iResultCode == -1) {
        System.out.println("Badge logs were uploaded.");
      }
    } else {
      System.out.println("Error: " + userID +
      " is not logged into a B2000 or B3000 badge.");
    }
  }
  catch (VAIException e)
  {
    iResultCode = e.getResultCode();
    System.out.println(e.getMessage());
  }
}
```

Working with Groups

Use the `Group` class to work with Vocera groups. Vocera groups organize users into roles such as Floor Manager, Cashier, Nurse, Cardiologist, Executive, and so forth. Groups provide a way to leave messages for many users at once ("Send a message to Nurses Assistants"), or to call someone who fits a specific role ("Call a sales person"), belongs to a certain department ("Call Accounts Receivable"), or has some other skill or authority that the caller requires ("Call a manager"). See the *Vocera Administration Guide* for complete information about groups.

Getting Subgroups

A group can have multiple levels of subgroups contained within it. To work with subgroups, the Group class provides the `getSubgroups()` method. This method takes one parameter, a `boolean` value that specifies whether to return only immediate subgroups or all subgroups, including nested subgroups. For example, the following figure shows the group structure for the I C U department group, which has six subgroups. Given a parameter value of `true`, the `getSubgroups()` method would return a set that did not include the I C U Float Nurse subgroup because it is not a direct member of I C U. Given a parameter value of `false`, the `getSubgroups()` method would return a set containing all six subgroups, including I C U Float Nurse.

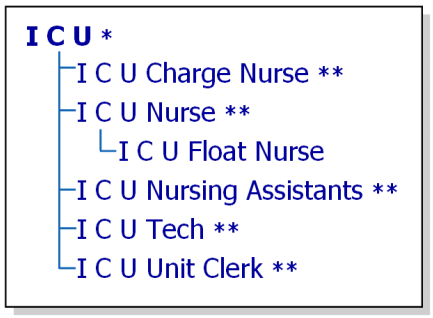

Figure 2: Subgroups

The following code example shows how to get all subgroups for the I C U department group:

Getting subgroups

```
void getICUSubgroups(VAI myVai) {
  try {
    Group gICU = Group.getGroupWithName(myVai, "I C U",
 "Global");
    GroupSet gsICUSubs = gICU.getSubgroups(true);
    for (int i=0; i < gsICUSubs.size(); i++) {
      System.out.println(gsBICUSubs.elementAt(i).getName());
    }
  }catch (VAIException ve) {
    System.out.println(ve.getMessage());
}
```

Managing Group Membership

Group membership can change over time, and in some environments it can change frequently. A user can be a member of multiple groups at the same time. An administrator can add or remove group members either with voice commands or through the Administration Console. Users can remove themselves from groups, and with the proper permission, they can add themselves or other users to groups.

The Group class provides the `getMembers()` method. This method takes one parameter, a `boolean` value that specifies whether to return only direct group members or direct members and members of nested groups. For example, the following figure shows a group structure where the Employees group contains six members: four direct members and two members in a nested group named Managers. Given a parameter value of `true`, the `getMembers()` method would return a set containing the four direct members. Given a parameter value of `false`, the `getMembers()` method would return a set containing all six members.

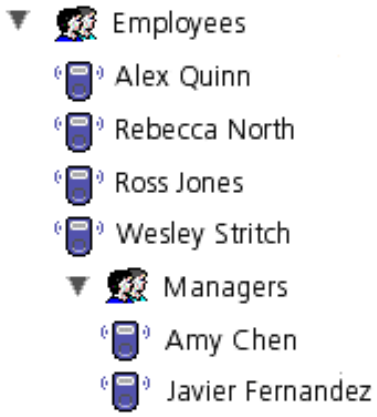

Figure 3: Nested groups

The following code example shows how VAI can manage group membership.

Managing groups

```
void manageGroups(VAI myVai) {

  try {

    User u;
    UserSet us = new UserSet(myVai);

    // Get the Managers group in the Global site
    Group gMgr =
 Group.getGroupWithName(myVai,"Managers","Global");

    // Get a User with the ID "jfernandez"
    u = User.getUserWithUserID(myVai, "jfernandez");

    // Add jfernandez to the user set
    us.add(u);

    // Remove jfernandez from the Managers group
    gMgr.removeMember(u);

    // Get another User with the ID "hwang"
    u = User.getUserWithUserID(myVai, "hwang");

    // Add hwang to the user set
    us.add(u);

    // Add hwang to the Managers group
    gMgr.addMember(u);

    // Create a new keyed property set for a group
    KeyedPropertySet kpsG = new KeyedPropertySet(myVai);
    kpsG.putString("Name","Technicians");
    kpsG.putString("Spoken Member Name", "a technician");

    // Create a new group called Technicians
    Group gTech = Group.create(myVai, kpsG);

    // Add members from the user set to the group
    for (int i = 0; i < us.size(); i++) {
      gTech.addMember(us.elementAt(i));
    }

  } catch (VAIException ve) {
    System.out.println(ve.getMessage());
  }
```

```
}
```

When you remove a member from a group, you do not delete the user from the database.

Note: The `Group.updateMembers(EntitySet esMembers)` method simplifies updates to group membership. This sets the members of a group in one operation, which is more efficient than making repeated calls to `addMember()` and `removeMember()`. All existing members of the group are removed, and members in `esMembers` are added in order. For more information about the `updateMembers()` method, see the Javadoc for the `Group` class.

Managing Group Permissions

When you create or modify a group, you specify values for properties that control the way the group behaves and the way users interact with it. Among these properties are permissions, such as Call Toll Numbers, Initiate Urgent Broadcasts, and Erase Voiceprint of Another User.

A Group object's properties include two sets: Permissions and AntiPermissions. The Permissions set specifies the permissions that are granted to members of that group, while the AntiPermissions set specifies permissions that are revoked for those members even if they belong to another group that confers the permissions. Both of these sets have the same keys, and all the values are of type `boolean`. To explicitly grant a permission, set the corresponding property in the Permissions set to `true`. To explicitly revoke a permission, set the corresponding property in the AntiPermissions set to `true`, as in, "Yes, it's true. I really want to revoke this permission." A permission cannot be both granted and revoked for the same group simultaneously. Therefore, a permission that has been revoked automatically overrides the granting of that same permission.

The complete set of permissions available to any single user is the total list of permissions granted to all the groups of which he or she is a member. Therefore, setting a property value to `false` in the Permissions set does not necessarily deny that permission to a user, nor does setting a property value to `false` necessarily grant that permission. The user may belong to other groups for which the property has been explicitly granted or revoked. For more information about working with permissions, see *Vocera Administration Guide*.

The following code example shows how VAI can explicitly grant and revoke permissions for a group.

Granting and revoking permissions for a group

```
public static int demoPermissions() {
  int iResultCode = -1;
  try {
    Group g =
      Group.getGroupWithName(myVai, "Doctors", "Global");

System.out.println("Before");
KeyedPropertySet kpsG = g.getProperties();
KeyedPropertySet kpsP = kpsG.getKeyedSet("Permissions");
KeyedPropertySet kpsAP =
 kpsG.getKeyedSet("AntiPermissions");
System.out.println(kpsP.toString());
System.out.println(kpsAP.toString());

    KeyedPropertySet kpsPerms =
      new KeyedPropertySet(myVai);
    KeyedPropertySet kpsAntiPerms =
      new KeyedPropertySet(myVai);

    //Explicitly grant these permissions
    kpsPerms.putBoolean("Call Toll-Free Numbers", true);
    kpsPerms.putBoolean("Call Toll Numbers", false);

    //Explicitly revoke these permissions
    kpsAntiPerms.putBoolean("Erase your Voiceprint", true);
```

```
    kpsAntiPerms.putBoolean("Record your Voiceprint",
 false);

    KeyedPropertySet kpsGroup =
      new KeyedPropertySet(myVai);
    kpsGroup.putSet("Permissions", kpsPerms);
    kpsGroup.putSet("AntiPermissions", kpsAntiPerms);

    g.update(myVai, kpsGroup);
    iResultCode = 0;

System.out.println("After");
kpsG = g.getProperties();
kpsP = kpsG.getKeyedSet("Permissions");
kpsAP = kpsG.getKeyedSet("AntiPermissions");
System.out.println(kpsP.toString());
System.out.println(kpsAP.toString());

    } catch (VAIException ve) {
      iResultCode = ve.getResultCode();
      System.out.println(ve.getMessage());
    }
  return iResultCode;
}
```

Working with Locations

Use the `Location` class to work with Vocera locations. Locations are names of places to which you assign one or more access points. When a badge connects to an access point, the Vocera server is able to report the corresponding location. The location names also appear in the Badge Status Monitor, replacing the MAC address of the access point.

Locations are identified by names. Therefore, in addition to the methods for standard entity operations, the `Location` class provides methods that return a `Location` or a `LocationSet` object, given various combinations of location names, site names, and internal names.

The following code example returns information about the Location objects defined for a specified site (or for all sites, if a site is not specified).

Getting location information

```
public String getLocationInfo(String sSite) {
  String sCurrSite = "";
  String sResult = "";
  String sCurrName = "No name";
  String sCurrDesc = "";
  KeyedPropertySet kpsLoc = new KeyedPropertySet(myVai);
  try {
    LocationSet lsSet = Location.getLocations(myVai, sSite);
    if (lsSet.size() > 0) {
      for (int i = 0; i < lsSet.size(); i++) {
        Location loc = lsSet.elementAt(i);
        sCurrName = loc.getName();
        sCurrSite =
          (sSite.equals("") || sSite == null)
          ? loc.getSiteName() : sSite;
        kpsLoc = loc.getProperties();
        sCurrDesc = kpsLoc.getString("Description");
        sResult = sResult + "\n" +
        "[Name] " + sCurrName +"\t" +
        "[Site] " + sCurrSite +"\t" +
        "[Description] " + sCurrDesc;
      }
    } else {
      sResult = "There are no Locations in the database.";
    }
  }catch (VAIException ve) {
    sResult = ve.getMessage();
```

```
  }
  return sResult;
}
```

The following code example shows how to create a location and set its properties, including properties for Access Points and Neighbors.

Creating a location

```
private void createLocation() {

  try {

    Location l;

    // Create a new keyed property set for a location
    KeyedPropertySet kpsL = new KeyedPropertySet(myVai);
    kpsL.putString("Name","Lab");
    kpsL.putString("Description", "Laboratory");
    kpsL.putSite("Site", "s-global");
    kpsL.putString("Spoken Name", "laboratory");

    // Create an indexed property set for access points
    IndexedPropertySet ipsAP = new
 IndexedPropertySet(myVai);
    ipsAP.add("00008A886356");
    ipsAP.add("00004A556454");
    kpsL.putSet("Access Points", ipsAP);

    // Create an indexed property set for neighbors
    IndexedPropertySet ipsN = new IndexedPropertySet(myVai);
    l = Location.getLocationWithName(myVai, "Oakmont",
 "Global");
    ipsN.add(l);
    l = Location.getLocationWithName(myVai, "Spyglass",
 "Global");
    ipsN.add(l);
    kpsL.putSet("Neighbors", ipsN);

    // Create the location
    Location lLab = Location.create(myVai, kpsL);

  } catch (VAIException ve) {
    System.out.println(ve.getMessage());
  }
}
```

Working with Sites

Use the `Site` class to work with Vocera sites. In Vocera, a site is a distinct physical location that shares a centralized Vocera server with one or more other physical locations. Site profiles associate users and groups with specific physical locations.

Sites are identified by names. Therefore, in addition to the methods for standard entity operations, the `Site` class provides methods that return a `Site` or a `SiteSet` object, given various combinations of site names and internal names.

Unlike some other entity classes, the `Site` class has some properties that VAI cannot update. To get a complete list of all `Site` properties (read/write and read-only), call `getPropertyKeys()`. To get a list of read/write properties (for example, to display in a UI for editing), call `getPropertyKeysForUpdate()`.

The `Site` class also provides methods for moving entities between sites. The `moveEntitiesToSite()` method is overloaded, enabling you to move all entities from one site to another, or to specify a set of entities to move. The following code example moves a group and all of its members to another site.

Moving a group to another site

```
public int transferGroup(String sGroupName,
                                String sFromSite,
                                String sToSite)
{
  int iResultCode = -1;
  try {
    Group g =
      Group.getGroupWithName(myVai, sGroupName, sFromSite);
    EntitySet es = g.getMembers(false);
    es.add(g);
    Site siFrom = Site.getSiteWithName(myVai, sFromSite);
    siFrom.moveEntitiesToSite(es, sToSite);
    iResultCode = 0;
  } catch (VAIException ve) {
    iResultCode = ve.getResultCode();
  }
  return iResultCode;
}
```

Many `Site` object properties configure telephony properties for a site, and so enable programmatic access to the features of the Telephony screen in the Administration Console. As in the Administration Console, many telephony properties cannot be modified through VAI unless telephony is enabled. For example, suppose that you want the Telephony server to omit the area code from the dial string when placing a local call. Using the Administration Console, you would perform the following steps.

1. Display the Basic Info page of the Telephony screen.
2. Verify that Enable Telephony Integration is selected (checked).
3. Display the Access Codes page of the Telephony screen.
4. Verify that Omit Area Code when Dialing Locally is selected (checked).
5. Save changes, if necessary.

The following code example shows the corresponding steps in VAI.

Omitting the area code from the dial string

```
public static int omitAreaCode(String sSiteName) {
  int iResultCode = -1;
  try {
    Site si = Site.getSiteWithName(myVai, sSiteName);
    KeyedPropertySet kpsSite = si.getProperties();
    KeyedPropertySet kpsTelInfo =
      kpsSite.getKeyedSet("Telephony Info");
    boolean bTelEnabled =
      kpsTelInfo.getBoolean("Telephony Enabled");
    KeyedPropertySet kpsNewTelInfo =
      new KeyedPropertySet(myVai);

    if (bTelEnabled == false) {
      kpsNewTelInfo.putBoolean("Telephony Enabled", true);
      KeyedPropertySet kpsUpdate = new
 KeyedPropertySet(myVai);
      kpsUpdate.putSet("Telephony Info", kpsNewTelInfo);
      si.update(myVai, kpsUpdate);
    }

    kpsNewTelInfo.putBoolean("Seven Digit Dialing", true);
```

```
    KeyedPropertySet kpsUpdate = new
 KeyedPropertySet(myVai);
    kpsUpdate.putSet("Telephony Info", kpsNewTelInfo);
    si.update(myVai, kpsUpdate);

    iResultCode = 0;
  } catch (VAIException ve) {
    iResultCode = ve.getResultCode();
  }
  return iResultCode;
}
```

Working with Users

Adding new users to the system and updating information for existing users are two primary tasks of a Vocera system administrator. When you add a user (or when a user self-registers), the Vocera system creates a profile for that user in the Vocera server database. Use the `User` class to work with Vocera user profiles.

After a user has had some time to work with the badge, you may need to edit the user's profile to add features that may be useful or remove features that the user does not want. In addition to a user's name and contact information, the profile stores user preferences, such as which Genie persona will prompt the user, whether warning tones are played when the badge has a low battery, or when the user has a new voice or text message.

Unlike some other entity classes, the `User` class has some properties that VAI cannot update. To get a list of `User` properties that you can read, call `getPropertyKeys()`. To get a list of properties that you can modify (for example, to display in a UI for editing), call `getPropertyKeysForUpdate()`.

Note: For security reasons, password properties cannot be read but they can be updated.

Identifying Users

Users are identified by names. Therefore, in addition to the methods for standard entity operations, the `User` class provides methods for working with names (`getFirstName()`, `getLastName()`). Also, the methods `getUserWithName()` and `getUsersWithName()` are overloaded to return a `User` or a `UserSet` object, respectively, given various combinations of first names, last names, and site names.

Because it's not uncommon for two or more users to have the same first and last names, the `User` class provides the following methods for distinguishing between users:

Table 4: Methods for distinguishing between users

Method	Description
`getUserWithMACAddr(VAI vai, java.lang.String sMACAddr)`	Returns user with a given MAC address, or null if no such user exists.
`getUserWithUserID(VAI vai, java.lang.String sUserID)`	Returns user with a given User ID, or null if no such user exists.
`getUserWithInternalName(VAI vai, java.lang.String sInternalName)`	Returns user with a given internal name.

Users and Group Membership

To find out which groups a user belongs to, call the `getContainingGroups()` method. This method takes a boolean argument that specifies whether to return only those groups of which the user is an immediate (direct) member, or to return all the groups of which the user is a member.

For example, suppose that the user Jane Doe is a member of the Charge Nurses group, and the Charge Nurses group is a member (subgroup) of the Nurses group. A call to `getContainingGroups(true)` would return only the Charge Nurses group, while a call to `getContainingGroups(false)` would return both the Charge Nurses group and the Nurses group.

The following code example prints the names of containing groups for a user specified by user ID.

Getting groups for a user

```
public int getGroupsForUser(String sUserID,
                            boolean bImmediate) {
  int iResultCode = -1;
  try {
    User u = User.getUserWithUserID(myVai, sUserID);
    GroupSet gs = u.getContainingGroups(bImmediate);
    Group gTemp = null;
    for (int i = 0; i < gs.size(); i++) {
      gTemp = gs.elementAt(i);
      System.out.println(gTemp.getName());
    }
    iResultCode = 0;
  } catch (VAIException ve) {
    iResultCode = ve.getResultCode();
  }
  return iResultCode;
}
```

When the value of the bImmediate parameter of `getContainingGroups()` is true, the method retruns only those groups in which the user is a direct member. Otherwise, it returns all the groups in which the user is a member.

Badge Users and Badge Status

The `User` class also provides `getBadgeStatus()`, a static method that returns status information about *every user* who is currently logged in and online. This method returns an array of `BadgeStatus` objects, where each `BadgeStatus` object has the public fields defined in the following table.

Note: The `getBadgeStatus()` method returns an array that contains a `BadgeStatus` object for *every user* who is currently logged in and online. Therefore, when a large number of users are online, the resulting array is large, too.

Table 5: BadgeStatus fields

Name	Type	Description
u	User	Represents a badge user.
sIPAddr	String	Dotted form of the user's IP address.
sLocation	String	Name of the user's current location.
bDND	boolean	True if the user is in Do Not Disturb mode.
bHold	boolean	True if the user is has a call on hold.
sCallState	String	Current call state. One of: Inactive, Call, Genie, Conference.
siLocalSite	Site	Represents the user's current site.
sMACAddr	String	MAC address of the device.

The following code example uses the `getBadgeStatus()` method and a `BadgeStatus` object to get the name of the location with which a specified badge user is associated, if that user is online.

Getting the current location for a badge

```
public String getUserLocation(String sUserID) {
  String sLoc = "";
  try {
    BadgeStatus[] bsa = User.getBadgeStatus(myVai);
    BadgeStatus bsObj = null;
    for (int i = 0; i < bsa.length; i++) {
      bsObj = bsa[i];
      User uTemp = bsObj.u;
      if (uTemp.getUserID().equalsIgnoreCase(sUserID)) {
        sLoc = bsObj.sLocation;
        break;
      } else {
        sLoc = sUserID + " is not online.";
      }
    }
  } catch (VAIException ve) {
    sLoc = ve.getMessage();
  }
  return sLoc;
}
```

Importing User Data

This section describes one way to import user data from an external database into the Vocera database. Several aspects of this sample have been simplified for clarity. For example, table structures and relationships are likely to be more complex in a production environment. Similarly, the Java code omits details such as error checking.

The following code example queries an external MySQL database using JDBC, then uses the results to create users in the Vocera database.

Importing users from an external database

```
import java.sql.*;
import vai.*;

public class DbDemo {
  public static VAI myVai = new VAI();

  private class MyVaiListener implements VAIListener {
    public void handleServerStateChange(int iState) {
      System.out.println("Handling Vocera server state
 change: "
        + VAI.getServerStateString(iState));
  }

    public void handleReportStatus(String sTitle,
                                   int iStatus,
                                   String sStatus,
                                   int iPercentDone,
                                   String sError)
    {
      System.out.println("Reporting a server status
 change.");
    }
  } // MyVaiListener

  public MyVaiListener myL;

  public DbDemo() {
    myL = new MyVaiListener();
  }

  // Database connnection parameters.
  // Replace them with values for your database.
  static String sHost = "host";
  static String sDbName = "database";
  static String sUsername = "user";
  static String sPassword = "password";
```

```
  // Build a JDBC connection string.
  // Values shown are for MySQL.
  static String sThinConn = "jdbc:mysql://" +
                            sHost + "/" + sDbName;

  // Edit this value for a database other than MySQL.
  static String sDriverName = "com.mysql.jdbc.Driver";

  public static void main(String[] args) {
    DbDemo demo = new DbDemo();

    try {
      // Replace vs_name with your Vocera server host name.
      myVai.open("vs_name",
                 "Administrator",
                 "admin",
                 demo.myL);

      // Connect to the database.
      Class.forName(sDriverName).newInstance();
      Connection conn =
                 DriverManager.getConnection(sThinConn,
                                             sUsername,
                                             sPassword);
      // Define and execute a query.
      Statement stmt = conn.createStatement();
      String query =
       "select e.UserID, e.FirstName, e.LastName, " +
              "d.CostCenter " +
       "from EMP e, DEPT d " +
       "where e.DeptID = d.ID";

      ResultSet rs = stmt.executeQuery(query);

      // Use query results to create Vocera users.
      KeyedPropertySet kpsUser = null;
      User u = null;
      while (rs.next()) {
        kpsUser = new KeyedPropertySet(myVai);
        kpsUser.putString("User ID",
                          rs.getString("UserID"));
        kpsUser.putString("First Name",
                          rs.getString("FirstName"));
        kpsUser.putString("Last Name",
                          rs.getString("LastName"));
        kpsUser.putString("Cost Center",
                          rs.getString("CostCenter"));
        // Assign a default password.
        kpsUser.putString("Password", "vocera");

        u = User.create(myVai, kpsUser);
        System.out.println("Created user: " +
                           rs.getString("UserID"));
      }
      // Close database connection.
      conn.close();

    } catch (VAIException ve) {
      System.out.println("VAI Exception: " +
 ve.getMessage());
    } catch (Exception e) {
      e.printStackTrace(System.out);
    }
    // Close VAI connection.
    myVai.close();
  }
}
```

The sample external database contains tables named DEPT and EMP created by the following SQL code.

SQL code for an external database

```
DROP TABLE IF EXISTS `DEPT`;
CREATE TABLE `DEPT` (
  `ID` int NOT NULL,
  `Name` varchar(50) NOT NULL,
  `CostCenter` int default NULL,
  UNIQUE KEY `ID` (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


DROP TABLE IF EXISTS `EMP`;
CREATE TABLE `EMP` (
  `UserID` varchar(70) NOT NULL,
  `LastName` varchar(50) NOT NULL,
  `FirstName` varchar(50) NOT NULL,
  `DeptID` int default NULL,
  FOREIGN KEY (`DeptID`) REFERENCES DEPT(`ID`)
                    ON DELETE CASCADE,
  UNIQUE KEY `UserID` (`UserID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

The DEPT table was populated by running the following SQL code.

SQL code that populates the DEPT table

```
insert into `DEPT` values(1, 'Engineering', 100)
insert into `DEPT` values(2, 'Marketing', 200)
insert into `DEPT` values(3, 'QA', 101)
insert into `DEPT` values(4, 'Sales', 300)
```

The EMP table was populated by running the following SQL code.

SQL code that populates the EMP table

```
insert into `EMP` values('jdoe', 'Doe', 'Jane', 1)
insert into `EMP` values('mdavis', 'Davis', 'Mills', 2)
insert into `EMP` values('cparker', 'Parker', 'Charlotte',
 3)
insert into `EMP` values('tmonk', 'Monk', 'Thelma', 4)
insert into `EMP` values('dgillespie', 'Gillespie', 'Desi',
 1)
```

Sending a Text Message

The `User` class has a `sendTextMessage()` method that sends a text message from one user to a set of users and groups. The following example is a method that uses `sendTextMessage()` to send a reminder to the badge of a user.

```
public void sendReminder(String sUserID, String sMessage) {
  try {
    String sSubject = "Reminder";
    User u = User.getUserWithUserID(myVai, sUserID);
    UserSet usMessageTo = new UserSet(myVai);
    usMessageTo.add(u);
    u.sendTextMessage(usMessageTo, sSubject, sMessage);
  } catch (VAIException ve) {
    System.out.println(ve.getMessage());
  }
}
```

Figure 4: Sending a text message

For more information about the `sendTextMessage()` method, see the Javadoc for the `User` class.

Working With Properties

VAI uses collections of key-value pairs called *property sets* to define and manipulate the characteristics of Vocera entities and the Vocera system. The base class is `PropertySet`, an abstract class that extends `java.lang.Object` and defines methods inherited by the following classes:

- The `KeyedPropertySet` class implements methods you can use to manipulate sets of key-value pairs where each key is a string. See Using Keyed Property Sets.
- The `IndexedPropertySet` class implements methods you can use to manipulate indexed sets of values. See Using Indexed Property Sets.

Using Keyed Property Sets

A `KeyedPropertySet` object represents the attributes of a Vocera entity or the Vocera system as a set of key-value pairs where each key is a string. For example, the key-value pair for a user's desk extension could be:

```
"Desk Phone" "1234".
```

Each `Entity` subclass provides a static method (for example, `User.getPropertyKeys`) that returns a String[] array containing the property key names for that class. Also, because some entity property values cannot be changed, each `Entity` subclass provides a static method (for example, `User.getPropertyKeysForUpdate`) that returns a list of the keys for properties that can be updated. The following code prints the property key names for the `User` class.

Getting property keys

```
String[] saKeys = User.getPropertyKeys();
for (int i = 0; i < saKeys.length; i++)
  System.out.println(saKeys[i]);
```

The following listing shows a portion of the output returned by `User.getPropertyKeys()`.

```
User ID
Password
Last Name
First Name
Alt Spoken Names
Alt Spoken Names.*
Ident Phrase
Email Address
...
Buddies
Buddies.*
Buddies.*.Name
Buddies.*.Nick Name
Buddies.*.VIP
Buddies.*.RingTone
...
```

Key names are strings. A key name followed by an asterisk indicates that the corresponding property value is itself an indexed property set. For example, the value of the Alt Spoken Names property is an indexed property set of strings, where each string represents an alternate spoken name.

In addition, the `Entity` class provides a `getProperties` method that returns a complete property set for any given subclass. For example, the following code prints the properties of a specified location.

Getting location properties

```
LocationSet ls = Location.getLocations(myVai);
if (ls.size() > 0) {
  Location loc = ls.elementAt(0);
  KeyedPropertySet kps = loc.getProperties();
  System.out.println(kps.toString());
}
```

Here is an example of the output generated by the previous code example. `KVSet` refers to a keyed property set, whereas `XVSet` refers to an indexed one.

```
KVSet
[
   Name                    = Cafeteria
   Spoken Name             = the caff
   Description             = The main cafeteria
   Site                    = s-global
   Access Points           = XVSet
   [
      1                       = 00064b4e9146
   ]
   Neighbors               = XVSet
   [
      1                       = l-h_q_lobby
   ]
]
```

When you update an entity, use a `KeyedPropertySet` to store the key-value pairs that you want to update. The set should contain only the properties that you want to update. For example, suppose you need to change a user's last name. You would create and submit a property set that contains only one key-value pair: the key "Last Name" and the new value for the user's last name. *Do not* fetch a set of all the user's properties, enter a new last name, and then submit the entire set. You might overwrite changes made by someone else, either through a console or another VAI instance, made in between your fetch and your posting.

Using a KeyedPropertySet to update an entity

```
try {
  KeyedPropertySet kps = new KeyedPropertySet(myVai);
  kps.putString("Description", "NICU Nurse Station");
  Location loc =
    Location.getLocationWithInternalName(myVai,
                                         "l-station3");
  loc.update(myVai, kps);
} catch (VAIException ve) {
  System.out.println(ve.getMessage());
}
```

Using Indexed Property Sets

The `IndexedPropertySet` class extends the `PropertySet` class. An indexed property set is like an array of property values, each of which can be a keyed or indexed property set, or a string. Indexed property sets are homogeneous in the sense that each indexed value is of the same type.

You can specify an index when storing or retrieving an `IndexedPropertySet` element. Index values are integers, and the index of the first element in an `IndexedPropertySet` is 0. You can also put a property value into an `IndexedPropertySet` without specifying an index, in which case the property value is appended to the set.

The following code example shows some techniques for creating and querying an `IndexedPropertySet`.

Creating and querying an IndexedPropertySet

```
public static int createAltSpokenNames() {
  int iResultCode = -1;
  try {
    // Search for existing user.
    String sUid = "Doe,Janet";
    UserSet usGlobal  = User.getUsers(myVai);
    int   iFind = usGlobal.findFirstMatch(sUid);
    User u = usGlobal.elementAt(iFind);

    // Add new alternate spoken names
    IndexedPropertySet ips = new IndexedPropertySet(myVai);
    ips.add("Jane");
    ips.add("J D");
    KeyedPropertySet kpsASN = new KeyedPropertySet(myVai);
    kpsASN.putSet("Alt Spoken Names", ips);
    u.update(myVai, kpsASN);

    // How to get elements from the Alt Spoken Names
 property set
    KeyedPropertySet kpsUser = u.getProperties();
    ips = kpsUser.getIndexedSet("Alt Spoken Names");
    String sASN2 = (String)ips.elementAt(1);
    System.out.println("Alt Spoken Name 2 = " + sASN2);

    iResultCode = 0;
  } catch (VAIException ve) {
    System.out.println(ve.getMessage());
    iResultCode = ve.getResultCode();
  }
  return iResultCode;
}
```

Here is a string representation of the indexed set of alternate spoken names created in the previous code example. In contrast to the integer indexes in the code example (which begin with 0), the indexes shown in the output begin with 1.

```
Alt Spoken Names         = XVSet
   [
      1                        = Jane
      2                        = J D
   ]
```

The following code example shows some techniques for updating elements in an `IndexedPropertySet`.

Updating elements in an IndexedPropertySet

```
public static int updateAltSpokenNames() {
  int iResultCode = -1;
```

```
  try {
    // Search for existing user.
    String sUid = "Doe,Janet";
    UserSet usGlobal  = User.getUsers(myVai);
    int   iFind = usGlobal.findFirstMatch(sUid);
    User u = usGlobal.elementAt(iFind);

    // Get the indexed set of Alt Spoken Names
    KeyedPropertySet kpsUser = u.getProperties();
    IndexedPropertySet ipsASN =
      kpsUser.getIndexedSet("Alt Spoken Names");

    // Replace ASN "Jay Jay" with "Miss Doe"
    String sOldName = "Jay Jay";
    String sCurrentName = "";
    String sNewName = "Miss Doe";
    for (int i = 0; i < ipsASN.size(); i++) {
      sCurrentName = (String)ipsASN.elementAt(i);
      if (sCurrentName.equals(sOldName)) {
        ipsASN.setElementAt(sNewName, i);
      }
    }

    // Update the user's alternate spoken names
    KeyedPropertySet kpsUpdate = new
 KeyedPropertySet(myVai);
    kpsUpdate.putSet("Alt Spoken Names", ipsASN);
    u.update(myVai, kpsUpdate);

    iResultCode = 0;
  } catch (VAIException ve) {
    System.out.println(ve.getMessage());
    iResultCode = ve.getResultCode();
  }
  return iResultCode;
}
```

Persisting Application Data

If your VAI application needs to persist data or settings from one session to the next, you need to consider implementing some type of persistent data storage. There are several ways to do this, including using configuration files or relational database systems. VAI provides a simple way to persist application data by allowing you to write a `PropertySet` file to the Vocera Server, where the application can reliably read the data for subsequent sessions.

The `PropertySet` class provides the following methods for persisting application data on the Vocera Server.

Table 6: Methods for writing and reading application data

Method	Description
`writeApplicationData(java.lang.String sAppName, java.lang.String sFileName, java.lang.String sPropertyPath)`	Writes application data to application data file stored on the Vocera server.
`readApplicationData(VAI vai, java.lang.String sAppName, java.lang.String sFileName, java.lang.String sPropertyPath)`	Reads data from an application data file stored on the Vocera Server into a `PropertySet`.

The following example shows how to use VAI methods to write and read application data.

Writing and reading application data

```
public static void writeAppData(String sAppName, String
 sFile){
```

```
   try {
     //Create a new keyed property set
     KeyedPropertySet kpsApp = new KeyedPropertySet(myVai);
     //Add some properties to the set
     kpsApp.putString("Version", "1.0");
     kpsApp.putString("Role", "Administrator");
     kpsApp.putString("Unit", "CICU");

     //Create an indexed property set of colors
     //and add it to the set
     IndexedPropertySet ipsColors = new
  IndexedPropertySet(myVai);
     ipsColors.add("Red");
     ipsColors.add("Green");
     ipsColors.add("Blue");
     kpsApp.putSet("Colors", ipsColors);

     //Write the application data to a file
     kpsApp.writeApplicationData(sAppName,sFile,"");
   }catch (Exception ex) {
     System.out.println(ex.getMessage());
   }
 }
 public static void readAppData(String sAppName, String
  sFile){
   try {
     //Create a new keyed property set
     KeyedPropertySet kpsApp = new KeyedPropertySet(myVai);
     //Read the application data
     kpsApp =
  (KeyedPropertySet)PropertySet.readApplicationData(
     myVai,
     sAppName,
     sFile,
     ""
   );
     //Print the application data
     System.out.println(kpsApp.toString());
   }catch (Exception ex) {
     System.out.println(ex.getMessage());
   }
 }
```

Here is the output generated from the `readAppData()` method in the previous code example:

```
KVSet
[
   Version                 = 1.0
   Role                    = Administrator
   Unit                    = CICU
   Colors                  = XVSet
   [
      1                       = Red
      2                       = Green
      3                       = Blue
   ]
]
```

For more details about using the `writeApplicationData()` and `readApplicationData()` methods, see the Javadoc reference for the `PropertySet` class.

Managing the Vocera Server

This section describes how to control, manage, and monitor the Vocera Server.

Connecting to the Vocera Server

This topic describes techniques for opening a basic VAI connection, authenticated by a user name and password, to a Vocera Server. See **Security Features** for more information about controlling access to VAI applications and to the Vocera Server.

In the simplest case, you can construct a `VAI` object using the default constructor, then call the `open()` method to establish a connection to the Vocera Server. Once you have opened a `VAI` object, the other methods in `VAI` and those of other classes in the interface can be called. Many of these methods (the static ones in particular) require the opened `VAI` object to be passed as their first argument. At the end of the session, call `close()` to disconnect from the Vocera Server.

Note: VAI cannot open a connection to a server that has been stopped. For example, if a person has stopped the server by clicking the Stop button in the Vocera Control Panel, someone must click the Start button to enable VAI to open a connection.

Using the VAI.open() Method

The `VAI.open()` call takes as arguments the IP address of the Vocera Server, user ID and password, and an instance of the `VAIListener` class. The IP address must be a dotted IP address, for example, `192.168.1.2`. You cannot specify `localhost` or `127.0.0.1`, its equivalent loopback address. Do not specify a port.

To open a connection to a Vocera Cluster installation, specify a comma-separated list of the addresses of the servers in the cluster. If a failover occurs, one of the standby nodes becomes active and takes control of the cluster. The `open()` call also takes a `VAIListener` object as an argument. This argument allows your application to monitor server state changes. Simple applications may not need this information, and can use a null value for the parameter. If your application loses its connection to the Vocera Server, it can monitor the server and automatically reopen a connection when another server in the cluster becomes active. See **Monitoring the Vocera Server** for details.

The user ID and password arguments are those normally prompted for by the Administration Console. You can supply `Administrator` as the user name and the system administration password as the password, or you can supply the user ID and password of a user who has full administrative privileges. For simplicity, examples in this section hard-code the login credentials. When security is a concern, an application should prompt for login credentials at the beginning of each VAI session.

The following code listing shows how to use the `open()` method.

Opening a connection

```
import vai.*;
```

```
public class VaiDemo {
  public static VAI myVai = new VAI();

  private class MyVaiListener implements VAIListener {
    public void handleServerStateChange(int iState) {
      System.out.println("Vocera server state has changed: "
       + VAI.getServerStateString(iState));
    }
    public void handleReportStatus(String s1, int i1,
                                   String s2, int i2,
                                   String s3) {
      System.out.println("Reporting a server status
 change.");
    }
  }

  public MyVaiListener myL;

  public VaiDemo() {
    myL = new MyVaiListener();
  }

  public int demoOpen() {
    int iResultCode = -1;
    try {
      myVai.open("192.168.1.2", "Administrator",
   "admin", this.myL);
      iResultCode = 0;
    } catch (VAIException ve) {
      System.out.println("Code: " + ve.getResultCode());
      System.out.println("Message: " + ve.getMessage());
      iResultCode = ve.getResultCode();
    }
    return iResultCode;
  }

  public static void main(String[] args) {
    VaiDemo demo = new VaiDemo();
    int iResultCode = demo.demoOpen();
    if (iResultCode == 0) {
      myVai.close();
    }
  }
}
```

The following `open()` method passes `null` for the `VAIListener` object, which means it does not monitor server state changes.

```
myVai.open("192.168.1.2","Administrator","admin",null);
```

The following `open()` method passes the hostname, an individual username who has full administrative privileges, the user's password, and a `VAIListener` object, which means it monitors server state changes.

```
myVai.open("vocserver","jdoe","sesame",this.myL);
```

The following `open()` method specifies a cluster of three Vocera Servers, Administrator username and password, and a `VAIListener` object, which means it monitors server state changes.

```
myVai.open("voc1,voc2,voc3","Administrator","admin",this.myL);
```

Note: If Active Directory authentication has been enabled on the Vocera Server, there is an alternative method called `openWithADLogin()` that allows you to open a connection to the Vocera Server using Active Directory credentials. For details, see the Javadoc for the VAI class.

Result Codes for the open() Method

If a connection cannot be established, the `open()` method returns a result code to indicate why it failed. The following table lists some of the reasons that the `open()` method might fail. For complete information, see the Javadoc for the `VAIException` class.

Table 7: Error codes for a failed connection

Error Code	Message
rcCannotConnect	Could not connect to server.
rcConnectionRefused	Connection to server refused.
rcInvalidPassword	Invalid password.
rcLicenseLimit	No more user licenses available.
rcLoginLimit	No more login licenses available.

Getting Vocera Server Properties

The `VAI` class provides the following methods for querying the properties of a Vocera Server.

Table 8: Methods for querying the properties of a Vocera Server

Method	Description
`getSystemProperties()`	Returns a keyed property set for a specified Vocera system. Properties include Product Major Version, Product Major Version, Locale, and Voice Prints Enabled. See **VAI Example** for a code example.
`getServerStateString(int iServerState)`	Returns a string describing the server's current state. Values include "Could not connect to server", "Server stopped", and "Server started". A full list of server states is found in the `VAIListener` interface.
`getLicenseInfo()`	Returns a `LicenseInfo` object. A `LicenseInfo` object exposes several public fields that represent various aspects of a Vocera license. For example, the cDigitalLines field stores an integer value representing the maximum allowed number of digital phone lines.

The following code example uses a `LicenseInfo` object to get information about the number of digital phone lines allowed and in use by a server.

Getting license information

```
public int getDigitalLinesInfo() {
  int iResultCode = -1;
  try {
    LicenseInfo li = myVai.getLicenseInfo();
    int iMaxLines = li.cDigitalLines;
    int iCurrLines = li.cCurrentDigitalLines;
    System.out.println("Currently using " +
                       iCurrLines +
                       " of " +
                       iMaxLines +
                       " available lines.");
    iResultCode = 0;
  } catch (VAIException ve) {
    iResultCode = ve.getResultCode();
  }
  return iResultCode;
}
```

Setting Vocera Server Properties

The `VAI` class provides the `updateSystemProperties()` method. This method takes one argument, a property set that contains the property values you want to update.

The following code example uses the `updateSystemProperties()` method to set properties for Company, Days To Keep Messages, and Default User.Low Battery Alert.

Updating Vocera Server properties

```
private void updateSysProps() {
  try {

    // Create a new keyed property set for system properties
    KeyedPropertySet kpsSys = new KeyedPropertySet(myVai);
    kpsSys.putString("Company","Vocera Communications");
    kpsSys.putInt("Days To Keep Messages", 7);

    // Create a keyed property set for Default User
 properties
    KeyedPropertySet kpsDefUser = new
 KeyedPropertySet(myVai);
    kpsDefUser.putBoolean("Low Battery Alert", false);

    // Add Default User property set to the System property
 set
    kpsSys.putSet("Default User", kpsDefUser);

    // Update system properties
    myVai.updateSystemProperties(kpsSys);

    } catch (VAIException ve) {
      System.out.println(ve.getMessage());
    }
  }
```

Controlling the Vocera Server

If you have an open VAI connection, you can stop and start a Vocera Server programmatically. However, VAI cannot open a connection to a server that has been stopped. For example, if a person has stopped the server by clicking the Stop button in the Vocera Control Panel, someone must click the Start button to enable VAI to open a connection. Similarly, if VAI code stops the server and then closes the connection, you will not be able to restart the server programmatically.

Note: Because you cannot connect to the server when it is stopped, you may want to embed the `open()` call in a loop that repeatedly tries to open a connection until it succeeds.

The `VAI` class implements the following methods for controlling a Vocera Server.

Table 9: Methods for controlling the Vocera Server

Method	Description
`startServer()`	Starts the Vocera Server, as if you had clicked the Run button in the Vocera control panel.
`stopServer()`	Stops the Vocera Server, as if you had clicked the Stop button in the Vocera control panel.
`restartServer()`	Shuts down the Vocera Server and all associated services (simulating a fail-over), and then restarts them. Calling `restartServer()` is *not* the same as calling `stopServer()` and then calling `startServer()`. The effects of calling `restartServer()` are more drastic.

The following code example opens a connection to a Vocera Server, stops the server, and then starts the server. This simple example is designed to demonstrate several `VAI` class methods. In a production environment, when you really want to restart the server (as opposed to stopping the server, doing something, and then starting the server), use the `restartServer()` method. Moreover, rather than depending on timers, you should rely on `VAIListener` call-backs to track changes to server states. See Monitoring the Vocera Server.

Stopping and starting the Vocera Server

```
public static void stopStartServer() {
  VaiDemo demo = new VaiDemo();
    try {
      myVai.open("10.0.1.2",
                 "Administrator",
                 "admin",
                 demo.myL);

      myVai.stopServer();

      // Give the server time to shut down.
      Thread.sleep(20000); // 20 seconds

      myVai.startServer();

      // Give the server time to restart.
      Thread.sleep(20000); // 20 seconds

      myVai.close();

    } catch (VAIException ve) {
      System.out.println("VAI Exception: " +
 ve.getMessage());
    } catch (InterruptedException ie) { // For Thread.sleep
      System.out.println(ie.getMessage());
    }
  }
```

Managing the Vocera Database

The `VAI` class implements the following methods for managing a Vocera database.

Table 10: Methods for managing a Vocera database

Method	Description
`backup()`	Backs up the Vocera database, creating a new .zip file in `\vocera\backup`.
`restore(java.lang.String sFileName)`	Restores the Vocera database from a specified backup file in the `\vocera\backup` directory.
`emptyDatabase()`	Empties the Vocera database.
`getBackupFileNames()`	Retrieves the names of backup files in the `\vocera\backup` directory on the Vocera Server computer.

The following code example prints a list of backup files, prompts the user to enter a backup file name, then uses the specified file to restore the Vocera database. Only the file name needs to be specified, not the full path. The path is <Vocera_Drive>:`\vocera\backup` on the Vocera Server.

Restoring the Vocera database from a backup file

```
public static void promptAndRestore() {
  VaiDemo demo = new VaiDemo();
  try {
    myVai.open("qalab4",
               "Administrator",
```

```
                "admin",
                demo.myL);

    // Retrieve the array of backup filenames
    String[] saBakFiles = myVai.getBackupFileNames();
    System.out.println("Backup Files:");
    for (int i = 0; i < saBakFiles.length; i++) {
      System.out.println(saBakFiles[i]);
    }
    System.out.println("Enter the name of a Backup File: ");

    java.io.BufferedReader in =
      new java.io.BufferedReader(
          new java.io.InputStreamReader(System.in)
      );
    String sFileName = in.readLine();

    // Restore the backup file.
    // (Error checking omitted for simplicity.)
    myVai.restore(sFileName);

    // Close the connection
    myVai.close();

  } catch (VAIException ve) {
    System.out.println("VAI Exception: " + ve.getMessage());
  } catch (java.io.IOException ioe) { // For in.readLine
    System.out.println(ioe.getMessage());
  }
}
```

Monitoring the Vocera Server

The `VAI.open()` call takes a `VAIListener` object as an argument. This argument allows your application to monitor server state changes. Simple applications may not need this information, and can use a null value.

If you choose to monitor server state changes in your application, make sure you implement the following methods defined in the `VAIListener` interface:

Table 11: Methods for monitoring the Vocera Server

Method	Description
`handleServerStateChange(int iState)`	Starting, stopping, and closing the VAI connection to the server all trigger events that your `VAIListener` implementation can handle based on the current state of the server. The iStatus parameter of the `handleServerStateChange()` method represents a server status code defined in `VAIListener`. **Important:** The `VAIListener` runs on an internal call-back thread separate from the `VAI` thread. Therefore, you cannot make calls from `VAIListener` back to the `VAI` instance. Otherwise, your program may hang.
`handleReportStatus(java.lang.String sTitle, int iStatus, java.lang.String sStatus, int iPercentDone, java.lang.String sError)`	Handles status information reported as a result of bulk operations, such as database backup and restore operations.

The following code listing shows how a simple listener responds as the server is stopped and restarted.

Using VAIListener to monitor the Vocera Server

```
import vai.*;

public class VaiDemo {
  public static VAI myVai = new VAI();
  public static String sServerIP =
 "10.37.41.20,10.37.41.21";
  public static String sAdminUser = "Administrator";
  public static String sAdminPassword = "admin";

  private class MyVaiListener implements VAIListener {

    boolean bStopped = false;
    boolean bStarted = true;

    public void handleServerStateChange(int iState)
      System.out.println("Vocera Server state has changed:
 " +
                         VAI.getServerStateString(iState));
      if (iState==VAIListener.ssStopped) {
        bStopped = true;
        bStarted = false;
      } else if (iState==VAIListener.ssStarted) {
        bStarted = true;
        bStopped = false;
      }
    }
    // handleReportStatus is defined in VAIListener
    // Note:  iStatus is one of rs codes in VAIListener,
    // sError is empty unless iStatus == rsError.
    public void handleReportStatus(String sTitle,
                  int iStatus,
                  String sStatus,
                  int iPercentDone,
                  String sError)
    {
      System.out.println("Reporting a server status
 change.");
      System.out.println("[Report Title] \t" + sTitle);
      System.out.println("[Status Code] \t" + iStatus);
      System.out.println("[Status Msg] \t" + sStatus);
      System.out.println("[Percent Done] \t" +
 iPercentDone);
      System.out.println("[Error Msg] \t" + sError);
      System.out.println("End of status report.");
    }
  }

  public MyVaiListener myL;

  public VaiDemo() {
    myL = new MyVaiListener();
  }

  public static void main(String[] args) {
    VaiDemo demo = new VaiDemo();

    try {
      int iResultCode = demo.openConnection();
      if (iResultCode==0) {
        myVai.stopServer();
        while (!demo.myL.bStopped) {
          Thread.sleep(5000);
        }
        myVai.startServer();
        while (!demo.myL.bStarted) {
          Thread.sleep(5000);
        }
        demo.openConnection();
      }
    } catch (Exception ex) {
      System.out.println(ex.getMessage());
    }
```

```
  }

  private void openConnection() {
    int iResultCode = rcInitResult;
    try {
      System.out.println("Opening connection to server...");
      myVai.open(sServerIP,sAdminUser,sAdminPassword, myL);
      iResultCode = rcOK;
    } catch (VAIException ve) {
      System.out.println(ve.getMessage());
      iResultCode = ve.getResultCode();
    }
    return iResultCode;
  }
}
```

Vocera Server States

When you call the `handleServerStateChange()` method defined in the `VAIListener` interface, you can specify how to handle each state change that occurs to the Vocera Server while your application is running. For example, if you determine that the server is started, you can reopen the connection to it.

Table 12: Vocera Server states

Server State	Description
ssCancelStart	Server start cancelled.
ssEmpty	Empty operation in progress.
ssNotConnected	No connection to server.
ssRestore	Restore operation in progress.
ssStandby	Server in cluster standby mode.
ssStarted	Server process started.
ssStarting	Server process starting.
ssStopped	Server process stopped.
ssStopping	Server process stopping.

Error Handling

VAI uses the Java exception mechanism to report runtime errors. When a method triggers a error, the Java runtime framework creates an *exception object* that contains information about the error. This process is called *throwing an exception*. To handle the error, an application must *catch* the exception.

For example, the `Address.create()` method throws an exception when it triggers a runtime error. Therefore, you must wrap the `Address.create()` call in a `try...catch` block, as shown in the following code example.

Catching an exception

```
KeyedPropertySet kpsAddress = new KeyedPropertySet(myVai);
kpsAddress.putString("Last Name", "Jones");
try {
  Address a = Address.create(myVai, kpsAddress);
} catch (VAIException ve) {
 System.out.println(ve.getMessage());
}
```

Using the VAIException Class

Most of the methods exposed in VAI classes throw a `VAIException`, defined in the class of that name, in case of an error. An error may be thrown for many reasons, including among others:

- The VAI connection to the Vocera server was not open when the method was called.
- One or more method arguments are invalid.
- The entity on which the method was called has been deleted.

The `VAIException` class contains two methods to help you determine the reason for the error and display it to the user of your application, if desired:

Table 13: Methods for returning information about exceptions

Method	Description
`getResultCode()`	Returns one of the integer constants defined in the `VAIException` class. VAI error code values are all greater than 0. Consequently, you can use a value of 0 as the result code for a successful operation. The `VAIException` class defines integer constants such as `rcLicenseLimit` to represent VAI error codes. These constants are described in the Javadocs for the `VAIException` class.
`getMessage()`	Returns a locale-specific string containing an error message.

The following code example shows how `getMessage()` and `getResultCode()` can provide information when an error occurs.

Getting exception messages and result codes

```
// Initial result code value.
public final static int rcInitResult = -1;
// Result code for successful operation.
public final static int rcOK = 0;

public int open(String sHost, String sUserName, String
 sPassword) {
  int iResultCode = rcInitResult;
  try {
    myVai.open(sHost, sUserName, sPassword, null);
    iResultCode = rcOK;
  } catch (VAIException ve) {
    iResultCode = ve.getResultCode();
    if (iResultCode == VAIException.rcLicenseLimit) {
      System.out.println(ve.getMessage());
  }
}
```

Security Features

This section describes VAI security features.

Controlling Access

VAI applications can use certificates to authenticate users with a Vocera Server. The `VAI` class provides the following methods for working with certificates.

Table 14: Methods for working with certificates

Method	Description
`makeCertificateString(java.lang.String sAdminLogin, java.lang.String sAdminPassword, java.lang.String[] saAppPasswords, boolean bUserIDPassword)`	Creates a digital certificate, represented as a String, to be passed to the `openWithCertificateString()` method. You can specify null instead of an array of passwords when you create a certificate string. A user can then log in to the VAI application by providing his or her Vocera password.
`makeCertificateFile(java.lang.String sFileName, java.lang.String sAdminLogin, java.lang.String sAdminPassword, java.lang.String[] saAppPasswords, boolean bUserIDPassword)`	Like `makeCertificateString()`, but stores the certificate in a file with the given fully-qualified name, to be passed to the `openWithCertificateFile()` method. You can also use the `mc.bat` utility on the **Vocera Developer Kit** CD to create a certificate file. See **Using the mc.bat Utility**.
`makeAppCertificateFile(java.lang.String sAppName, java.lang.String sFileName, java.lang.String sAdminLogin, java.lang.String sAdminPassword, java.lang.String[] saAppPasswords, boolean bUserIDPassword)`	Like `makeCertificateFile()`, but stores the certificate file with the given application on the Vocera Server, to be passed to the `openWithAppCertificateFile()` method.
`openWithCertificateString(java.lang.String sServerList, java.lang.String sLogin, java.lang.String sPassword, java.lang.String sCertificate, VAIListener l)`	Opens the VAI interface object using an application password and a certificate represented as a String.
`openWithCertificateFile(java.lang.String sServerList, java.lang.String sLogin, java.lang.String sPassword, java.lang.String sFileName, VAIListener l)`	Opens the VAI interface object using an application password and a certificate file.
`openWithAppCertificateFile(java.lang.String sServerList, java.lang.String sLogin, java.lang.String sPassword, java.lang.String sAppName, java.lang.String sFileName, VAIListener l)`	Opens the VAI interface object using an application password and a certificate file stored with the given application on the Vocera Server. This method is useful for developing secure GUI applications hosted on the Vocera Server that do not require System Administrator permission to log in.

The VAI security mechanism supports application-specific passwords, so you can enable users to log in to a VAI application without giving them a Vocera administrator user name and password.

The following code example combines several aspects of working with certificates and application-specific passwords. In practice, you would probably perform the steps separately.

Using certificates with application passwords

```
public int openUsingCert1() {
  int iResultCode = -1;
  try {
    String[] saPass = { "green", "yellow", "purple" };
    String sCert =
      VAI.makeCertificateString("Administrator",
                "admin",
                saPass,
                false);
    myVai.openWithCertificateString("192.168.1.1",
                  "mdavis",
                  "yellow",
                  sCert,
                  myL);
    iResultCode = 0;
  } catch (VAIException ve) {
    iResultCode = ve.getResultCode();
  }
  return iResultCode;
}
```

Important: You should never hard-code passwords within an application. Always provide a login user interface with which a user can supply a password.

You can also use certificates without specifying application passwords, as shown in the following code example. This example is designed to show several related techniques at a glance. In a production application, you would perform these steps separately.

Using certificates without application passwords

```
public int openUsingCert2() {
  int iResultCode = -1;
  try {
    String sCert = VAI.makeCertificateString("Administrator",
                        "admin",
                        null,
                        true);
    myVai.openWithCertificateString("192.168.1.1",
                  "mdavis",
                  "sowhat",
                  sCert,
                  myL);
    iResultCode = 0;
  } catch (VAIException ve) {
    iResultCode = ve.getResultCode();
  }
  return iResultCode;
}
```

Using the mc.bat Utility

Vocera provides a batch file named `mc.bat` that you can use to encrypt login credentials and create a certificate file. The batch file provides a simple command-line interface that prompts you for the values needed to create a certificate.

THIS PARAGRAPH WAS AFTER A TASK, IN DOCBOOK The following figure shows an example of how the `mc.bat` utility was used to create a certificate file named `certificate.txt`.

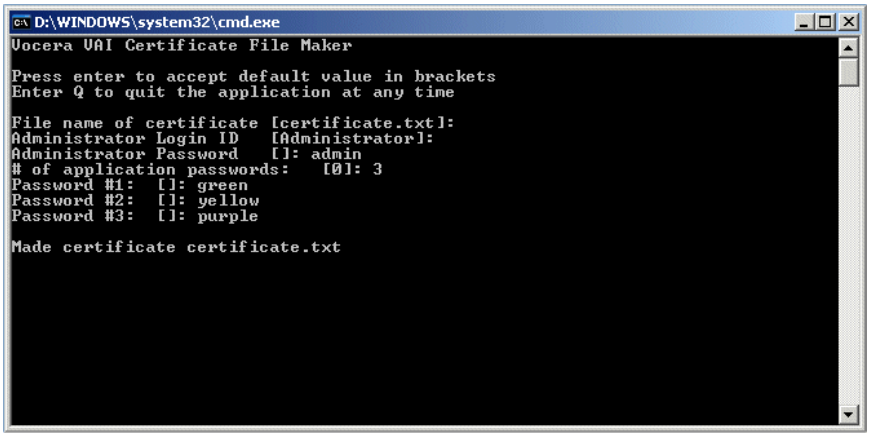

Figure 5: Using mc.bat to create a certificate

To use the mc.bat utility to create a certificate file:

1. Copy the `mc.bat` file from the `VAI\server` directory on the **Vocera Developer Kit** CD into a location on your Vocera server machine.
2. Run `mc.bat`.

   The program opens a Command Prompt window, allowing you to enter several parameters needed to create a certificate file.
3. Enter values for the filename, administrator login ID, administrator password, number of application passwords, and each individual application password.

   When you are finished, the utility creates a certificate file in the same directory.

VAI and Tiered Administrators

The Vocera Administration Console lets you grant users different levels of access to administrative features, effectively distributing administration responsibility for the Vocera server to several tiered administrators. Tiered administrators are Vocera users with some but not all administrative privileges based on their membership in one or more groups. For more information about tiered administrators, see the *Vocera Administration Guide*.

Unlike the Vocera Administration Console, VAI does not support tiered administrators. Anyone who is able to log into a VAI application has access to whatever Vocera administrative features that the application exposes. However, you can choose to expose only certain administrative features in your VAI application, or you can restrict the application to only certain users.

Encrypted Passwords

You can use VAI to update passwords, but you cannot retrieve them. VAI uses strong public key cryptography to protect passwords. Once encrypted, passwords are never decrypted anywhere within VAI and within the Vocera Server code.

You should not hard-code credentials into your application. Prompt for them at the beginning of each session.

Authorizing VAI Applications

On the System > License Info tab of the Vocera Voice Server Administration Console, the Vocera administrator may optionally enter a comma-separated list of IP addresses in the VAI Application IP Addresses field to limit the list of computers which are allowed to establish VAI connections to the Vocera Voice Server. If you leave this field blank, a VAI application from any machine is allowed to connect to the Vocera Voice Server. Applications still need to authenticate to access data on the Vocera Voice Server. For more information, see the *Vocera Administration Guide*.

A VAI application that is deployed directly on any node of a Vocera Voice Server cluster is automatically authorized to connect; you do not have to add it explicitly to the VAI Application IP Addresses field.

Best Practices for Multiuser Applications

A VAI application can be designed to support multiple simultaneous users. For example, you can develop a Web application with a client interface that runs in a browser. For best performance, your multiuser VAI application should follow these guidelines:

- **Use a shared connection** – To optimize performance and network I/O and to reduce the multiuser stress on the server, you should design multiuser VAI applications to use only a single, shared connection to the Vocera Server.

  Important: The Vocera Server cannot handle many simultaneous VAI connections. You should always test your multiuser application on a test server to see if it can handle the load of multiple simultaneous users.

- **Open the connection with a digital certificate** – If your application is designed for people without System Administration permission, use one of the `VAI` methods to create a digital certificate to authenticate users logging into the application. When you open the VAI connection, use one of the `VAI` methods to open a connection using the digital certificate. See Controlling Access.
- **Check login credentials** – Implement methods to verify the credentials of each user logging into the application. The `VAI` class provides several methods for verifying login credentials.
- **Make your application thread-safe** – To prevent thread interference and memory consistency errors, synchronize access to shared resources.

A servlet is one example of a multiuser Web application. The following figure shows a servlet running in Tomcat on the Vocera Server computer. Multiple users can connect to the servlet using a browser.

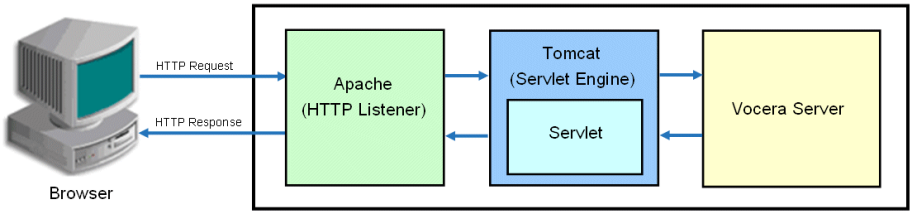

Figure 6: Servlet example

Users can log into a multiuser VAI application using one of the following types of credentials:

- **Administrator credentials** – System Adminstrator credentials for the Vocera system. This can be the Vocera user ID and password of a user with System Administrator permission, or the "Administrator" ID and password.
- **User credentials** – Vocera user ID and password. The Vocera password can be a null string (""). However, for security reasons your application should require a non-blank password.
- **Application credentials** – VAI application credentials that satisfy an application certificate.

The `VAI` class provides the following methods for checking whether login credentials are valid.

Table 15: Methods for working with certificates

Method	Description
`checkAdminCredentials(java.lang.String sLogin, java.lang.String sPassword)`	Returns `true` if the credentials are for a valid Vocera System Administrator.
`checkADAdminCredentials(java.lang.String sLogin, java.lang.String sPassword, java.lang.String sADConfigName)`	Returns `true` if the Active Directory credentials are for a valid Vocera System Administrator.
`checkUserCredentials(java.lang.String sLogin, java.lang.String sPassword)`	Returns `true` if the credentials are for a valid Vocera user.
`checkADUserCredentials(java.lang.String sLogin, java.lang.String sPassword, java.lang.String sADConfigName)`	Returns `true` if the Active Directory credentials are for a valid Vocera user.
`checkAppCredentials(java.lang.String sLogin, java.lang.String sPassword, java.lang.String sAppName, java.lang.String sFileName)`	Returns `true` if the credentials are valid for the application certificate.

For more information about VAI methods for checking login credentials, see the VAI Javadoc reference.

Property Reference

The following topics are a reference to the property keys and values of various VAI entities. All property values can be updated except where specifically indicated.

Address Properties

The following table lists the properties of an Address Book Entry. The Vocera address book is a convenient way for badge users to contact places and people who are not badge users.

Since: 4.0

Table 16: Address properties

Key	Description
Last Name	The last name of a person or the name of a place. **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
First Name	The first name of a person. If the Address is a place rather than a person, enter "" (an empty string). **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
Alt Spoken Names	Property set containing up to three variations of the spoken name of the person or place. **Datatype:** IndexedPropertySet **Required:** No
Alt Spoken Names.*	Represents each Alternate Spoken Name in the property set. **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Ident Phrase	An identifying phrase that distinguishes a person or place from another with the same name. Example: `Rita Clark in Staffing` **Datatype:** String **Maximum Length:** 100 characters **Required:** No

Key	Description
Email Address	An optional email address, which allows users to send voice messages as an email attachment. Example: `jdoe@vocera.com` **Datatype:** String **Maximum Length:** 40 characters **Required:** No
Desk Phone	The desk phone number or extension for the person or place. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Pager Phone	Pager number for the person or place. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Site	The home site for the person or place. If the entire organization uses this address book entry, choose the Global site. If you don't specify a site, the Global site is used. **Datatype:** Site object, or a string representing the site's internal name. **Required:** No

Contact Properties

The following table lists the properties of a Contact. The `Contact` class represents an outside buddy as a VAI entity.

Since: 4.0

Table 17: Contact properties

Key	Description
Last Name	An outside buddy's last name. **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
First Name	An outside buddy's first name. **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
Email Address	An optional email address, which allows users to send voice messages as an email attachment. Example: `jdoe@vocera.com` **Datatype:** String **Maximum Length:** 60 characters **Required:** No
Desk Phone	The desk phone number or extension for the outside buddy. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Pager Phone	Pager number for the outside buddy. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Owner	The user for whom this is a personal contact (that is, the owner of this outside buddy). Cannot be updated. **Datatype:** User object

Device Properties

The following table lists the properties of a Device. The `Device` class represents a device, such as a badge, that connects to the Vocera system.

Since: 4.1

Table 18: Device properties

Key	Description
MAC Address	Specifies the unique MAC address of the device. **Datatype:** String **Maximum Length:** 12 characters **Required:** Yes
Serial No	The device serial number. For B2000 badges, the serial number is 12 characters. For B1000A badges, the serial number is 15 characters. **Datatype:** String **Maximum Length:** 15 characters **Required:** No
Label	The label applied to the device for identification purposes. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Status	The device status. The value you specify must match one of the existing device status values. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Tracking Time	A time used to track the device. This time could be mapped to any internal tracking event, such as the date when the device was assigned to a user or sent for repair. The time value is specified as the number of milliseconds since 1/1/70 00:00 GMT. **Datatype:** long **Required:** No
Owning Group	The group that owns the device. **Datatype:** Group object, or a string representing the group's internal name. **Required:** No
Notes	Notes about the device. For example, you could include more information about the device status. **Datatype:** String **Maximum Length:** 1000 characters **Required:** No
Site	The device's home site. If you don't specify a site, the Global site is used. **Datatype:** Site object, or a string representing the site's internal name. **Required:** No
Shared	Indicates whether the device is shared by multiple users. **Datatype:** boolean **Required:** No

Group Properties

The following table lists the properties of a Group. Vocera groups organize users into roles such as Floor Manager, Cashier, Nurse, Cardiologist, Executive, and so forth.

Since: 4.0

Table 19: Group properties

Key	Description
Group Type	A string indicating the type of group. Enter `Ordinary` or `Department` **Datatype:** String **Required:** No
Name	The name of the group. The name must start with a letter or digit. It must contain only letters, digits, spaces, apostrophes ('), underscores (_), or dashes (-). No other characters are allowed. **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
Spoken Name	An optional alternate spoken name for the group. For example, some people might say "the Sales team" instead of "Sales." If you enter `the Sales team` as a spoken name, the Genie will recognize "Call the sales team." **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Spoken Member Name	Enter a name that describes a member of the group. For example, in the group called Sales, a group member would be known as `a sales person`. This would allow the Genie to recognize a command such as, "Call a sales person." **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Spoken Members Name	Optional plural name that collectively describes the members of the group. For example, in the group called Sales, the collection of group members could be called `sales people`. This would allow the Genie to recognize a command such as, "Send a message to all sales people." **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Phone	Phone number or extension. If the telephony integration option is installed, outside callers who dial the Vocera hunt number can connect to the group by entering the group extension at the Genie prompt, instead of saying the group name. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Pager	Pager number for the group. You can configure Vocera to forward a group's calls to this specified pager. **Datatype:** String **Maximum Length:** 75 characters **Required:** No **Since:** 4.1
Scheduling Type	Specify a scheduling option to indicate how calls to the group should be distributed. Enter `Sequential` or `Round Robin`. Choose `Sequential` if you want one person to be the main contact. The second member in the group list is called only if the first person is not available, a third member is called only if the first two are unavailable, and so forth. Choose `Round Robin` if you want calls to be distributed as evenly as possible among group members. When you choose round robin, Vocera iterates through members in the group until someone accepts the call; however, the person who most recently accepted a group call is tried last. **Datatype:** String **Required:** No

Key	Description
PIN	Specify a value of the PIN for long distance calls. A telephony PIN authorizes members of a Vocera department to make phone calls and allows an organization to charge departments for those calls. A PIN can include digits, special characters, and PIN macros. Enter a PIN value only if you are working with a department group. **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Cost Center	The department's Cost Center ID, which enables Vocera to track system usage by department and potentially allows an organization to charge its departments for relative usage. Enter a Cost Center value only if you are working with a department group. **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Auto Remove	Specifies whether membership in the group is temporary. If `true`, Vocera automatically removes users from the group when they log out, but leaves the rest of the user profile in the database. Users are not added into the group automatically when they log back in. **Datatype:** boolean **Required:** No
Off Site Calls	Specifies whether calls to members of the group can be received at sites other than the group's home site. This property behaves the same for all groups, including groups assigned to the Global site. If your Vocera system has only one site, this property does not apply. **Datatype:** boolean **Required:** No **Since:** 4.2
Off Site Broadcasts	Specifies whether broadcasts to members of the group can be received at sites other than the group's home site. This property behaves the same for all groups, including groups assigned to the Global site. If your Vocera system has only one site, this property does not apply. **Datatype:** boolean **Required:** No **Since:** 4.2
No Call	Specifies whether the group is used to grant or revoke permissions only and should not be callable. If `true`, calling and broadcasting to this group is disabled. **Datatype:** boolean **Required:** No **Since:** 4.2
Site	The group's home site. If your organization has multiple sites connected to the same Vocera server, specify the home site that represents the member's physical location. If the group's membership spans multiple sites, specify the Global site. **Datatype:** Site object **Required:** No

Key	Description
Forwarding	Specify a forwarding option to indicate whether calls should be forwarded, and, if so, where to forward them. Enter `None`, `Phone`, `Pager` or `User`. • `None` means that if a call to the group is not answered, the caller is prompted to leave a message, and that message is delivered to all members of the group. • `Phone` transfers the unanswered call to the number that you enter for the Forwarding Number property. This feature requires the telephony integration option. • `Pager` transfers the unanswered call to the number that you enter for the Pager property. If the value for the Pager property is empty, this option is invalid. This feature requires the telephony integration option. • `User` transfers the call to a particular badge user, group, or address book entry when no members of the original group can take the call. **Datatype:** String **Required:** No
Forwarding Number	Phone number used when Forwarding = `Phone` **Datatype:** String **Required:** No
Forwarding Name	The User, Address, or Group to forward to when Forwarding = `User` **Datatype:** Entity **Required:** No
Forwarding When	Specify which calls to forward. Enter `All` to forward all calls, or `Unanswered` to forward only unanswered calls. **Datatype:** String **Required:** No
Manager Group	The group of users permitted to manage this group. Specify a group that has management privileges. **Datatype:** Group **Required:** No
Member Domain Group	The group of users permitted to add themselves to this group. **Datatype:** Group **Required:** No
Device Manager Group	The group of users permitted to manage the devices for this group. **Datatype:** Group **Required:** No **Since:** 4.1
Permissions	Permissions granted to members of this group. For the properties in this set, a value of `true` explicitly grants that permission for the group. **Datatype:** KeyedPropertySet **Required:** No
Permissions.Perform System Administration	Sets whether to grant permission to perform system administration, which gives a group full administrative privileges in the Administration Console, and automatically grants those group members every other permission. This permission overrides any revoked permissions inherited by membership in other groups, except the revoked Perform Server Administration permission itself. **Datatype:** boolean **Required:** No

Key	Description
Permissions.Record Name Prompts for Another User	Sets whether to grant permission to record name prompts for other users, as well as groups and address book entries. Name prompts improve the usability of the Vocera system; the Genie plays these name prompts when necessary, instead of synthesizing speech. **Datatype:** boolean **Required:** No
Permissions.Log In as Another User	Sets whether to grant permission to log in as someone else, ignoring any voiceprint authentication. This permission is useful when an administrator needs to log in as another user for whom voiceprint authentication has been enabled. **Datatype:** boolean **Required:** No
Permissions.Call Internal Numbers	Sets whether to grant permission to place calls to internal telephone extensions by saying the key phrase "Dial extension" (for example, "Dial extension 4085"). This feature requires Telephony Integration. **Datatype:** boolean **Required:** No
Permissions.Call Toll-Free Numbers	Sets whether to grant permission to place calls to phone numbers in toll-free calling areas. This feature requires Telephony Integration. **Datatype:** boolean **Required:** No
Permissions.Call Toll Numbers	Sets whether to grant permission to place calls to phone numbers that are not in toll-free calling areas. This feature requires Telephony Integration. **Datatype:** boolean **Required:** No
Permissions.Forward Calls to Badges	Sets whether to grant permission to forward incoming calls to other badges. When this permission is granted, users can specify forwarding options through either the User Console or voice commands. **Datatype:** boolean **Required:** No
Permissions.Forward Calls to Internal Numbers	Sets whether to grant permission to forward incoming calls to internal phone numbers. This feature requires Telephony Integration. When this permission is granted, users can specify forwarding options through either the User Console or voice commands. **Datatype:** boolean **Required:** No
Permissions.Forward Calls to Toll-Free Numbers	Sets whether to grant permission to forward incoming calls to phone numbers in toll-free calling areas. This feature requires Telephony Integration. When this permission is granted, users can specify forwarding options through either the User Console or voice commands. **Datatype:** boolean **Required:** No
Permissions.Forward Calls to Toll Numbers	Sets whether to grant permission to forward incoming calls to phone numbers that are not in toll-free calling areas. This feature requires Telephony Integration. When this permission is granted, users can specify forwarding options through either the User Console or voice commands. **Datatype:** boolean **Required:** No
Permissions.Initiate Broadcasts	Sets whether to grant permission to broadcast to all users in a group at the same time. **Datatype:** boolean **Required:** No
Permissions.Initiate Broadcasts to Everyone	Sets whether to grant permission to broadcast to all users in the Everyone group for your site. **Datatype:** boolean **Required:** No

Key	Description
Permissions.Initiate Urgent Broadcasts	Sets whether to grant permission to broadcast an urgent call to every member in a group at the same time. An urgent broadcast has priority and breaks through to everyone's badge, even if the badge is blocking calls or is in DND mode. See the *Vocera User Guide* for more information about urgent broadcasts. **Datatype:** boolean **Required:** No
Permissions.Place Urgent Calls	Sets whether to grant permission to place an urgent call or initiate an urgent three-way conference call. An urgent call or urgent three-way conference call has priority and breaks through to a badge, even if the badge is blocking calls or is in DND mode. See the *Vocera User Guide* for more information about urgent calls. **Datatype:** boolean **Required:** No
Permissions.Call Users at Other Sites	Sets whether to grant permission to contact a user whose home site or current site is different from the home site or current site of the caller. **Datatype:** boolean **Required:** No
Permissions.Join Conference	Sets whether to grant permission to enter or leave a conference. Vocera does not require users to have a permission to use a conference; that is, any user who is in a conference has access to the conference feature. **Datatype:** boolean **Required:** No
Permissions.Send Messages To Everyone	Sets whether to grant permission to send a message to all users in the Everyone group for your site. **Datatype:** boolean **Required:** No
Permissions.Have Toll-Free Pager Number	Sets whether to grant permission to have a pager number that is in a toll-free calling area. This feature requires Telephony Integration. Vocera does not require users to have permission to call pagers. If you allow users the permission to have pager numbers, you are implicitly allowing other users the permission to call those numbers. **Datatype:** boolean **Required:** No
Permissions.Have Toll Pager Number	Sets whether to grant permission to have a pager number that is in a toll calling area. This feature requires Telephony Integration. Vocera does not require users to have permission to call pagers. If you allow users the permission to have pager numbers, you are implicitly allowing other users the permission to call those numbers. **Datatype:** boolean **Required:** No
Permissions.Require Authentication to Log In	Sets whether group members must recite a series of random digits when they log in. If the voice does not match the recorded voiceprint, users cannot log in. This permission has no effect until a user records a voiceprint. Also, this permission is effective only if the Voice Prints Enabled system property is set to `true`. **Datatype:** boolean **Required:** No

Key	Description
Permissions.Require Authentication to Play Messages	Sets whether group members must recite a series of random digits when they play messages. If the voice does not match the recorded voiceprint, users cannot play messages. This permission has no effect until a user records a voiceprint. Also, this permission is effective only if the Voice Prints Enabled system property is set to `true`. **Datatype:** boolean **Required:** No
Permissions.Record your Voiceprint	Sets whether to grant permission to record their voiceprint. This permission is effective only if the Voice Prints Enabled system property is set to `true`. **Datatype:** boolean **Required:** No
Permissions.Erase your Voiceprint	Sets whether to grant permission to erase their previously-recorded voiceprints. This permission is effective only if the Voice Prints Enabled system property is set to `true`. **Datatype:** boolean **Required:** No
Permissions.Erase Voiceprint of Another User	Sets whether to grant permission to erase the voiceprint of another user. This permission is effective only if the Voice Prints Enabled system property is set to `true`. **Datatype:** boolean **Required:** No
Permissions.Locate Users or Group Members	Sets whether to grant permission to locate other users or group members. You can then issue badge commands such as "Where is Melissa Schaefer?" to find the physical location of a user or group member. This feature is useful only if location names have been defined and access points have been assigned to locations. **Datatype:** boolean **Required:** No
Permissions.Have VIP Status	Sets whether to grant permission to complete a call even when users are blocking calls or have placed their badges in Do Not Disturb mode. **Datatype:** boolean **Required:** No
Permissions.Block and Accept Calls	Sets whether to grant permission to issue the Block and Accept voice commands to perform selective call screening. Beginning users who are granted this permission may unintentionally block calls when all they need is temporary use of the DND button. You should enable these commands for advanced users only. This permission does not affect the ability to block calls through the User Console. **Datatype:** boolean **Required:** No
Permissions.Record Utterances	Sets whether to grant permission to record utterances during Genie interactions. Use this permission for troubleshooting speech recognition problems. **Datatype:** boolean **Required:** No
Permissions.Monitor Users from Administration Console	Sets whether to grant permission to view information about logged-in group members and their badges in the Administration Console. This VAI permission is equivalent to the View Users And Groups permission in the Administration Console. **Datatype:** boolean **Required:** No
Permissions.Add/Edit/Delete Users	Sets whether to grant permission to maintain the Vocera database by adding, editing, and deleting all features in a user profile, such as alternate spoken names, group membership, and so forth. **Datatype:** boolean **Required:** No

Key	Description
Permissions.Add/Edit/Delete Temporary Users	Sets whether to grant permission to maintain the Vocera database by adding, editing, and deleting all features of the profiles of temporary users. **Datatype:** boolean **Required:** No
Permissions.Edit Users	Sets whether to grant permission to maintain the Vocera database by editing existing user profiles. **Datatype:** boolean **Required:** No
Permissions.Add/Edit/Delete Address Book Entries	Sets whether to grant permission to maintain the Vocera database by adding, editing, and deleting address book entries. Also grants permission to record a spoken name for address book entries. **Datatype:** boolean **Required:** No
Permissions.Access Genie from Phone Using Caller ID	Sets whether to grant permission to call the Vocera hunt number from a phone and access the Genie using a caller ID associated with the phone. The caller's ID is matched against a user's phone number in the Vocera database. **Datatype:** boolean **Required:** No **Since:** 4.1
Permissions.Perform System Device Management	Sets whether to grant permission to add, edit, and delete devices and view the Status Monitor. **Datatype:** boolean **Required:** No **Since:** 4.1
AntiPermissions	For the properties in this set, a value of `true` explicitly revokes that permission for the group even if members belong to another group that grants the permission. **Datatype:** KeyedPropertySet **Required:** No
AntiPermissions.Perform System Administration	Sets whether to revoke permission to perform system administration. When this permission is revoked, the group no longer has full administrative privileges in the Administration Console, and members are no longer granted every other permission. **Datatype:** boolean **Required:** No
AntiPermissions.Record Name Prompts for Another User	Sets whether to revoke permission to record name prompts for other users, as well as groups and address book entries. **Datatype:** boolean **Required:** No
AntiPermissions.Log In as Another User	Sets whether to revoke permission to log in as someone else, ignoring any voiceprint authentication. **Datatype:** boolean **Required:** No
AntiPermissions.Call Internal Numbers	Sets whether to revoke permission to place calls to internal telephone extensions by saying the key phrase "Dial extension" (for example, "Dial extension 4085"). This feature requires Telephony Integration. **Datatype:** boolean **Required:** No
AntiPermissions.Call Toll-Free Numbers	Sets whether to revoke permission to place calls to phone numbers in toll-free calling areas. This feature requires Telephony Integration. **Datatype:** boolean **Required:** No

Key	Description
AntiPermissions.Call Toll Numbers	Sets whether to revoke permission to place calls to phone numbers that are not in toll-free calling areas. This feature requires Telephony Integration. **Datatype:** boolean **Required:** No
AntiPermissions.Forward Calls to Badges	Sets whether to revoke permission to forward incoming calls to other badges. **Datatype:** boolean **Required:** No
AntiPermissions.Forward Calls to Internal Numbers	Sets whether to revoke permission to forward incoming calls to internal phone numbers. This feature requires Telephony Integration. **Datatype:** boolean **Required:** No
AntiPermissions.Forward Calls to Toll-Free Numbers	Sets whether to revoke permission to forward incoming calls to phone numbers in toll-free calling areas. This feature requires Telephony Integration. **Datatype:** boolean **Required:** No
AntiPermissions.Forward Calls to Toll Numbers	Sets whether to revoke permission to forward incoming calls to phone numbers that are not in toll-free calling areas. This feature requires Telephony Integration. **Datatype:** boolean **Required:** No
AntiPermissions.Initiate Broadcasts	Sets whether to revoke permission to broadcast to all users in a group at the same time. **Datatype:** boolean **Required:** No
AntiPermissions.Initiate Broadcasts to Everyone	Sets whether to revoke permission to broadcast to all users in the Everyone group for your site. **Datatype:** boolean **Required:** No
AntiPermissions.Initiate Urgent Broadcasts	Sets whether to revoke permission to broadcast an urgent call to every member in a group at the same time. **Datatype:** boolean **Required:** No
AntiPermissions.Place Urgent Calls	Sets whether to revoke permission to place an urgent call or initiate an urgent three-way conference call. **Datatype:** boolean **Required:** No
AntiPermissions.Call Users at Other Sites	Sets whether to revoke permission to contact a user whose home site or current site is different from the home site or current site of the caller. **Datatype:** boolean **Required:** No
AntiPermissions.Join Conference	Sets whether to revoke permission to enter or leave a conference. To prevent a user from conferencing, revoke the Join Conference permission and use the Administration Console to remove the user from a conference. **Datatype:** boolean **Required:** No
AntiPermissions.Send Messages To Everyone	Sets whether to revoke permission to send a message to all users in the Everyone group for your site. **Datatype:** boolean **Required:** No
AntiPermissions.Have Toll-Free Pager Number	Sets whether to revoke permission to have a pager number that is in a toll-free calling area. This feature requires Telephony Integration. **Datatype:** boolean **Required:** No

Key	Description
AntiPermissions.Have Toll Pager Number	Sets whether to revoke permission to have a pager number that is in a toll calling area. This feature requires Telephony Integration. **Datatype:** boolean **Required:** No
AntiPermissions.Require Authentication to Log In	Sets whether to revoke permission that would require group members to recite a series of random digits when they log in. If `true`, members can log in without authentication. **Datatype:** boolean **Required:** No
AntiPermissions.Require Authentication to Play Messages	Sets whether to revoke permission that would require group members to recite a series of random digits when they play messages. If the voice does not match the recorded voiceprint, users cannot play messages. If `true`, members can play messages without authentication. **Datatype:** boolean **Required:** No
AntiPermissions.Record your Voiceprint	Sets whether to revoke permission for group members to record their voiceprint. This permission is effective only if the Voice Prints Enabled system property is set to `true`. **Datatype:** boolean **Required:** No
AntiPermissions.Erase your Voiceprint	Sets whether to revoke permission for group members to erase their previously-recorded voiceprints. This permission is effective only if the Voice Prints Enabled system property is set to `true`. **Datatype:** boolean **Required:** No
AntiPermissions.Erase Voiceprint of Another User	Sets whether to revoke permission to erase the voiceprint of another user. This permission is effective only if the Voice Prints Enabled system property is set to `true`. **Datatype:** boolean **Required:** No
AntiPermissions.Locate Users or Group Members	Sets whether to revoke permission to locate other users or group members. This feature is useful only if location names have been defined and access points have been assigned to locations. **Datatype:** boolean **Required:** No
AntiPermissions.Have VIP Status	Sets whether to revoke permission to complete a call even when users are blocking calls or have placed their badges in Do Not Disturb mode. **Datatype:** boolean **Required:** No
AntiPermissions.Block and Accept Calls	Sets whether to revoke permission to issue the Block and Accept voice commands to perform selective call screening. This permission does not affect the ability to block calls through the User Console. **Datatype:** boolean **Required:** No
AntiPermissions.Record Utterances	Sets whether to revoke permission to record utterances during Genie interactions. **Datatype:** boolean **Required:** No
AntiPermissions.Monitor Users from Administration Console	Sets whether to revoke the permission to view information about logged-in group members and their badges in the Administration Console. This VAI permission is equivalent to the View Users And Groups permission in the Administration Console. **Datatype:** boolean **Required:** No

Key	Description
AntiPermissions.Add/Edit/Delete Users	Sets whether to revoke permission to maintain the Vocera database by adding, editing, and deleting all features in a user profile, such as alternate spoken names, group membership, and so forth. **Datatype:** boolean **Required:** No
AntiPermissions.Add/Edit/Delete Temporary Users	Sets whether to revoke permission to maintain the Vocera database by adding, editing, and deleting all features of the profiles of temporary users. **Datatype:** boolean **Required:** No
AntiPermissions.Edit Users	Sets whether to revoke permission to maintain the Vocera database by editing existing user profiles. **Datatype:** boolean **Required:** No
AntiPermissions.Add/Edit/Delete Address Book Entries	Sets whether to revoke permission to maintain the Vocera database by adding, editing, and deleting address book entries. Also revokes permission to record a spoken name for address book entries. **Datatype:** boolean **Required:** No
AntiPermissions.Access Genie from Phone Using Caller ID	Sets whether to revoke permission to call the Vocera hunt number from a phone and access the Genie using a caller ID associated with the phone. **Datatype:** boolean **Required:** No **Since:** 4.1
AntiPermissions.Perform System Device Management	Sets whether to revoke permission to add, edit, and delete devices and view the Status Monitor. **Datatype:** boolean **Required:** No **Since:** 4.1

Location Properties

The following table lists the properties of a Location. Locations are names of places to which you assign one or more access points.

Since: 4.0

Table 20: Location properties

Key	Description
Name	Location name. The name must start with a letter or digit. It must contain only letters, digits, spaces, apostrophes ('), underscores (_), or dashes (-). No other characters are allowed. **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
Spoken Name	Alternate spoken name for the location, if needed. The alternate spoken name gives the server an additional field to check, increasing the chances that a location name will be understood by the Genie. **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Description	Description of the location. Example: `H Q Lobby` **Datatype:** String **Maximum Length:** 100 characters **Required:** No

Key	Description
Site	Physical site of the access point. If your organization has multiple sites connected to the same Vocera server, specify the site that represents the access point's physical location. If your organization does not have multiple sites, accept the default Global site. **Datatype:** Site object, or a string representing the site's internal name **Required:** No
Access Points	Property set containing access points associated with the location. **Datatype:** IndexedPropertySet **Required:** No
Access Points.*	Represents each access point assigned to the location. To enter an access point, type its MAC address (12 hexadecimal characters). **Datatype:** String **Maximum Length:** 12 characters **Required:** No
Neighbors	Property set containing neighboring locations. **Datatype:** IndexedPropertySet **Required:** No
Neighbors.*	Represents each neighboring location. **Datatype:** Location object, or the internal name of the location of a neighboring access point **Required:** No

Site Properties

The following table lists the properties of a Site. In Vocera, a site is a distinct physical location that shares a centralized Vocera server with one or more other physical locations.

Since: 4.0

Table 21: Site properties

Key	Description
Name	Name of the site. The name must start with a letter or digit. It must contain only letters, digits, spaces, apostrophes ('), underscores (_), or dashes (-). No other characters are allowed. **Note:** If you change the name of a site that has a Telephony server associated with it, you must set the value of the VOCERA_SITE environment variable on the Telephony server machine to the name of the new site. **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
Spoken Name	Alternate spoken name of the site. For example, if users commonly refer to a site by a nickname or an acronym, enter that variation here. **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Description	Description of the site. Example: Intensive Care Unit **Datatype:** String **Maximum Length:** 100 characters **Required:** No

Key	Description
Cost Center	The site's cost center ID, which enables Vocera to track system usage by site and potentially allows an organization to charge sites for relative usage. **Datatype:** String **Maximum Length:** 100 characters **Required:** No
Time Zone	The site's time zone. Enter a Windows Time Zone string (such as "Etc/GMT+8", "America/Los_Angeles", and "PST8PDT"), or "" (empty string) to use the time zone of the Vocera server. **Datatype:** String **Required:** No
Panic Group	The group that receives emergency broadcasts for this site. **Datatype:** Group **Required:** No **Since:** 4.3
Inhibit Panic Chime	If `true`, emergency broadcasts are started without an opening chime. **Datatype:** boolean **Required:** No **Since:** 4.3
Telephony Info	Property set containing telephony information for this site. **Datatype:** KeyedPropertySet **Required:** No
Telephony Info.Telephony Enabled	If `true`, telephony features are enabled for the site. **Datatype:** boolean **Required:** No
Telephony Info.Telephony Interface Type	Type of telephony interface. Enter `IP`, `Digital`, or `Analog`. **Datatype:** String **Required:** No
Telephony Info.Telephony # of Lines	Number of telephone lines. **Datatype:** int **Required:** Yes (if telephony is enabled for the site)
Telephony Info.Telephony Protocol	Signaling protocol that your PBX uses at the network layer. For IP PBX integration, enter the following value: `SIP Version 2.0`. For Digital PBX integration, enter one of the following values: `ISDN PRI`, `EURO ISDN PRI`, or `Wink Start`. DO NOT update this property if Telephony Interface Type = `Analog`. **Datatype:** String **Required:** No
Telephony Info.Telephony ISDN Protocol	ISDN protocol used by your PBX. Enter one of the following values: `NI2`, `DMS`, `5ESS`, `4ESS`, `NT1`, `CTR4`, `QTE`, `NE1`, or `QNT`. **Datatype:** String **Required:** No
Telephony Info.Telephony Framing	Framing that your PBX uses at the physical layer. Enter one of the following values: `ESF`, `D4`, or `CEPT1`. Update this property only if Telephony Interface Type = `Digital`. **Datatype:** String **Required:** No
Telephony Info.Telephony Line Code	Line code that your PBX uses at the physical layer. Enter one of the following values: `B8ZS`, `AMI`, or `HDB3`. Update this property only if Telephony Interface Type = `Digital`. **Datatype:** String **Required:** No

Key	Description
Telephony Info.Area Code	Area code of the region in which the Vocera server is installed. **Datatype:** String **Maximum Length:** 10 characters **Required:** Yes (if telephony is enabled for the site)
Telephony Info.Local Access	Sequence of numbers you use to get an outside line. For example, a PBX might require you to dial a 0 or a 9 or an 8 to get an outside line. By default, Vocera prepends this access code to any number within the local area code. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Telephony Info.Long Distance Access	Sequence of numbers you enter before placing a long distance call. For example, a PBX system might require you to dial a 9 to get an outside line and then dial a 1 before a long-distance telephone number. In this situation, enter `91`. By default, Vocera prepends this access code to any number that includes an area code that is not the local area code. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Telephony Info.System Phone Number	Area code and phone number of the DID line or hunt group you set up for the Vocera system. To use this number with numeric pagers, enter an asterisk after the last digit of the phone number. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Telephony Info.Direct Access Phone Number	Area code and phone number of the DID line you set up for specially licensed user access to the Vocera system. If you have not obtained Vocera Access Anywhere user licenses or you are not using ISDN or SIP signaling protocol, this property should not be updated. **Datatype:** String **Maximum Length:** 75 characters **Required:** No **Since:** 4.1
Telephony Info.Voice Mail Access	Sequence of numbers you enter to access the company's voice mail system. A typical entry includes X, then the sequence of digits that you dial to get into the voicemail system from an internal phone, and possibly special dialing characters such as the * or # to indicate the end of the sequence. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Telephony Info.Seven Digit Dialing	If `true`, the area code is omitted from the dialing sequence for a local call. **Datatype:** boolean **Required:** No
Telephony Info.PIN Setup	Template for adding a PIN to a dialing sequence for long distance calls. A PIN template can include digits, special characters, and PIN macros. **Datatype:** String **Maximum Length:** 75 characters **Required:** No

Key	Description
Telephony Info.Default PIN	The default PIN for long distance calls for the site. If a telephony PIN is not specified in the user's profile and the user does not belong to a department group that has a PIN, then the site PIN is used. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Telephony Info.Telephony Extension Length	Specify the number of digits in an extension, or 0 (zero) to allow variable length extensions. **Datatype:** int **Required:** No
Telephony Info.Access Code Info	By default, numbers in the local area code use the Default Local Access Code and all others use the Default Long-Distance Access Code. This property set contains telephone numbers that are exceptions to the access code policy. Each member of the indexed set is itself a property set. **Datatype:** IndexedPropertySet **Required:** No
Telephony Info.Access Code Info.*	Represents each defined access code exception in the property set. **Datatype:** KeyedPropertySet **Required:** No
Telephony Info.Access Code Info.*.Number Range	KeyedPropertySet that defines the number range for an access code exception. **Datatype:** KeyedPropertySet **Required:** No
Telephony Info.Access Code Info.*.Number Range.Area Code	Area code for which the exception is defined. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Telephony Info.Access Code Info.*.Number Range.Match Type	Type of number range to match. Enter one of the following values: `All`, `Starts With`, or `Range`. **Datatype:** String **Required:** No
Telephony Info.Access Code Info.*.Number Range.Starts With	Sequence of characters to match. Used when Match Type = `Starts With`. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Telephony Info.Access Code Info.*.Number Range.From	Start of range to match. Used when Match Type = `Range`. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Telephony Info.Access Code Info.*.Number Range.To	End of range to match. Used when Match Type = `Range`. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Telephony Info.Access Code Info.*.Access Code	Access code that the specified area code requires. **Datatype:** String **Maximum Length:** 10 characters **Required:** Yes (for each access code exception)
Telephony Info.Toll Info	By default, numbers in the local area code are considered toll-free, and others are considered to require toll-call permissions. This property set contains telephone numbers that are exceptions to the toll-call policy. **Datatype:** IndexedPropertySet **Required:** No

Key	Description
Telephony Info.Toll Info.*	Represents each defined toll info exception. Each keyed set specifies an area code and phone numbers that can be called even by users who do not have toll-call permissions granted. **Datatype:** KeyedPropertySet **Required:** No
Telephony Info.Toll Info.*.Number Range	Property set that defines the number range for a toll info exception. **Datatype:** KeyedPropertySet **Required:** No
Telephony Info.Toll Info.*.Number Range.Area Code	Area code for which the exception is defined. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Telephony Info.Toll Info.*.Number Range.Match Type	Type of number range to match. Enter one of the following values: `All`, `Starts With`, or `Range`. **Datatype:** String **Required:** No
Telephony Info.Toll Info.*.Number Range.Starts With	Sequence of characters to match. Used when Match Type = `Starts With`. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Telephony Info.Toll Info.*.Number Range.From	Start of range to match. Used when Match Type = `Range`. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Telephony Info.Toll Info.*.Number Range.To	End of range to match. Used when Match Type = `Range`. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Telephony Info.Toll Info.*.Toll Free	If `true`, the area code is toll free. **Datatype:** boolean **Required:** No
Telephony Info.Paging Info	Specifies properties for interacting with pagers. **Datatype:** KeyedPropertySet **Required:** No
Telephony Info.Paging Info.Pager Number Length	Specify the number of digits in a pager number, or 0 (zero) to allow variable length numbers. **Datatype:** int **Required:** No
Telephony Info.Paging Info.Outside Page Setup	Template that determines how Vocera formats the string passed to a pager outside the Vocera system. The default value of this property is %N;%V%D. For more information, see the *Vocera Telephony Configuration Guide*. **Datatype:** String **Required:** No
Telephony Info.Paging Info.Inside Page Setup	Template that determines how Vocera formats the string passed to a pager inside the Vocera system. The default value of this property is %N;%V%D. For more information, see the *Vocera Telephony Configuration Guide*. **Datatype:** String **Required:** No
Telephony Info.Paging Info.Outside Page Setup for DialIn	Template that determines how Vocera formats the string passed to an outside pager by a person calling into the Vocera hunt group or DID number. The default value of this property is %N;%X. For more information, see the *Vocera Telephony Configuration Guide*. **Datatype:** String **Required:** No

Key	Description
Telephony Info.Paging Info.Inside Page Setup for DialIn	Template that determines how Vocera formats the string passed to an inside pager by a person calling into the Vocera hunt group or DID number. The default value of this property is %N;%X. For more information, see the *Vocera Telephony Configuration Guide*. **Datatype:** String **Required:** No
Telephony Info.DID Info	Property set containing direct inward dialing (DID) information. **Datatype:** IndexedPropertySet **Required:** No
Telephony Info.DID Info.*	Represents each defined range of direct inward dialing (DID) numbers. Each keyed set specifies a prefix and the range of phone numbers to use for direct inward dialing. **Datatype:** KeyedPropertySet **Required:** No
Telephony Info.DID Info.*.Number Range	Property set that defines a number range to use for direct inward dialing. **Datatype:** KeyedPropertySet **Required:** No
Telephony Info.DID Info.*.Number Range.Match Type	Type of number range to match. Enter one of the following values: `All`, `Starts With`, or `Range`. **Datatype:** String **Required:** No
Telephony Info.DID Info.*.Number Range.Starts With	Sequence of characters to match. Used when Match Type = `Starts With`. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Telephony Info.DID Info.*.Number Range.From	Start of range to match. Used when Match Type = `Range`. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Telephony Info.DID Info.*.Number Range.To	End of range to match. Used when Match Type = `Range`. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Telephony Info.DID Info.*.Prefix	Area code and prefix assigned to the range. For example, if the local area code of the PBX is 408, and the corporate prefix for all extensions is 790, you typically enter `(408)-790`. In some situations, your PBX administrator may assign a different prefix for you to use. **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
Telephony Info.Dynamic Phone Info	Property set that specifies a range of dynamic phone extensions. This allows you to configure Vocera to supply telephone extensions on demand to users who need them. **Datatype:** KeyedPropertySet **Required:** No
Telephony Info.Dynamic Phone Info.Enabled	If `true`, dynamic extensions are enabled. **Datatype:** boolean **Required:** No
Telephony Info.Dynamic Phone Info.First	Specifies the first dynamic extension in the range. **Datatype:** String **Maximum Length:** 7 characters **Required:** No

Key	Description
Telephony Info.Dynamic Phone Info.Last	Specifies the last dynamic extension in the range. The Last value must be greater than the First value. **Datatype:** String **Maximum Length:** 7 characters **Required:** No
Telephony Info.Dynamic Phone Info.Lifetime	Specifies the lifetime, in days or hours, of the assignment of dynamic extensions. Enter 0 (zero) to make the extensions permanent. **Datatype:** int **Required:** No
Telephony Info.Dynamic Phone Info.Hours	Specifies whether the lifetime of dynamic extensions is measured in hours (`true`) or days (`false`). **Datatype:** boolean **Required:** No **Since:** 4.1
Telephony Info.Dynamic Phone Info.Last Allocated	The last allocated dynamic extension. **Cannot be updated.** **Datatype:** String
Telephony Info.Shared Server Info	Property set that contains information needed to allow multiple sites to share a Telephony server. **Datatype:** IndexedPropertySet **Required:** No
Telephony Info.Shared Server Info.*	Represents each defined site that shares this Telephony server. Each keyed set specifies the site, hunt group number, and reserved range of lines for incoming calls. **Datatype:** KeyedPropertySet **Required:** No
Telephony Info.Shared Server Info.*.Site	Principal site for which Telephony is enabled. **Datatype:** Site object, or a string representing the site's internal name. **Required:** No
Telephony Info.Shared Server Info.*.System Phone Number	Area code and phone number of the DID line or hunt group you set up for the Vocera system. **Datatype:** String **Maximum Length:** 75 characters
Telephony Info.Shared Server Info.*.First Reserved Line No	First of the reserved lines for incoming calls. **Datatype:** String
Telephony Info.Shared Server Info.*.Reserved Line Count	Number of reserved lines for incoming calls. **Datatype:** int
Telephony Info.Shared Server Info.*.Extension Prefix	Prefix of the dial string used to place calls through the tie line to the selected site that is sharing the principal's telephony server. Alternatively, this field could also be used to specify a prefix for Direct Inward Dialing (DID) numbers at the selected site. **Datatype:** String **Since:** 4.1
Telephony Info.Call Signaling Address	The IP address of your IP PBX or VoIP gateway. By default, port 5060 is used. If you need to change the port, enter the call signaling address in the form *IP_Address:Port*. **Datatype:** String **Maximum Length:** 75 characters **Since:** 4.3
Telephony Info.Cisco Info	Property set containing default Cisco integration properties. **Datatype:** KeyedPropertySet **Required:** No **Since:** 4.3
Telephony Info.Cisco Info.Enabled	**Datatype:** boolen **Required:** No **Since:** 4.3

Key	Description
Telephony Info.Cisco Info.Extension Mobility Enabled	**Datatype:** boolean **Required:** No **Since:** 4.3
Telephony Info.Cisco Info.Phone	The voice access number for CUCM. This number should match the route pattern/number for the Vocera SIP trunk. You can find route patterns in CUCM Console by choosing Call Routing > Route/Hunt > Route Pattern. **Datatype:** String **Maximum Length:** 75 characters **Required:** No **Since:** 4.3
Telephony Info.Cisco Info.First Line	The first phone line used for the internal range of Vocera lines. **Datatype:** String **Maximum Length:** 7 characters **Required:** No **Since:** 4.3
Telephony Info.Cisco Info.Last Line	The last phone line used for the internal range of Vocera lines. **Datatype:** String **Maximum Length:** 7 characters **Required:** No **Since:** 4.3
Telephony Info.Cisco Info.IP Address	The IP address of the CUCM in dotted-decimal notation (for example, 192.168.15.10). **Datatype:** String **Maximum Length:** 50 characters **Required:** No **Since:** 4.3
Telephony Info.Cisco Info.User Name	The Vocera application user ID for CUCM. **Datatype:** String **Maximum Length:** 50 characters **Required:** No **Since:** 4.3
Telephony Info.Cisco Info.Password	The Vocera application user ID for CUCM. **Use only for update.** **Important:** Your application should restrict passwords to be between 5 and 15 characters. Otherwise, passwords that you set in your VAI application may not be valid for the Vocera Administration Console and User Console. VAI itself does not restrict the length of passwords. **Datatype:** String **Required:** No **Since:** 4.3

User Properties

The following table lists the properties of a User.

Since: 4.0

Table 22: User properties

Key	Description
User ID	Vocera user ID. Enter an ID that is not already assigned to another user on the system, being careful to choose a name that you and the user can easily remember. The user ID is not case-sensitive. The User ID must start with a letter or digit. It must contain only letters, digits, spaces, periods ('), underscores (_), or dashes (-). No other characters are allowed. **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
Password	The user's Vocera password. The password is case-sensitive. Use this property to create or update the user's password. **Important:** Your application should restrict passwords to be between 5 and 15 characters. Otherwise, passwords that you set in your VAI application may not be valid for the Vocera Administrator Console and User Console. VAI itself does not restrict the length of passwords. **Datatype:** String **Required:** No
Phone Password	Password used to authenticate the user when accessing the Genie from a phone. **Important:** Your application should restrict the phone password to be between 5 and 15 characters consisting of letters or numbers. Special characters are not allowed. VAI itself does not restrict the length of passwords or prevent you from entering a password with invalid characters. **Datatype:** String **Required:** No **Since:** 4.1
Last Name	The user's last name. The name must start with a letter or digit. It must contain only letters, digits, spaces, apostrophes ('), underscores (_), or dashes (-). No other characters are allowed. **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
First Name	The user's first name. The name must start with a letter or digit. It must contain only letters, digits, spaces, apostrophes ('), underscores (_), or dashes (-). No other characters are allowed. **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
Alt Spoken Names	Property set containing up to three variations of the spoken name of the user. **Datatype:** IndexedPropertySet **Required:** No
Alt Spoken Names.*	Represents each Alternate Spoken Name in the property set. **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Ident Phrase	An identifying phrase that distinguishes this user from others with the same name. Example: `Rita Clark in Staffing` **Datatype:** String **Maximum Length:** 100 characters **Required:** No

Key	Description
Email Address	An optional email address, which allows users to send voice messages as an email attachment. Example: `jdoe@vocera.com` **Datatype:** String **Maximum Length:** 40 characters **Required:** No
Desk Phone	The desk phone number or extension for the user. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Cell Phone	The user's cell phone number. You can enter digits, special dialing characters, or special dialing macros. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Home Phone	The user's home phone number. You can enter digits, special dialing characters, or special dialing macros. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Pager Phone	The user's pager number. You can enter digits, special dialing characters, or special dialing macros. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Dynamic Phone	The user's dynamically assigned phone extension. Cannot be updated. **Datatype:** String
Vocera Phone	The user's Vocera phone extension. **Datatype:** String **Maximum Length:** 75 characters **Required:** No **Since:** 4.1 SP4
Conference Group	The current conference group for the user. A user can have only one conference group at a time. You must specify a valid group name. **Datatype:** String
Employee ID	Optional unique value that identifies a Vocera user. **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Cost Center	The user's cost center ID, which enables Vocera to track system usage by site and potentially allows an organization to charge sites for relative usage. **Datatype:** String **Maximum Length:** 100 characters **Required:** No
PIN	Specify a value of the PIN for long distance calls. A telephony PIN allows an organization to authorize or account for telephone usage and to distribute telephone costs among different users, departments, or sites. A PIN can include digits, special characters, and PIN macros. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Site	The user's home site. If you don't specify a site, the Global site is used. **Datatype:** Site object, or a string representing the site's internal name. **Required:** No

Key	Description
Expire Time	If the user is a temporary user, this property specifies when the profile expires. Enter a date with the format mm/dd/yyyy. The date must be later than the current date. **Datatype:** String **Required:** No
Call Blocking	Defines the default call blocking behavior for any users or groups not specified in the Block List or Accept List. Enter one of the following values: `Accept` or `Block`. **Datatype:** String **Required:** No
Block List	Property set containing users or groups whose calls are blocked. **Datatype:** IndexedPropertySet **Required:** No
Block List.*	Represents each user or group whose calls are blocked. **Datatype:** Entity (User or Group) **Required:** No
Accept List	Property set containing users or groups whose calls are accepted (not blocked). **Datatype:** IndexedPropertySet **Required:** No
Accept List.*	Represents each user or group whose calls are accepted (not blocked). **Datatype:** Entity (User or Group) **Required:** No
Buddies	Property set containing the user's buddies. Each member of the indexed set is itself a property set. **Datatype:** IndexedPropertySet **Required:** No
Buddies.*	Represents each defined buddy (personal contact) in the property set. **Datatype:** KeyedPropertySet **Required:** No
Buddies.*.Name	Contact, User, or Group object that identifies the buddy. **Datatype:** Entity object (Contact, User, or Group) **Required:** Yes
Buddies.*.Nick Name	Name used to call the buddy. The name must start with a letter or digit. It must contain only letters, digits, spaces, apostrophes ('), underscores (_), or dashes (-). No other characters are allowed. **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
Buddies.*.VIP	If `true`, the buddy has VIP status and can call the user even when the user is blocking calls or in DND mode. **Datatype:** boolean **Required:** No
Buddies.*.RingTone	One of the available ring tones, for example, `Ring-Tone-01`, `Ring-Tone-02`, and so on. When the buddy calls the user, the specified ring tone is used. **Datatype:** String **Required:** No
Verbal Call Announcement	If `true`, the caller's name will be announced after the ring tone. **Datatype:** boolean **Required:** No
Verbal Genie Greeting	If `true`, the user will hear a spoken greeting ("Vocera") after pressing the call button. **Datatype:** boolean **Required:** No

Key	Description
Tonal Genie Greeting	If `true`, the user will hear a short tone after pressing the call button. **Datatype:** boolean **Required:** No
Auto Answer	If `true`, incoming calls are connected immediately, without asking the user whether he wants to take the call. **Datatype:** boolean **Required:** No
Auto Who Called	If `true`, the user can press the Call button on the badge to play an announcement of the names of callers who unsuccessfully tried to call since the last time the user pressed the Call button, and who left messages. **Datatype:** boolean **Required:** No
Out Of Range Alert	If `true`, the user will hear a warning tone when the badge moves out of the range of the wireless network. **Datatype:** boolean **Required:** No
Low Battery Alert	If `true`, the badge will warn the user whenever the battery needs to be recharged. **Datatype:** boolean **Required:** No
Auto Logout	If `true`, the user will be automatically logged out and the badge will be turned off when the badge is placed in a charger. **Datatype:** boolean **Required:** No
VMessage Alert	If `true`, the user will hear an alert tone when he receives a new voice message. **Datatype:** boolean **Required:** No
TMessage Alert	If `true`, the user will hear an alert tone when he receives a new text message. **Datatype:** boolean **Required:** No
Disable Alerts In DND	If `true`, all alerts are suppressed when the user's badge is in Do Not Disturb (DND) mode. **Datatype:** boolean **Required:** No
Play Older Messages First	If `true`, messages are played in the order in which they are received. Otherwise, messages are played in reverse order (newest first). **Datatype:** boolean **Required:** No
Timestamp Played Messages	If `true`, the user will hear the date and time each message was sent when he plays messages. **Datatype:** boolean **Required:** No
Fast Call Setup	If `true`, a call is connected as soon as the recipient accepts it. Otherwise, the Genie always completes the call announcement before connecting the call. **Datatype:** boolean **Required:** No
VMessage Reminder	If `true`, the user will hear a tone every 10 minutes until he retrieves new voice messages. **Datatype:** boolean **Required:** No

Key	Description
TMessage Reminder	If `true`, the user will hear a tone every 10 minutes until he retrieves new text messages. **Datatype:** boolean **Required:** No
DND Reminder	If `true`, the user will hear a tone every 10 minutes while his badge is in Do Not Disturb (DND) mode. **Datatype:** boolean **Required:** No
Enable Pages	If `true`, the user can receive numeric pages. Otherwise, pages are disabled. **Datatype:** boolean **Required:** No
Announce Through Speaker	If `true`, Vocera plays incoming call and message announcements through the badge speaker when a headset is plugged into the user's badge. Otherwise, both announcements and actual calls or messages are played through the headset. **Datatype:** boolean **Required:** No
Press Button To Accept Call	If `true`, the user is required to accept or reject incoming calls by pressing the Call or DND/Hold button. The user cannot say "Yes" and "No" voice commands to accept and reject incoming calls. This feature is useful in certain high-noise environments. **Datatype:** boolean **Required:** No **Since:** 4.1
Announce Group Calls	If `true`, when the user receives a call made to a group, the Genie will identify the group that was called. **Datatype:** boolean **Required:** No **Since:** 4.1
Block Voice Messages	If `true`, Vocera suppresses notifications when the user receives a message. **Datatype:** boolean **Required:** No **Since:** 4.1
Ring Tone	One of the available ring tones, for example, `Ring-Tone-01`, `Ring-Tone-02`, and so on. When the user receives a call on his badge, the specified ring tone is used. **Datatype:** String **Required:** No
Genie Persona	One of the available Genie names, for example, `Mark` or `Jean`. The Genie is the voice that prompts users when they interact with the Vocera system. **Datatype:** String **Required:** No
Forwarding	Sets whether and where incoming calls are forwarded. Enter one of the following values: `None`, `Desk Phone`, `Cell Phone`, `Home Phone`, `Voice Mail`, `Other Phone`, or `Other User`. **Datatype:** String **Required:** No
Forwarding Number	Phone number used when Forwarding = `Other Phone`. **Datatype:** String **Maximum Length:** 30 characters **Required:** No
Forwarding Name	User or Group to forward to when Forwarding = `Other User`. **Datatype:** User or Group **Required:** No

Key	Description
Forwarding When	Specify when to forward calls. Enter one of the following values: `Never`, `All`, `Unanswered`, or `Offline`. **Datatype:** String **Required:** No

System Properties

Vocera system properties are accessed from the `VAI` class. Use the `VAI` methods `getSystemProperties()` and `updateSystemProperties()`, respectively, to read and write these properties.

Since: 4.0

Table 23: System properties

Key	Description
Product Major Version	Vocera major version number. Example: given a product version of 3.1, the product major version is 3. **Cannot be updated.** **Datatype:** String
Product Minor Version	Vocera minor version number. Example: given a product version of 3.1, the product minor version is 1. **Cannot be updated.** **Datatype:** String
Product Revision	Vocera revision. Example: given a product version of 3.1SP1, the product revision is SP1. Given a product version of 3.1, the product revision is 0. **Cannot be updated.** **Datatype:** String
Time Last Update	Time in milliseconds since 1/1/70 00:00 GMT of the last update of a Vocera system property. **Cannot be updated.** **Datatype:** long
Self Register	If `true`, users can add themselves to the Vocera system through the User Console. **Datatype:** boolean **Required:** No
Badge Log In	If `true`, voice commands that enable users to log into and log out of badges are enabled. Otherwise, users cannot share badges, and you must specify each user's Badge ID. **Datatype:** boolean **Required:** No
Voice Prints Enabled	If `true`, the voiceprints feature is enabled to provide more secure authentication when users log in or check messages. **Datatype:** boolean **Required:** No
Auto Record Voice Prints	If `true`, the Vocera server automatically prompts users to record their voiceprints the next time they log in. Users are prompted only if they have not yet recorded a voiceprint. **Datatype:** boolean **Required:** No
Admin Password	Password for the Administrator user. Used only for update. **Important:** Your application should restrict passwords to be between 5 and 15 characters. Otherwise, passwords that you set in your VAI application may not be valid for the Vocera Administrator Console and User Console. VAI itself does not restrict the length of passwords. **Datatype:** String **Required:** Yes

Key	Description
Dictation Enabled	If `true`, dictation features are enabled for the Vocera system. Dictation feaures require a special license. Separate configuration is also required. **Datatype:** boolean **Required:** No **Since:** 4.1
Block all VMI Messages in DND	If `true`, blocks all VMI messages—even urgent messages—for users in Do Not Disturb mode. **Datatype:** boolean **Required:** No **Since:** 4.3
Block non-urgent VMI Messages in DND	If `true`, blocks non-urgent VMI messages for users in Do Not Disturb mode. **Datatype:** boolean **Required:** No **Since:** 4.3
VMP Enabled	If `true`, Vocera Messaging Platform (VMP) integration is enabled. **Datatype:** boolean **Required:** No **Since:** 4.3
Company	Name of your company or organization. This value appears in reports and logs. **Datatype:** String **Maximum Length:** 100 characters **Required:** No
Days To Keep Messages	Number of days to retain messages on the system. The default is 7 days (one week). **Datatype:** int **Required:** No
Time To Sweep	Time of day, in milliseconds from midnight, when messages are deleted from the Vocera Server. **Datatype:** long **Required:** No
Locale	Identifies the server's locale. Examples: `AU`, `CA`, `GB`, `NZ`, and `US`. **Datatype:** String **Required:** No
Override Info	Property set that specifies which system settings override the corresponding property in the Vocera User Console. A value of true indicates an override. The OverrideInfo.Override Opt list specifies the actual property values. **Datatype:** KeyedPropertySet **Required:** No
OverrideInfo.Verbal Call Announcement	If `true`, override each user's Verbal Call Announcement property. **Datatype:** boolean **Required:** No
OverrideInfo.Verbal Genie Greeting	If `true`, override each user's Verbal Genie Greeting property. **Datatype:** boolean **Required:** No
OverrideInfo.Tonal Genie Greeting	If `true`, override each user's Tonal Genie Greeting property. **Datatype:** boolean **Required:** No
OverrideInfo.Auto Answer	If `true`, override each user's Auto Answer property. **Datatype:** boolean **Required:** No

Key	Description
OverrideInfo.Auto Who Called	If `true`, override each user's Auto Who Called property. **Datatype:** boolean **Required:** No
OverrideInfo.Out Of Range Alert	If `true`, override each user's Out Of Range Alert property. **Datatype:** boolean **Required:** No
OverrideInfo.Low Battery Alert	If `true`, override each user's Low Battery Alert property. **Datatype:** boolean **Required:** No
OverrideInfo.Auto Logout	If `true`, override each user's Auto Logout property. **Datatype:** boolean **Required:** No
OverrideInfo.AP Tour	If `true`, override each user's AP Tour property. **Datatype:** boolean **Required:** No
OverrideInfo.VMessage Alert	If `true`, override each user's VMessage Alert property. **Datatype:** boolean **Required:** No
OverrideInfo.TMessage Alert	If `true`, override each user's TMessage Alert property. **Datatype:** boolean **Required:** No
OverrideInfo.Disable Alerts In DND	If `true`, override each user's Disable Alerts In DND property. **Datatype:** boolean **Required:** No
OverrideInfo.Play Older Messages First	If `true`, override each user's Play Older Messages First property. **Datatype:** boolean **Required:** No
OverrideInfo.Timestamp Played Messages	If `true`, override each user's Timestamp Played Messages property. **Datatype:** boolean **Required:** No
OverrideInfo.Fast Call Setup	If `true`, override each user's Fast Call Setup property. **Datatype:** boolean **Required:** No
OverrideInfo.VMessage Reminder	If `true`, override each user's VMessage Reminder property. **Datatype:** boolean **Required:** No
OverrideInfo.TMessage Reminder	If `true`, override each user's TMessage Reminder property. **Datatype:** boolean **Required:** No
OverrideInfo.DND Reminder	If `true`, override each user's DND Reminder property. **Datatype:** boolean **Required:** No
OverrideInfo.Enable Pages	If `true`, override each user's Enable Pages property. **Datatype:** boolean **Required:** No
OverrideInfo.Announce Through Speaker	If `true`, override each user's Announce Through Speaker property. **Datatype:** boolean **Required:** No
OverrideInfo.Press Button To Accept Call	If `true`, override each user's Press Button To Accept Call property. **Datatype:** boolean **Required:** No **Since:** 4.1

Key	Description
OverrideInfo.Announce Group Calls	If `true`, override each user's Announce Group Calls property. **Datatype:** boolean **Required:** No **Since:** 4.1
OverrideInfo.Block Voice Messages	If `true`, override each user's Block Voice Messages property. **Datatype:** boolean **Required:** No
OverrideInfo.Enable Genie Access From Phone	If `true`, override each user's Enable Genie Access From Phone property. **Datatype:** boolean **Required:** No **Since:** 4.1
OverrideInfo.Ring Tone	If `true`, override each user's Ring Tone property. **Datatype:** boolean **Required:** No
OverrideInfo.Genie Persona	If `true`, override each user's Genie Persona property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Verbal Call Announcement	Specifies the value of the overridden Verbal Call Announcement property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Verbal Genie Greeting	Specifies the value of the overridden Verbal Genie Greeting property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Tonal Genie Greeting	Specifies the value of the overridden Tonal Genie Greeting property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Auto Answer	Specifies the value of the overridden Auto Answer property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Auto Who Called	Specifies the value of the overridden Auto Who Called property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Out Of Range Alert	Specifies the value of the overridden Out Of Range Alert property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Low Battery Alert	Specifies the value of the overridden Low Battery Alert property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Auto Logout	Specifies the value of the overridden Auto Logout property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt AP Tour	Specifies the value of the overridden AP Tour property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt VMessage Alert	Specifies the value of the overridden VMessage Alert property. **Datatype:** boolean **Required:** No

Key	Description
OverrideInfo.Override Opt TMessage Alert	Specifies the value of the overridden TMessage Alert property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Disable Alerts In DND	Specifies the value of the overridden Disable Alerts In DND property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Play Older Messages First	Specifies the value of the overridden Play Older Messages First property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Timestamp Played Messages	Specifies the value of the overridden Timestamp Played Messages property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Fast Call Setup	Specifies the value of the overridden Fast Call Setup property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt VMessage Reminder	Specifies the value of the overridden VMessage Reminder property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt TMessage Reminder	Specifies the value of the overridden TMessage Reminder property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt DND Reminder	Specifies the value of the overridden DND Reminder property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Enable Pages	Specifies the value of the overridden Enable Pages property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Announce Through Speaker	Specifies the value of the overridden Announce Through Speaker property. **Datatype:** boolean **Required:** No
OverrideInfo.Override Opt Press Button To Accept Call	Specifies the value of the overridden Press Button To Accept Call property. **Datatype:** boolean **Required:** No **Since:** 4.1
OverrideInfo.Override Opt Announce Group Calls	Specifies the value of the overridden Announce Group Calls property. **Datatype:** boolean **Required:** No **Since:** 4.1
OverrideInfo.Override Opt Block Voice Messages	Specifies the value of the overridden Block Voice Messages property. **Datatype:** boolean **Required:** No **Since:** 4.1
OverrideInfo.Override Opt Enable Genie Access From Phone	Specifies the value of the overridden Enable Genie Access From Phone property. **Datatype:** boolean **Required:** No **Since:** 4.1

Key	Description
OverrideInfo.Override Opt Ring Tone	Specifies the value of the overridden Ring Tone property. Must be one of the available ring tones, for example, `Ring-Tone-01`, `Ring-Tone-02`, and so on. When the user receives a call on his badge, the specified ring tone is used. **Datatype:** String **Required:** No
OverrideInfo.Override Opt Genie Persona	Specifies the value of the overridden Genie Persona property. Must be one of the available Genie names, for example, `Mark` or `Jean`. The Genie is the voice that prompts users when they interact with the Vocera system. **Datatype:** String **Required:** No
Mail Info	Property set containing email properties. **Datatype:** KeyedPropertySet **Required:** No
Mail Info.Server Type	Mail server type that matches the protocol supported by your email server. Enter one of the following values: `pop3` or `imap`. **Datatype:** String **Required:** No
Mail Info.Host	Name of the POP or IMAP server that receives and stores your email. Example: `mail.yourcompany.com`. **Datatype:** String **Maximum Length:** 60 characters **Required:** No
Mail Info.User Name	Address or ID of the Vocera system mailbox that the IT administrator reserved for email sent to Vocera badges (for example, vocerabadge@yourcompany.com). **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Mail Info.Password	Password the Vocera server must use to log in to the Vocera system mailbox. **Use only for update.** **Important:** Your application should restrict passwords to be between 5 and 15 characters. Otherwise, passwords that you set in your VAI application may not be valid for the Vocera Administrator Console and User Console. VAI itself does not restrict the length of passwords. **Datatype:** String **Required:** No
Mail Info.SMTP Host	Name of the server used for outgoing mail. Example: `mail.yourcompany.com`. **Datatype:** String **Maximum Length:** 60 characters **Required:** No
Mail Info.SMTP User Name	User name or address used to log into the outgoing mail server. **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Mail Info.SMTP Password	Password the Vocera server must use to log into the outgoing mail server. **Use only for update.** **Important:** Your application should restrict passwords to be between 5 and 15 characters. Otherwise, passwords that you set in your VAI application may not be valid for the Vocera Administrator Console and User Console. VAI itself does not restrict the length of passwords. **Datatype:** String **Required:** No

Key	Description
Mail Info.SMTP Authentication	If `true`, the mail server requires its subscribers to provide authentication when sending an email message. **Datatype:** boolean **Required:** No
Mail Info.Mail Check Interval	Time interval in seconds that the system waits to check for mail. **Datatype:** int **Required:** No
Mail Info.Default Recipient	Email address to receive warning messages that the Vocera server can issue. The Vocera server sends alert messages to this address to notify the user of significant system events, such as low disk space and cluster failovers. **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Mail Info.Domain Name	Domain name used in email addresses at your site. Entering a value for this field ensures that anyone can reply to email sent from the badge. **Datatype:** String **Maximum Length:** 60 characters **Required:** No
Backup Info	Property set containing Vocera system backup properties. **Datatype:** KeyedPropertySet **Required:** No
Backup Info.Auto Backup Enabled	If `true`, automatic backups are enabled. **Datatype:** boolean **Required:** No
Backup Info.Auto Backup Frequency	Frequency of automatic backups in days. **Datatype:** int **Required:** No
Backup Info.Auto Backup Time	Time of day in milliseconds from midnight on which to start the backup. **Datatype:** long **Required:** No
Backup Info.Max Backup Files	Maximum number of backup files to save. The maximum is the total number of all backup files, regardless of whether they were created automatically or manually. When you exceed the maximum number of files, Vocera deletes the oldest file and saves a new one. **Datatype:** int **Required:** Yes
Logging Info	Property set containing Vocera system logging properties. **Datatype:** KeyedPropertySet **Required:** No
LoggingInfo.Auto Mail Enabled	If `true`, the Vocera Server automatically emails logs to specified recipients. **Datatype:** boolean **Required:** No
LoggingInfo.Auto Mail Only On Restart	If `true`, the Vocera Server automatically emails the most recently closed log file only when the server restarts. Otherwise, the Vocera Server automatically emails the most recently closed log file immediately after the server opens a new one; consequently, the system mails a log file at least once a day. **Datatype:** boolean **Required:** No
LoggingInfo.Auto Mail Recipient 1	First email address for automatic log mailing. **Datatype:** String **Maximum Length:** 60 characters **Required:** No

Key	Description
LoggingInfo.Auto Mail Recipient 2	Second email address for automatic log mailing. **Datatype:** String **Maximum Length:** 60 characters **Required:** No
Department Info	Property set containing speech recognition options for departments. **Datatype:** KeyedPropertySet **Required:** No
Department Info.Rec First Name and Department	If `true`, Vocera recognizes the first name of a user as well as the user's department when someone issues a voice command. Example: `Bill in Housekeeping`. **Datatype:** boolean **Required:** No
Department Info.Rec Full Name and Department	If `true`, Vocera recognizes the full name (both first and last name) as well as the user's department when someone issues a voice command. Example: `Jane Doe in Sales`. **Datatype:** boolean **Required:** No
Freq Dept Info	Property set containing frequently called departments information. **Datatype:** KeyedPropertySet **Required:** No **Since:** 4.3
Freq Dept Info.Frequent Dept Preference Enabled	Indicates whether the use of frequently called departments has been enabled. **Datatype:** boolean **Required:** No **Since:** 4.3
Freq Dept Info.Frequent Dept Adaptation Enabled	Indicates whether adaptation of frequently called departments has been enabled. If `true`, this property enables the gathering of call history data to calculate probabilities for frequently called departments. **Datatype:** boolean **Required:** No **Since:** 4.3
Cluster Info	Property set containing cluster information. **Datatype:** KeyedPropertySet **Required:** No
Cluster Info.Cluster Enabled	If `true`, clustering is enabled. **Datatype:** boolean **Required:** No
Cluster Info.Cluster Members	Property set containing cluster members. **Datatype:** IndexedPropertySet **Required:** No
Cluster Info.Cluster Members.*	The set of properties for each member of the cluster. **Datatype:** KeyedPropertySet **Required:** No
Cluster Info.Cluster Members.*.Host	Numeric IP address of the machine. **Datatype:** String **Maximum Length:** 15 characters **Required:** No
Cluster Info.Cluster Members.*.Description	A brief description of the cluster member to help identify the machine. **Datatype:** String **Maximum Length:** 100 characters **Required:** No
Report Server Info	Property set containing Vocera Report Server information. **Datatype:** KeyedPropertySet **Required:** No

Key	Description
Report Server Info.Report Server IP Address	IP address of the Vocera Report Server. **Datatype:** String **Maximum Length:** 50 characters **Required:** No
Device Info	Property set containing device information. **Datatype:** KeyedPropertySet **Required:** No **Since:** 4.1
Device Info.Status Choices	Property set containing device status values. **Datatype:** IndexedPropertySet **Required:** No **Since:** 4.1
Device Info.Status Choices.*	Represents each device status value. **Datatype:** String **Maximum Length:** 50 characters **Required:** No **Since:** 4.1
Auto Logout Info	Property set that defines whether users will be automatically logged out and the badge will be turned off when the badge is placed in a charger. **Datatype:** KeyedPropertySet **Required:** No **Since:** 4.1
Auto Logout Info.Auto Logout Enabled	If `true`, users will be automatically logged out and the badge will be turned off when the badge is placed in a charger. **Datatype:** boolean **Required:** No **Since:** 4.1
Auto Logout Info.Auto Logout Period	Number of minutes after which an inactive badge user is logged off the Vocera system. When the value is 0 (zero), this feature is disabled. **Datatype:** int **Required:** No **Since:** 4.1
Application Info	Property set that allows administrators to designate information about VAI applications. **Datatype:** KeyedPropertySet **Required:** No **Since:** 4.1 SP3
Application Info.Application Server IP Address	IP address(es) of computers that are allowed to run VAI applications. **Datatype:** String **Maximum Length:** 80 characters **Required:** No **Since:** 4.1 SP3
Handoff Info	Property set that allows administrators to integrate Vocera Server with Vocera Care Transition (formerly Optivox), which allows you to standardize, manage, and monitor hand-offs in healthcare. **Datatype:** KeyedPropertySet **Required:** No **Since:** 4.3
Handoff Info.Handoff Enabled	If `true`, Care Transition integration with Vocera Server is enabled. **Datatype:** boolean **Required:** No **Since:** 4.3

Key	Description
Handoff Info.Handoff Customer ID	Care Transition customer ID. **Datatype:** String **Required:** No **Since:** 4.3
Handoff Info.Handoff Server Phone	The phone number of the Care Transition IVR system. **Datatype:** String **Required:** No **Since:** 4.3
Handoff Info.Handoff Server IP Addr	The IP address of the Care Transition server. **Datatype:** String **Required:** No **Since:** 4.3
Default User	Property set containing default user properties for newly-created users. **Datatype:** KeyedPropertySet **Required:** No
Default User.Password	Default password for new users. The password is case-sensitive. **Used only for update.** **Important:** Your application should restrict passwords to be between 5 and 15 characters. Otherwise, passwords that you set in your VAI application may not be valid for the Vocera Administrator Console and User Console. VAI itself does not restrict the length of passwords. **Datatype:** String **Required:** No
Default User.Verbal Call Announcement	If `true`, users will hear the caller's name announced after the ring tone. **Datatype:** boolean **Required:** No
Default User.Verbal Genie Greeting	If `true`, users will hear a spoken greeting ("Vocera") after pressing the call button. **Datatype:** boolean **Required:** No
Default User.Tonal Genie Greeting	If `true`, users will hear a short tone after pressing the call button. **Datatype:** boolean **Required:** No
Default User.Auto Answer	If `true`, incoming calls are connected immediately, without asking users whether they want to take the call. **Datatype:** boolean **Required:** No
Default User.Auto Who Called	If `true`, users can press the Call button on the badge to play an announcement of the names of callers who unsuccessfully tried to call since the last time the user pressed the Call button, and who left messages. **Datatype:** boolean **Required:** No
Default User.Out Of Range Alert	If `true`, users will hear a warning tone when the badge moves out of the range of the wireless network. **Datatype:** boolean **Required:** No
Default User.Low Battery Alert	If `true`, the badge will warn users whenever the battery needs to be recharged. **Datatype:** boolean **Required:** No
Default User.Auto Logout	If `true`, users will be automatically logged out and the badge will be turned off when the badge is placed in a charger. **Datatype:** boolean **Required:** No

Key	Description
Default User.VMessage Alert	If `true`, users will hear an alert tone when they receive a new voice message. **Datatype:** boolean **Required:** No
Default User.TMessage Alert	If `true`, users will hear an alert tone when they receive a new text message. **Datatype:** boolean **Required:** No
Default User.Disable Alerts In DND	If `true`, all alerts are suppressed when a user's badge is in Do Not Disturb (DND) mode. **Datatype:** boolean **Required:** No
Default User.Play Older Messages First	If `true`, messages are played in the order in which they are received. Otherwise, messages are played in reverse order (newest first). **Datatype:** boolean **Required:** No
Default User.Timestamp Played Messages	If `true`, users will hear the date and time each message was sent when they play messages. **Datatype:** boolean **Required:** No
Default User.Fast Call Setup	If `true`, a call is connected as soon as the recipient accepts it. Otherwise, the Genie always completes the call announcement before connecting the call. **Datatype:** boolean **Required:** No
Default User.VMessage Reminder	If `true`, users will hear a tone every 10 minutes until they retrieve new voice messages. **Datatype:** boolean **Required:** No
Default User.TMessage Reminder	If `true`, users will hear a tone every 10 minutes until they retrieve new text messages. **Datatype:** boolean **Required:** No
Default User.DND Reminder	If `true`, users will hear a tone every 10 minutes while their badges are in Do Not Disturb (DND) mode. **Datatype:** boolean **Required:** No
Default User.Enable Pages	If `true`, users can receive numeric pages. Otherwise, pages are disabled for new users. **Datatype:** boolean **Required:** No
Default User.Announce Through Speaker	If `true`, Vocera plays incoming call and message announcements through the badge speaker when a headset is plugged into the user's badge. Otherwise, both announcements and actual calls or messages are played through the headset. **Datatype:** boolean **Required:** No
Default User.Block Voice Messages	If `true`, Vocera suppresses notifications when a user receives a message. However, the user may still hear a voice message alert tone (if the Voice Message Alert option is selected), and a telephone icon appears on the badge display when the user has unplayed voice messages. **Datatype:** boolean **Required:** No **Since:** 4.1

Key	Description
Default User.Enable Genie Access From Phone	If `true`, it enables the ability to access the Genie from a telephone to perform Vocera functions other than basic calling. The number of users that can use the phone access feature is controlled by your Vocera license. Only users that have been enabled to use the phone access feature can take advantage of this feature. **Datatype:** boolean **Required:** No **Since:** 4.1
Default User.Ring Tone	One of the available ring tones, for example, `Ring-Tone-01`, `Ring-Tone-02`, and so on. When a user receives a call on his badge, the specified ring tone is used. **Datatype:** String **Required:** No
Default User.Genie Persona	One of the available Genie names, for example, `Mark` or `Jean`. The Genie is the voice that prompts users when they interact with the Vocera system. **Datatype:** String **Required:** No
Default Site	Property set containing default site properties. **Datatype:** KeyedPropertySet **Required:** No
Default Site.Telephony Info	Property set containing default telephony properties. **Datatype:** KeyedPropertySet **Required:** No
Default Site.Telephony Info.Telephony Enabled	If `true`, telephony features are enabled for the site. **Datatype:** boolean **Required:** No
Default Site.Telephony Info.Telephony Interface Type	Type of telephony interface. Enter `IP`, `Digital`, or `Analog`. **Datatype:** String **Required:** No
Default Site.Telephony Info.Telephony # of Lines	Number of telephone lines. **Datatype:** int **Required:** Yes (if telephony is enabled for the site)
Default Site.Telephony Info.Telephony Protocol	Signaling protocol that your PBX uses at the network layer. For IP PBX integration, enter the following value: `SIP Version 2.0`. For Digital PBX integration, enter one of the following values: `ISDN PRI`, `EURO ISDN PRI`, or `Wink Start`. DO NOT update this property if Telephony Interface Type = `Analog`. **Datatype:** String **Required:** No
Default Site.Telephony Info.Telephony ISDN Protocol	ISDN protocol used by your PBX. Enter one of the following values: `NI2`, `DMS`, `5ESS`, `4ESS`, `NT1`, `CTR4`, `QTE`, `NE1`, or `QNT`. **Datatype:** String **Required:** No
Default Site.Telephony Info.Telephony Framing	Framing that your PBX uses at the physical layer. Enter one of the following values: `ESF`, `D4`, or `CEPT1`. Update this property only if Telephony Interface Type = `Digital`. **Datatype:** String **Required:** No
Default Site.Telephony Info.Telephony Line Code	Line code that your PBX uses at the physical layer. Enter one of the following values: `B8ZS`, `AMI`, or `HDB3`. Update this property only if Telephony Interface Type = `Digital`. **Datatype:** String **Required:** No

Key	Description
Default Site.Telephony Info.Area Code	Area code of the region in which the Vocera server is installed. **Datatype:** String **Maximum Length:** 10 characters **Required:** Yes (if telephony is enabled for the site)
Default Site.Telephony Info.Local Access	Sequence of numbers you use to get an outside line. For example, a PBX might require you to dial a 0 or a 9 or an 8 to get an outside line. By default, Vocera prepends this access code to any number within the local area code. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Default Site.Telephony Info.Long Distance Access	Sequence of numbers you enter before placing a long distance call. For example, a PBX system might require you to dial a 9 to get an outside line and then dial a 1 before a long-distance telephone number. In this situation, enter `91`. By default, Vocera prepends this access code to any number that includes an area code that is not the local area code. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Default Site.Telephony Info.System Phone Number	Area code and phone number of the DID line or hunt group you set up for the Vocera system. To use this number with numeric pagers, enter an asterisk after the last digit of the phone number. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Default Site.Telephony Info.Direct Access Phone Number	Area code and phone number of the DID line you set up for specially licensed user access to the Vocera system. If you have not obtained Vocera Access Anywhere user licenses or you are not using ISDN or SIP signaling protocol, this property should not be updated. **Datatype:** String **Maximum Length:** 75 characters **Required:** No **Since:** 4.1
Default Site.Telephony Info.Voice Mail Access	Sequence of numbers you enter to access the company's voice mail system. A typical entry includes X, then the sequence of digits that you dial to get into the voicemail system from an internal phone, and possibly special dialing characters such as the * or # to indicate the end of the sequence. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Default Site.Telephony Info.Seven Digit Dialing	If `true`, the area code is omitted from the dialing sequence for a local call. **Datatype:** boolean **Required:** No
Default Site.Telephony Info.PIN Setup	Template for adding a PIN to a dialing sequence for long distance calls. A PIN template can include digits, special characters, and PIN macros. **Datatype:** String **Maximum Length:** 75 characters **Required:** No

Key	Description
Default Site.Telephony Info.Default PIN	The default PIN for long distance calls. If a telephony PIN is not specified in the user's profile and the user does not belong to a department group that has a PIN, then the site PIN is used. **Datatype:** String **Maximum Length:** 75 characters **Required:** No
Default Site.Telephony Info.Telephony Extension Length	Specify the number of digits in an extension, or 0 (zero) to allow variable length extensions. **Datatype:** int **Required:** No
Default Site.Telephony Info.Access Code Info	By default, numbers in the local area code use the Default Local Access Code and all others use the Default Long-Distance Access Code. This property set contains telephone numbers that are exceptions to the access code policy. Each member of the indexed set is itself a property set. **Datatype:** IndexedPropertySet **Required:** No
Default Site.Telephony Info.Access Code Info.*	Represents each defined access code exception in the property set. **Datatype:** KeyedPropertySet **Required:** No
Default Site.Telephony Info.Access Code Info.*.Number Range	Property set that defines a number range to use for direct inward dialing. **Datatype:** KeyedPropertySet **Required:** No
Default Site.Telephony Info.Access Code Info.*.Number Range.Area Code	Area code for which the exception is defined. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Default Site.Telephony Info.Access Code Info.*.Number Range.Match Type	Type of number range to match. Enter one of the following values: `All`, `Starts With`, or `Range`. **Datatype:** String **Required:** No
Default Site.Telephony Info.Access Code Info.*.Number Range.Starts With	Sequence of characters to match. Used when Match Type = `Starts With`. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Default Site.Telephony Info.Access Code Info.*.Number Range.From	Start of range to match. Used when Match Type = `Range`. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Default Site.Telephony Info.Access Code Info.*.Number Range.To	End of range to match. Used when Match Type = `Range`. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Default Site.Telephony Info.Access Code Info.*.Access Code	Access code that the specified area code requires. **Datatype:** String **Maximum Length:** 10 characters **Required:** Yes (for each access code exception)
Default Site.Telephony Info.Toll Info	By default, numbers in the local area code are considered toll-free, and others are considered to require toll-call permissions. This property set contains telephone numbers that are exceptions to the toll-call policy. **Datatype:** IndexedPropertySet **Required:** No

Key	Description
Default Site.Telephony Info.Toll Info.*	Represents each defined toll info exception. Each keyed set specifies an area code and phone numbers that can be called even by users who do not have toll-call permissions granted. **Datatype:** KeyedPropertySet **Required:** No
Default Site.Telephony Info.Toll Info.*.Number Range	Property set that defines the number range for a toll info exception. **Datatype:** KeyedPropertySet **Required:** No
Default Site.Telephony Info.Toll Info.*.Number Range.Area Code	Area code for which the exception is defined. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Default Site.Telephony Info.Toll Info.*.Number Range.Match Type	Type of number range to match. Enter one of the following values: `All`, `Starts With`, or `Range`. **Datatype:** String **Required:** No
Default Site.Telephony Info.Toll Info.*.Number Range.Starts With	Sequence of characters to match. Used when Match Type = `Starts With`. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Default Site.Telephony Info.Toll Info.*.Number Range.From	Start of range to match. Used when Match Type = `Range`. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Default Site.Telephony Info.Toll Info.*.Number Range.To	End of range to match. Used when Match Type = `Range`. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Default Site.Telephony Info.Toll Info.*.Toll Free	If `true`, the area code is toll free. **Datatype:** boolean **Required:** No
Default Site.Telephony Info.Paging Info	Specifies properties for interacting with pagers. **Datatype:** KeyedPropertySet **Required:** No
Default Site.Telephony Info.Paging Info.Pager Number Length	Specify the number of digits in a pager number, or 0 (zero) to allow variable length numbers. **Datatype:** int **Required:** No
Default Site.Telephony Info.Paging Info.Outside Page Setup	Template that determines how Vocera formats the string passed to a pager outside the Vocera system. The default value of this property is %N;%V%D. For more information, see the *Vocera Telephony Configuration Guide*. **Datatype:** String **Required:** No
Default Site.Telephony Info.Paging Info.Inside Page Setup	Template that determines how Vocera formats the string passed to a pager inside the Vocera system. The default value of this property is %N;%V%D. For more information, see the *Vocera Telephony Configuration Guide*. **Datatype:** String **Required:** No
Default Site.Telephony Info.Paging Info.Outside Page Setup for DialIn	Template that determines how Vocera formats the string passed to an outside pager by a person calling into the Vocera hunt group or DID number. The default value of this property is %N;%X. For more information, see the *Vocera Telephony Configuration Guide*. **Datatype:** String **Required:** No

Key	Description
Default Site.Telephony Info.Paging Info.Inside Page Setup for DialIn	Template that determines how Vocera formats the string passed to an inside pager by a person calling into the Vocera hunt group or DID number. The default value of this property is %N;%X. For more information, see the *Vocera Telephony Configuration Guide*. **Datatype:** String **Required:** No
Default Site.Telephony Info.DID Info	Property set containing direct inward dialing (DID) information. **Datatype:** IndexedPropertySet **Required:** No
Default Site.Telephony Info.DID Info.*	Represents each defined range of direct inward dialing (DID) numbers. Each keyed set specifies a prefix and the range of phone numbers to use for direct inward dialing. **Datatype:** KeyedPropertySet **Required:** No
Default Site.Telephony Info.DID Info.*.Number Range	Property set that defines a number range to use for direct inward dialing. **Datatype:** KeyedPropertySet **Required:** No
Default Site.Telephony Info.DID Info.*.Number Range.Match Type	Type of number range to match. Enter one of the following values: `All`, `Starts With`, or `Range`. **Datatype:** String **Required:** No
Default Site.Telephony Info.DID Info.*.Number Range.Starts With	Sequence of characters to match. Used when Match Type = `Starts With`. **Datatype:** String **Maximum Length:** 10 characters **Required:** No
Default Site.Telephony Info.DID Info.*.Number Range.From	Start of range to match. Used when Match Type = `Range`. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Default Site.Telephony Info.DID Info.*.Number Range.To	End of range to match. Used when Match Type = `Range`. **Datatype:** String **Maximum Length:** 20 characters **Required:** No
Default Site.Telephony Info.DID Info.*.Prefix	Area code and prefix assigned to the range. For example, if the local area code of the PBX is 408, and the corporate prefix for all extensions is 790, you typically enter `(408)-790`. In some situations, your PBX administrator may assign a different prefix for you to use. **Datatype:** String **Maximum Length:** 50 characters **Required:** Yes
Default Site.Telephony Info.Dynamic Phone Info	Property set that specifies a range of dynamic phone extensions. This allows you to configure Vocera to supply telephone extensions on demand to users who need them. **Datatype:** KeyedPropertySet **Required:** No
Default Site.Telephony Info.Dynamic Phone Info.Enabled	If `true`, dynamic extensions are enabled. **Datatype:** boolean **Required:** No
Default Site.Telephony Info.Dynamic Phone Info.First	Specifies the first dynamic extension in the range. **Datatype:** String **Maximum Length:** 7 characters **Required:** No

Key	Description
Default Site.Telephony Info.Dynamic Phone Info.Last	Specifies the last dynamic extension in the range. The Last value must be greater than the First value. **Datatype:** String **Maximum Length:** 7 characters **Required:** No
Default Site.Telephony Info.Dynamic Phone Info.Lifetime	Specifies the lifetime, in days or hours, of the assignment of dynamic extensions. Enter 0 (zero) to make the extensions permanent. **Datatype:** int **Required:** No
Default Site.Telephony Info.Dynamic Phone Info.Hours	Specifies whether the lifetime of dynamic extensions is measured in hours (`true`) or days (`false`). **Datatype:** boolean **Required:** No **Since:** 4.1
Default Site.Telephony Info.Dynamic Phone Info.Last Allocated	The last allocated dynamic extension. **Cannot be updated.** **Datatype:** String
Default Site.Telephony Info.Shared Server Info	Property set that contains information needed to allow multiple sites to share a Telephony server. **Datatype:** IndexedPropertySet **Required:** No
Default Site.Telephony Info.Shared Server Info.*	Represents each defined site that shares this Telephony server. Each keyed set specifies the site, hunt group number, reserved range of lines for incoming calls, and the tie line prefix. **Datatype:** KeyedPropertySet **Required:** No
Default Site.Telephony Info.Shared Server Info.*.Site	Principal site for which Telephony is enabled. **Datatype:** Site object
Default Site.Telephony Info.Shared Server Info.*.System Phone Number	Area code and phone number of the DID line or hunt group you set up for the Vocera system. **Datatype:** String
Default Site.Telephony Info.Shared Server Info.*.First Reserved Line No	First of the reserved lines for incoming calls. **Datatype:** String
Default Site.Telephony Info.Shared Server Info.*.Reserved Line Count	Number of reserved lines for incoming calls. **Datatype:** int
Default Site.Telephony Info.Shared Server Info.*.Extension Prefix	Prefix of the dial string used to place calls through the tie line to the selected site that is sharing the principal's telephony server. Alternatively, this field could also be used to specify a prefix for Direct Inward Dialing (DID) numbers at the selected site. **Datatype:** String **Since:** 4.1
Default Site.Telephony Info.Call Signaling Address	The IP address of your IP PBX or VoIP gateway. By default, port 5060 is used. If you need to change the port, enter the call signaling address in the form *IP_Address:Port*. **Datatype:** String **Maximum Length:** 75 characters **Since:** 4.3
Default Site.Telephony Info.Cisco Info	Property set containing default Cisco integration properties. **Datatype:** KeyedPropertySet **Required:** No **Since:** 4.3
Default Site.Telephony Info.Cisco Info.Enabled	**Datatype:** boolen **Required:** No **Since:** 4.3
Default Site.Telephony Info.Cisco Info.Extension Mobility Enabled	**Datatype:** boolean **Required:** No **Since:** 4.3

Key	Description
Default Site.Telephony Info.Cisco Info.Phone	The voice access number for CUCM. This number should match the route pattern/number for the Vocera SIP trunk. You can find route patterns in CUCM Console by choosing Call Routing > Route/Hunt > Route Pattern. **Datatype:** String **Required:** No **Since:** 4.3
Default Site.Telephony Info.Cisco Info.First Line	The first phone line used for the internal range of Vocera lines. **Datatype:** String **Required:** No **Since:** 4.3
Default Site.Telephony Info.Cisco Info.Last Line	The last phone line used for the internal range of Vocera lines. **Datatype:** String **Required:** No **Since:** 4.3
Default Site.Telephony Info.Cisco Info.IP Address	The IP address of the CUCM in dotted-decimal notation (for example, 192.168.15.10). **Datatype:** String **Required:** No **Since:** 4.3
Default Site.Telephony Info.Cisco Info.User Name	The Vocera application user ID for CUCM. **Datatype:** String **Required:** No **Since:** 4.3
Default Site.Telephony Info.Cisco Info.Password	The Vocera application user ID for CUCM. **Datatype:** String **Required:** No **Since:** 4.3